А.О. Хруцький

Основи розробки проектних підсистем на базі SolidWorks API

Кривий Ріг
Видавничий центр ДВНЗ «КНУ»
2016

УДК 004.415.2.041 (075.8)
ББК 30.2-5-05я7
Х–95

Рецензенти: **К. С. Заболотній**, д–р. техн. наук, професор, завідувач кафедрою гірничих машин та інжинірингу Національного гірничого університету;
Р. Д. Іскович-Лотоцький, д–р. техн. наук, професор, завідувач. кафедрою галузевого машинобудування Вінницького національного технічного університету;
А. К. Семенченко, д–р. техн. наук, професор, завідувач. кафедрою гірничих машин та мехатронних систем машинобудування Донбаського національного технічного університету.

Рекомендовано до друку вченою радою ДВНЗ «Криворізький національний університет» як навчальний посібник для студентів вищих навчальних закладів (протокол № 10 від 23.06.2016 р).

Х–95 **Хруцький А.О.**
Основи розробки проектних процедур на базі SolidWorks API: навч. посіб. для студ. вищих навч. закладів. – Кривий Ріг: Видавничий центр ДВНЗ «КНУ», 2016. – 303 с.

ISBN 978-966-132-036-8

Посібник присвячено основам розробки підсистем САПР на базі SolidWorks API. Розглянуто існуючі підходи до рішення деяких завдань і проблем інженерної діяльності із автоматизації різних етапів проектно-конструкторських робіт з використанням програмного інтерфейсу та вбудованої мови програмування VBA.

Посібник призначено для підготовки студентів вищих навчальних закладів машинобудівних спеціальностей, які мають досвід роботи з САПР SolidWorks, а також може бути корисним інженерно-технічним працівникам та конструкторам.

УДК 004.415.2.041 (075.8)
ББК 30.2-5-05я7

ISBN 978-966-132-036-8

Зміст

Передмова

Сучасна концепція застосування комп'ютерних технологій проектування ґрунтується на системному підході, тобто на створенні та впровадженні систем, в яких спілкування конструктора з ЕОМ відбувається в режимі діалогу. Такі системи отримали назву «системи автоматизованого проектування і розрахунків» або скорочено САПР.

САПР розв'язують весь комплекс завдань розроблення конструкторської та технологічної документації. Це досягається через об'єднання сучасних технічних засобів і математичного забезпечення, параметри і характеристики яких обираються з максимальним урахуванням особливостей завдань проектно-конструкторського процесу.

Сучасний пакет САПР SolidWorks надає інженерам-конструкторам широкий спектр функціональних можливостей, що дозволяють створювати і модифікувати як двовимірну, так і тривимірну геометрію. Проте при усьому багатстві функцій SolidWorks, працюючи через стандартний інтерфейс, призначений для користувача, інженер має доступ приблизно до 60–70 % функціональних можливостей, закладених розробниками у цю систему. Крім того, пакет SolidWorks як універсальна система, призначений і для загального машинобудування, і для цілої низки інших галузей.

Для отримання повного доступу до усіх функціональних можливостей САПР SolidWorks створено програмний інтерфейс (API).

Application Programming Interface (API) SolidWorks – це інтерфейс, що дозволяє створювати прикладне програмне забезпечення для САПР SolidWorks, що автоматизує різні етапи проектно-конструкторських робіт. Використання SolidWorks API – найбільш дешевий і зручний спосіб для ідеального налаштування SolidWorks для розв'язання різноманітних завдань.

API містить сотні функцій розроблених за допомогою мов програмування Microsoft Visual Basic for Appliction (VBA), Microsoft Visual C, C++, які можна використовувати у програмах Microsoft Excel, Word, Access, SolidWorks та ін. Ці функції надають інженерові повний прямий доступ до функціональних можливостей SolidWorks.

Динамічні бібліотеки типів і констант, що відповідають за роботу API, автоматично інсталюються на комп'ютер при установці програми. Таким чином, кожне робоче місце САПР SolidWorks за замовчуванням оснащено інтерфейсом прикладного програмування, що дає розробникам широке поле для діяльності.

Нині налічується понад тисяча програм, що використовують SolidWorks API для розв'язання низки спеціальних завдань. Така велика кількість додатків є важливою характерною особливістю пакету SolidWorks і позитивним чином виокремлює його зі списку інших систем, наявних сьогодні на ринку.

У посібнику розглянуто структуру й основні принципи побудови підсистем САПР на базі SolidWorks API. Наведено наявні підходи до розв'язання деяких завдань і проблем інженерної діяльності, а також формування уміння алгоритмізації розв'язання окремих конструкторсько-проектних завдань.

Призначенням пропонованого навчального посібника є надання необхідних відомостей про загальні принципи побудови прикладного програмного забезпечення для САПР SolidWorks, що автоматизує різні етапи проектно-конструкторських робіт з використанням програмного інтерфейсу та вбудованої мови програмування VBA.

Загалом посібник адресовано студентам технічних спеціальностей, які не мають намір професійно займатися програмуванням, однак розуміють необхідність доведення методик проектування, які характерні для їх спеціалізації, до рівня близького до систем автоматизованого проектування (САПР). Опанування матеріалу посібника надасть змогу читачеві як самостійно створювати такі САПР-додатки, так і грамотно формулювати технічні завдання для професійних програмістів.

1. Основи САПР. САПР як об'єкт проектування

1.1. Мета і завдання створення САПР

Основне завдання створення САПР – автоматизація робіт на стадіях проектування і підготовки виробництва у рамках життєвого циклу промислових виробів.

Основна мета створення САПР – підвищення ефективності праці інженерів, включаючи:

– скорочення трудомісткості і часу проектування;
– скорочення собівартості проектування і виготовлення, зменшення витрат на експлуатацію;
– підвищення якості і техніко-економічного рівня результатів проектування;
– скорочення витрат на натурне моделювання і випробування.

Досягнення поставленої мети забезпечується такими шляхами:

– автоматизація оформлення документації;
– інформаційна підтримка і автоматизація процесу ухвалення рішень;
– уніфікація проектних рішень і процесів проектування;
– повторне використання проектних рішень, даних і напрацювань;
– стратегічне проектування;
– заміна натурних випробувань і макетування математичним моделюванням.

Використання САПР при проектуванні дає такі переваги [1, 7]:

1. Швидке виконання креслень. Прискорення процесу проектування загалом дозволяє у стислі терміни випускати продукцію і швидше реагувати на зміну ринкових вимог.

2. Підвищення точності виконання. На кресленнях, побудованих за допомогою системи САПР, місце будь-якої точки визначене точно, не залежно від масштабу креслення, що дозволяє збільшувати або зменшувати будь-яку частину цього креслення у будь-яке число разів.

3. Підвищення якості. Провідні САПР мають інструменти для пошуку і виправлення помилок.

4. Можливість багаторазового використання креслень для проектування, коли до складу креслення входить низка компонентів, що мають однакову форму.

5. Прискорення розрахунків та аналізу при проектуванні. Потужні засоби комп'ютерного моделювання, наприклад, метод кінцевих елементів, звільняють конструктора від використання традиційних форм і дозволяють проектувати нестандартні геометричні форми.

6. Зниження витрат на оновлення виробів. Засоби аналізу й імітації у САПР дозволяють суттєво скоротити витрати часу і грошей на тестування та удосконалення прототипів, які є дорогими етапами процесу проектування.

7. Високий рівень складності проектування. Можливість проектування нестандартних геометричних форм, які швидко оптимізуються.

1.2. Основні визначення і поняття

Розглянемо основні визначення і поняття [1, 5, 9].

Проектування – процес визначення характеристик моделі або її частин, описаних у формі, придатній для реалізації виробу.

Автоматизація проектування – систематичне застосування ЕОМ у процесі проектування при науково обґрунтованому розподілі функцій між проектувальником та ЕОМ і науково обґрунтованому виборі методів машинного розв'язання завдань.

Науково обґрунтований розподіл функцій між людиною і ЕОМ припускає, що людина повинна розв'язувати завдання, що носять творчий характер, а ЕОМ – завдання, розв'язання яких піддається алгоритмізації.

Неавтоматизоване проектування – проектування, що повністю здійснюється людиною без використання комп'ютерної техніки.

Автоматизоване проектування – проектування, що здійснюється людиною при взаємодії з комп'ютером.

Автоматичне проектування – проектування, при якому усі перетворення опису об'єкта й алгоритму його функціонування здійснюються комп'ютером без участі людини.

Розглянемо основні наявні технології у проектуванні [6, 9].

CAD (computer-aided design) – ***автоматизоване проектування*** – технологія, що полягає у використанні комп'ютерних систем для полегшення створення, зміни, аналізу й оптимізації

проектів. Будь-яка програма, що працює з комп'ютерною графікою належить до систем автоматизованого проектування. Основна функція CAD – визначення геометрії конструкції (деталі механізму, архітектурні елементи, електронні схеми та ін.). Це одна з переваг CAD, що дозволяє економити час і скорочувати кількість помилок, пов'язаних з необхідністю визначати геометрію конструкції з нуля кожного разу при розрахунках.

CAM (computer-aided manufacturing) – ***автоматизоване виробництво*** – технологія, що полягає у використанні комп'ютерних систем для планування, управління і контролю операцій виробництва через прямий або непрямий інтерфейс з виробничими ресурсами підприємства. Одним з підходів до автоматизації виробництва є числове програмне управління верстатами, що полягає у використанні запрограмованих команд для управління верстатами, які можуть шліфувати, різати, фрезерувати та іншими способами перетворювати заготівки на готові деталі.

CAE (computer-aided engineering) – автоматизоване конструювання – технологія, що полягає у використанні комп'ютерних систем для аналізу геометрії, моделювання і вивчення поведінки продукту задля удосконалення та оптимізації його конструкції. Програми для кінематичних розрахунків, наприклад, здатні визначати траєкторії руху і швидкості ланок у механізмах. Програми динамічного аналізу можуть використовуватися для визначення навантажень і зміщень у складних пристроях.

CIM (computer-integrated manufacturing) – ***комп'ютеризоване інтегроване виробництво*** – нова технологія, що намагається поєднати «острівці автоматизації» разом і перетворити їх на безперебійно та ефективно працюючу систему. CIM використовує комп'ютерну базу даних для ефективного управління усім підприємством, зокрема бухгалтерією, плануванням, доставкою та іншими завданнями, а не тільки проектуванням і виробництвом, які охоплювалися системами CAD, CAM і CAE.

PDM (Product Data Management) – ***управління виробничою інформацією*** – інструментальний засіб, який допомагає адміністраторам, інженерам та конструкторам управляти як даними, так і процесами розроблення виробу на сучасних виробничих підприємствах або групі суміжних підприємств.

CALS (Computer Aided Logistic Systems) – ***автоматизовані логістичні системи*** – технологія комплексної комп'ютеризації сфер промислового виробництва, що передбачає зберігання, оброблення і передавання інформації у комп'ютерних середовищах, оперативний доступ до даних у потрібний час і у потрібному місці. Застосування CALS-систем дозволяє істотно скоротити обсяги проектних робіт, оскільки описи багатьох складових частин устаткування, машин і систем, що проектувалися раніше, зберігаються у базах цих мережевих серверів, доступних будь-якому користувачеві технології CALS.

1.3. Класифікація САПР

САПР класифікуються [1, 8, 9]:

1) за типом об'єкта проектування:
- вироби машинобудування;
- вироби приладобудування;
- технологічні процеси в машино- і приладобудуванні;
- об'єкти будівництва;
- технологічні об'єкти у будівництві;
- програмні вироби;
- організаційні системи.

2) за складністю об'єкта проектування:
- прості об'єкти (число складових частин до 10^2);
- об'єкти середньої складності (число складових частин від 10^2 до 10^3);
- складні об'єкти (число складових частин від 10^3 до 10^4);
- дуже складні об'єкти (число складових частин від 10^4 до 10^6);
- об'єкти дуже високої складності (число складових частин понад 10^6).

3) за рівнем автоматизації проектування:
- низькоавтоматизовані (рівень автоматизації менше 25 %);
- середньоавтоматизовані (рівень автоматизації 25–50 %);
- високоавтоматизовані (рівень автоматизації понад 50 %).

4) за комплексністю автоматизації проектування:
- одноетапні (виконує один етап проектування);
- багатоетапні (виконує декілька етапів проектування);
- комплексні (виконує усі етапи проектування).

5) за характером документів, що випускаються:

– текстові – виконують тільки текстові документи на паперовій стрічці або аркуші;
– текстові і графічні – текстові і графічні документи на паперовій стрічці або аркуші;
– на електронних носіях – документи на магнітних, лазерних дисках, флеш-носіях;
– комбіновані.

6) за кількістю документів, що випускаються:

– малої продуктивності (випускає документів за рік до 10^5);
– середньої продуктивності (випускає документів за рік від 10^5 до 10^6);
– високої продуктивності (випускає документів за рік понад 10^6).

7) за кількістю рівнів:

– однорівнева САПР, побудована на основі комп'ютера середнього або високого класу зі штатним набором периферійних пристроїв, який може бути доповнений засобами обробленя графічної інформації;
– дворівнева САПР, побудована на основі комп'ютера середнього або високого класу й одного або декількох автоматизованих робочих місць (АРМ), що включають міні-ЕОМ;
– трирівнева САПР, побудована на основі ЕОМ високого класу, АРМ і периферійного програмно-керованого устаткування.

8) за характером базової підсистеми:

– САПР на базі підсистеми машинної графіки і геометричного моделювання орієнтовані на додатки, де основною процедурою проектування є конструювання, тобто визначення просторових форм і взаємного розташування об'єктів. До цієї групи систем належать більшість графічних ядер САПР у галузі машинобудування. Уніфіковані графічні ядра, уживані більш ніж в одній САПР, − це ядра Parasolid фірми EDS Unigraphics і ACIS фірми Intergraph;
– САПР на базі СУБД орієнтовані на додатки, в яких при порівняно нескладних математичних розрахунках переробляється великий обсяг даних. Переважно використовуються у техніко-економічних розрахунках, наприклад, при проектуванні бізнес-планів, при проектуванні об'єктів, подібних до щитів управління в системах автоматики;

– САПР на базі конкретного прикладного пакету у вигляді автономно використовуваних програмно-методичних комплексів, наприклад, імітаційного моделювання виробничих процесів, розрахунку міцності за методом кінцевих елементів, синтезу та аналізу систем автоматичного управління та ін. Часто такі САПР належать до систем CAE. Прикладами можуть служити програми логічного проектування на базі мови VHDL, математичні пакети типу MathCAD;
– комплексні (інтегровані) САПР, які складаються з сукупності підсистем попередніх видів. Характерними прикладами комплексних САПР є CAE/CAD/CAM-системи в машинобудуванні, наприклад SolidWorks. Для управління такими складними системами застосовують спеціалізовані системні середовища.

1.4. Загальносистемні принципи САПР

При створенні й розвитку САПР застосовуються такі загальносистемні принципи [1, 4]:

– ***принцип включення*** полягає у тому, що вимоги до створення, функціонування і розвитку САПР визначаються з боку складнішої системи, наприклад, галузі або проектної організації;
– ***принцип системної єдності*** полягає у тому, що на всіх стадіях створення, функціонування і розвитку САПР цілісність системи повинна забезпечуватися зв'язками між підсистемами САПР, а також функціонуванням підсистеми управління САПР;
– ***принцип розвитку*** вимагає, щоб САПР розроблялася і функціонувала як система, що розвивається, з можливістю вдосконалення компонентів;
– ***принцип комплексності*** вимагає, щоб у САПР забезпечувалася зв'язність проектування окремих елементів і усього об'єкта;
– ***принцип інформаційної єдності*** полягає у тому, що компоненти САПР повинні використовувати терміни, символи, умовні позначення, проблемно-орієнтовані мови програмування і способи подання інформації, встановлені у галузях відповідними нормативними документами;

– ***принцип сумісності*** полягає у тому, що мови, символи, коди і компоненти САПР мають бути погоджені;
– ***принцип інваріантності*** полягає у тому, що підсистеми і компоненти САПР мають бути за можливості універсальними.

1.5. Загальні вимоги, що висуваються до САПР

Загальні вимоги до САПР розрізняються на [1, 4]:

1) системні вимоги, що відносяться до усієї САПР:
– ефективність – САПР повинна забезпечувати ефективне виконання усієї сукупності функцій автоматизованого проектування задля отримання досить якісних рішень і проектної документації у прийнятні терміни;
– універсальність – забезпечення виконання усього процесу проектування без перебудови САПР;
– сумісність – засоби, що входять у САПР, мають бути технічно, інформаційно, програмно й експлуатаційно сумісні;
– гнучкість і відкритість – САПР повинна забезпечувати можливість своєї перебудови у досить широких межах і дозволяти заміну застарілих засобів, модернізацію і розширення складу;
– надійність – САПР повинна забезпечувати надійність роботи для нормального функціонування упродовж усього циклу проектування;
– точність (достовірність) – САПР повинна забезпечити необхідний рівень точності (достовірності) рішень, що приймаються, і даних. Точність залежить від точності методів округлення, розрядності, збоїв в устаткуванні, захищеності від зовнішніх дій;
– захищеність – САПР має бути захищена від зовнішніх дій (перешкод, збоїв у системі живлення, некомпетентного і несанкціонованого втручання);
– можливість одночасної роботи досить широкого кола користувачів – САПР повинна бути системою колективного користування для досить великої кількості фахівців (розробників САПР, проектувальників та ін.);
– прийнятна вартість − вартість САПР має бути така, щоб створені на її базі проекти забезпечили прийнятний економічний ефект.

2) функціональні вимоги, що стосуються організації роботи САПР:
- реалізація математичних моделей, ухвалення рішень і проектних процедур;
- наявність систем пошуку даних;
- забезпечення наочності;
- можливість роботи як в пакетному, так і в діалоговому режимі;
- документування результатів проектування;
- видача результатів на технологічне устаткування.

3) технічні вимоги, що стосуються програмної реалізації та обладнання, на якому працює САПР:
- продуктивність;
- швидкодія;
- пропускна спроможність;
- розрядність;
- система кодування інформації;
- місткість оперативної пам'яті;
- види носіїв даних.

4) організаційно-експлуатаційні вимоги, що стосуються комфортної та ефективної роботи людини із САПР:
- ергономіка і технічна естетика;
- безпека персоналу при експлуатації (вимоги електробезпеки і пожежної безпеки);
- підготовка персоналу (рівень навченості і кваліфікації персоналу);
- централізоване технічне обслуговування.

1.6. Будова САПР

Складовими структурними частинами САПР є підсистеми [1, 4, 8, 9], що мають усі властивості систем і створені як самостійні системи. Кожна підсистема – це виокремлена за деякими ознаками частина САПР, що забезпечує виконання деяких функціонально-закінчених послідовностей проектних завдань з отриманням відповідних проектних рішень і проектних документів.

За призначенням підсистеми САПР поділяють на два види: проектуючі та обслуговуючі.

Проектуючі підсистеми безпосередньо виконують проектні процедури, що реалізують певний етап проектування або групу пов'язаних проектних завдань, наприклад:

– підсистеми геометричного тривимірного моделювання механічних об'єктів;
– підсистема виготовлення конструкторської документації;
– підсистема трасування з'єднань у електронних платах;
– підсистема аналізу схемотехніки;
– підсистема компонування машини;
– підсистема проектування складальних одиниць;
– підсистема проектування деталей;
– підсистема проектування схеми управління;
– підсистема технологічного проектування.

Залежно від відношення до об'єкта проектування розрізняють два види проектуючих підсистем:

– об'єктно-орієнтовані (об'єктні), що використовують проектні процедури й операції, безпосередньо пов'язані з конкретним типом об'єктів проектування;
– незалежні (інваріантні) – виконують уніфіковані проектні процедури й операції для багатьох типів об'єктів проектування.

До об'єктних підсистем відносять підсистеми, що виконують одну або декілька проектних процедур або операцій, безпосередньо залежних від конкретного об'єкта проектування, наприклад:

– підсистема проектування технологічних систем;
– підсистема моделювання динаміки конструкції, що проектується.

До інваріантних підсистем відносять підсистеми, що виконують уніфіковані проектні процедури й операції, наприклад:

– підсистема розрахунків деталей машин;
– підсистема розрахунків режимів різання;
– підсистема розрахунку техніко-економічних показників.

Обслуговуючі підсистеми забезпечують функціонування проектуючих підсистем, їх сукупність часто називають системним середовищем (чи оболонкою) САПР.

Типовими обслуговуючими підсистемами є:

– підсистеми управління проектними даними;
– підсистеми розроблення і супроводу програмного забезпечення CASE (Computer Aided Software Engineering);

– навчальні підсистеми для опанування користувачами технологій, реалізованих в САПР;
– підсистеми графічного введення / виводу;
– система управління базами даних (СУБД);
– підсистема графічного відображення об'єктів проектування;
– підсистема документування;
– підсистема інформаційного пошуку.

Загальна схема автоматизованої проектної процедури наведена на рис. 1.1.

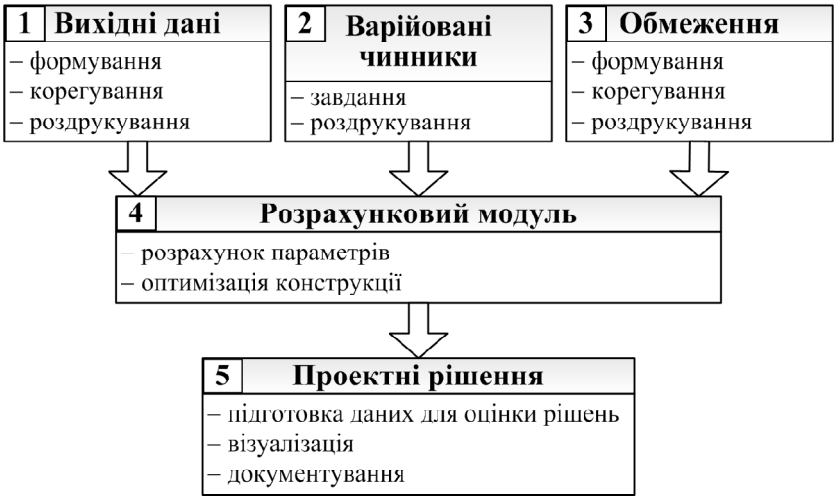


Рис. 1.1. Схема програмного забезпечення проектної процедури САПР

Розглянемо наведені у процедурі блоки. Для розв'язання задач проектування використовуються 3 типи вихідних даних:
– вихідні дані, уведені користувачем;
– варійовані чинники;
– обмеження.

Вихідні дані (*1*) формує користувач, тому є необхідність у їх перевірці і корегуванні. Блок вихідних даних включає формування, корегування та роздрукування списку вхідних даних.

Варійовані чинники (*2*) – це дані, отримані з таблиць та довідників, тому перевіряти їх не має потреби. Блок варійованих чинників включає завдання або завантаження та роздрукування списку варійованих чинників.

Обмеження (*3*) використовуються для задавання умов оптимізації. Блок обмежень включає формування, корегування та роздрукування списку обмежень.

Блок розрахункового модуля (*4*) містить процедури з пошуку оптимального рішення згідно з поставленою метою.

Проектні рішення (*5*) – це результати роботи проектної процедури. Блок проектних рішень включає підготовку даних для оцінки рішень, візуалізацію та документування проектних рішень.

Наведена схема може видозмінюватися відповідно до завдань проектування.

1.7. Види забезпечення САПР

САПР складається із підсистем, які у свою чергу складаються з компонентів, що забезпечують функціонування підсистеми. Компоненти виконують певну функцію у підсистемі і є найменшим (неподільним) елементом. Сукупність однотипних компонентів утворює засіб забезпечення САПР [1, 4, 9].

Математичне забезпечення описує у взаємозв'язку об'єкт, процес і засоби автоматизації проектування та являє собою сукупність математичних методів чисельного розв'язання задач, функціональних моделей проектованих об'єктів, математичних моделей та алгоритмів проектування. Воно поділяється на математичні методи і побудовані на їх основі математичні моделі об'єктів проектування та на формалізований опис технології автоматизованого проектування.

Технічне забезпечення – це сукупність взаємопов'язаних і взаємодіючих технічних засобів (ЕОМ, периферійні пристрої, мережеве устаткування, лінії зв'язку, вимірювальні засоби).

Програмне забезпечення – сукупність усіх машинних програм та експлуатаційної документації до них, необхідних для виконання автоматизованого проектування. Воно поділяється на загальносистемне і прикладне.

Прикладне програмне забезпечення реалізує математичне забезпечення для безпосереднього виконання проектних процедур. Включає пакети програм, призначені для обслуговування певних етапів проектування або розв'язання груп однотип-

них завдань усередині різних етапів (модуль проектування трубопроводів, пакет моделювання схемотехніки, геометричний роз'вязувач САПР).

Загальносистемне програмне забезпечення створено для організації функціонування технічних засобів, тобто планування й управління обчислювальним процесом, розподілення наявних ресурсів. Прикладом компонента загальносистемного програмного забезпечення є операційна система.

Інформаційне забезпечення – це сукупність баз даних (БД), систем управління базами даних (СУБД) та інших даних і відомостей, які використовуються при проектуванні. Уся сукупність використовуваних при проектуванні даних називається інформаційним фондом САПР, а база даних разом з СУБД носить назву банку даних. Крім того, до інформаційного забезпечення належать довідкові дані про комплектуючі вироби, типові проектні рішення, параметри елементів, зведення про стан поточних розроблень у вигляді проміжних та остаточних проектних рішень, структур і параметрів проектованих об'єктів.

Лінгвістичне забезпечення – сукупність мов спілкування між проектувальниками та ЕОМ, мов програмування та мов обміну даними між технічними засобами САПР, включаючи терміни і визначення, правила формалізації природної мови і методи компресії та розгортання текстів.

Методичне забезпечення – це сукупність документів, які мають характер інструкцій, що регламентують порядок експлуатації системи, опис технології функціонування САПР. Включає методи вибору і застосування користувачами технологічних прийомів для отримання конкретних результатів, теорію процесів, що відбуваються у проектованих об'єктах, методи аналізу, синтезу систем та їх складових частин, різні методики проектування.

Організаційне забезпечення – сукупність штатних розписів, посадових інструкцій та інших документів, що встановлюють склад проектної організації, її підрозділів, зв'язки між ними, їх функції, а також форму представлення результату проектування, порядок розгляду проектних документів та документів, що визначають роботу проектного підприємства.

Ергономічне забезпечення об'єднує взаємопов'язані вимоги, спрямовані на узгодження психологічних, психофізіологіч-

них, антропометричних характеристик і можливостей людини з технічними характеристиками засобів автоматизації і параметрами робочого середовища на робочому місці.

Правове забезпечення складається з правових норм, що регламентують правовідносини при функціонуванні САПР та юридичний статус результатів її функціонування.

1.8. Основні принципи побудови САПР

САПР повинна працювати просто, зручно і надійно [8].

Простота роботи означає, що робота користувача має бути максимально наближена до роботи у «просто SolidWorks», інтерфейс команд повинен бути таким же, що і стандартний.

Усі налаштування повинні виконуватися потай від користувача залежно від основних властивостей проекту, що задаються при його створенні. Користувач повинен працювати за простою схемою: вибір команди у меню, вибір параметрів з максимально спрощеного діалогу, декілька вказівок на екрані, завершення команди шляхом вибору явно видимої опції виходу.

Зручність роботи складається з непомітних, на перший погляд, дрібниць. Наприклад, зручно для установки ширини лінії обрати з двох варіантів: «тонка» або «основна» чи за необхідності задати ширину у міліметрах на папері, а не вираховувати її в одиницях малюнка з урахуванням масштабного коефіцієнта.

Зручність полягає в тому, що користувач має можливість у будь-який момент завершити будь-яку операцію штатними засобами, а не проходити усі етапи введення даних.

Усі подібні «зручності» мають бути реалізовані при розробленні системи.

Надійність системи означає захист від помилкових дій користувача. Програми мають бути стійкі як до безладних клацань, так і до неумисних помилок користувачів. Наприклад, програма не повинна допускати введення нульових або негативних значень, якщо це неприпустимо за логікою роботи, повинна ще при виборі об'єктів відкидати примітиви неприпустимих типів, повинна розуміти, чи відмовився користувач від продовження дії при виборі об'єкта, або просто схибив.

Надійна програма у будь-якому випадку, у тому числі при помилці, повинна відновити усю робочу обстановку, що діяла до її запуску.

1.9. Стадії створення САПР

Процес створення САПР включає такі стадії [8, 9]:
– передпроектні дослідження;
– технічне завдання;
– технічна пропозиція;
– технічний проект;
– робочий проект;
– введення у дію.

Стадія передпроектного дослідження визначає призначення, принципи побудови (створення) САПР. Проводиться вивчення наявних у проектній організації процесів проектування, закономірностей вдосконалення об'єктів проектування, оцінка техніко-економічної доцільності створення САПР і формування сукупності початкових вимог до функцій і структури САПР.

Розробка технічного завдання проводиться на підставі результатів передпроектних досліджень, а також узагальнення досвіду робіт в галузі САПР. Технічне завдання після узгодження і затвердження є основним документом, що регламентує проведення робіт на наступних стадіях створення САПР, підсистеми або компонента САПР.

Мета робіт на стадії технічної пропозиції – детальне техніко-економічне обґрунтування доцільності створення САПР з функціями і характеристиками, обумовленими у технічному завданні. При розробленні технічної пропозиції проводиться порівняльний аналіз низки варіантів системи, вибір раціонального варіанта САПР та уточнюються вимоги до робіт на наступних стадіях створення САПР.

Метою робіт на стадії технічного проекту є розроблення остаточних технічних рішень, що дають повне уявлення про створювану САПР або її підсистему із заданими функціями і технічними характеристиками. У технічному проекті встановлюється структура системи, склад підсистем і компонентів САПР та зв'язків між ними; складаються технічні завдання на

створення або адаптацію компонентів технічного, програмного й інформаційного забезпечення.

Метою робіт на стадії робочого проекту є складання робочої документації, достатньої для розроблення, налагодження і випробування компонентів САПР та введення у дію підсистем САПР. На стадії робочого проекту мають бути створені, відлагоджені і випробувані компоненти програмного, технічного й інформаційного забезпечення, необхідного для введення підсистеми або черги САПР в дослідну експлуатацію. Допускається проводити доопрацювання документації робочого проекту за результатами випробувань і дослідної експлуатації. Таке доопрацювання може плануватися як окремий етап робіт зі створення САПР.

Стадія введення в дію включає дослідну експлуатацію і приймальні випробування підсистем і компонентів САПР. Введення у дію наступної черги САПР здійснюється шляхом введення у дію нових або модифікованих підсистем САПР. Введення у дію системи здійснюють після дослідного функціонування і приймальних випробувань у замовника.

Контрольні запитання і завдання

1. Яка мета та задачі створення САПР?
2. Окресліть переваги використання САПР при проектуванні.
3. Як забезпечується досягнення мети створення САПР?
4. Дайте визначення проектуванню.
5. У чому полягає автоматизація проектування?
6. Охарактеризуйте неавтоматизоване, автоматизоване та автоматичне проектування.
7. Охарактеризуйте основні існуючі технології у проектуванні.
8. Чим CALS-системи відрізняються від CIM-систем та PDM-систем?
9. Як класифікуються САПР?
10. Назвіть загальносистемні принципи, що використовуються при створенні і розвитку САПР.
11. Як реалізовано принцип розвитку при створенні САПР?
12. У чому полягає принцип інформаційної єдності при створенні САПР?
13. Як реалізовано принцип комплексності при створенні САПР?

14. У чому полягає принцип включення?
15. Як реалізовано принцип сумісності?
16. Що мається на увазі під принципом системної єдності?
17. Як реалізовано принцип інваріантності?
18. Охарактеризуйте системні вимоги до САПР.
19. Перерахуйте функціональні вимоги до САПР.
20. Охарактеризуйте технічні вимоги до САПР.
21. Які організаційно-експлуатаційні вимоги висуваються до САПР?
22. Що називається підсистемою САПР?
23. Охарактеризуйте проектуючі підсистеми САПР.
24. Наведіть приклади проектуючих підсистем САПР.
25. Чим відрізняються інваріантні та об'єктні проектуючі підсистеми САПР?
26. Наведіть приклади об'єктних проектних підсистем САПР.
27. Наведіть приклади інваріантних проектних підсистем САПР.
28. Охарактеризуйте обслуговуючі підсистеми САПР.
29. Наведіть приклади обслуговуючих підсистем САПР.
30. Опишіть будову проектної процедури в САПР.
31. Охарактеризуйте стадії створення САПР.
32. Що називають забезпеченням САПР?
33. Охарактеризуйте основні види забезпечення САПР.
34. Що відноситься до математичного забезпечення?
35. Які існують види програмного забезпечення?
36. Наведіть приклади інформаційного забезпечення.
37. Чим відрізняються методичне та інформаційне забезпечення.
38. Які документи належать до організаційного забезпечення?
39. Що належить до лінгвістичного забезпечення?
40. Що забезпечує правове забезпечення?
41. Що регламентує ергономічне забезпечення?

2. Макроси в SolidWorks

2.1. Поняття макросу

Макросом (або макрокомандою) називається набір інструкцій, що повідомляють програму (таку, як MS Word, MS Excel або SolidWorks), які дії слід виконати, щоб досягти певної мети [3, 5].

Список інструкцій, що становлять макрос, як правило, складається з макрооператорів. Деякі оператори виконують особливі дії, пов'язані з виконанням самого макросу, але більшість операторів відповідають командам меню й опціям діалогових вікон програми, в якому виконується макрос.

Макроси розробляються за допомогою вбудованої мови програмування Visual Basic for Application.

Visual Basic for Application (VBA) – штатна мова програмування систем Microsoft, до складу якої входить пакет проектування SolidWorks. VBA є середовищем програмування, розробленим спеціально для створення макросів в інших додатках. Це спрощений варіант мови програмування Microsoft Visual Basic, що вбудовується у програми системи Windows.

2.2. Робота з панеллю інструментів Макрос

Для входу в середовище VBA використовується панель інструментів **Макрос** вікна SolidWorks (рис. 2.1).

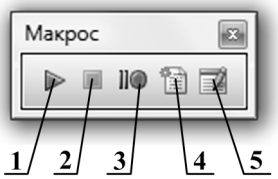


Рис. 2.1. Панель інструментів Макрос: *1* – запуск макросу; *2* – зупинка запису макросу; *3* – запис / пауза запису макросу; *4* – створення макросу; *5* – редагування макросу

Створення макросу

Новий макрос можна створити за допомогою панелі інструментів **Макрос**. Створення нового макросу відрізняється від за-

пису макросу. При створенні нового макросу він програмується безпосередньо у додатку для редагування макросів, наприклад, Microsoft Visual Basic. При запису макросу він створюється усередині програми SolidWorks, відстежуючи та записуючи дії користувача.

Послідовність створення нового макросу:

1. Запустити інструмент **Новий макрос** на панелі інструментів **Макрос**, або обрати у головному меню **Инструменты→Макрос→Новый**.
2. Ввести ім'я файлу макросу у поле **Имя файла**.
3. Натиснути кнопку **Сохранить**. Відкриється вікно середовища VBA для редагування макросів, в якому можна створити новий макрос (рис. 2.2).

Запис/Пауза макросу

Операції, що виконуються за допомогою інтерфейсу користувача SolidWorks, можна записати і відтворювати, використовуючи макроси. Ці макроси містять виклики, еквівалентні викликам функцій API, які здійснювалися при виконанні операцій за допомогою інтерфейсу користувача. У макрос може записати вибір за допомогою миші, вибір у меню і введену за допомогою клавіатури інформацію.

Послідовність запису макросу:

1. Запустити інструмент **Запись/Пауза макроса** на панелі інструментів **Макрос** або обрати у головному меню **Инструменты→Макрос→Запись**.
2. Виконати кроки, які треба записати.
3. Запустити інструмент **Остановить запись макроса** на панелі інструментів **Макрос** або обрати у головному меню **Инструменты→Макрос→Остановить** з**апись**.
4. У діалоговому вікні ввести ім'я в поле **Имя файла** і натиснути кнопку **Сохранить**.

Пауза під час запису макросу:

1. Запустити інструмент **Запись/Пауза макроса**, або обрати у головному меню **Инструменты→Макрос→Пауза**.
2. Для поновлення запису макросу знову натиснути **Запись/Пауза макроса** або обрати у головному меню **Инструменты→Макрос→Пауза**.

Зупинка запису макросу:
1. Під час запису макросу, запустити інструмент **Остановить запись макроса** на панелі інструментів **Макрос** або обрати у головному меню **Инструменты→Макрос→Стоп**.
2. Ввести ім'я макросу в полі **Имя файла** і натиснути кнопку **Сохранить**.

Редагування макросу

Записаний макрос можна редагувати або налагоджувати.

Послідовність редагування макросу:
1. Запустити інструмент **Редактировать макрос** на панелі інструментів **Макрос** або обрати у головному меню **Инструменты→Макрос→Изменить.**
2. У діалоговому вікні, що відкрилося, обрати файл макросу (**.swp**) і натиснути **Открыть**. Можна також відкрити та відредагувати файли **.swb** (файл створений у середовищі Microsoft Visual Basic). При редагуванні він автоматично буде перетворений у файл **.swp**.
3. Відредагувати макрос та зберегти його.

Запуск макросу:
1. Запустити інструмент **Выполнить макрос** на панелі інструментів Макрос або обрати у головному меню **Инструменты→Макрос→Выполнить.**
2. У діалоговому вікні знайти файл макросу (**.swp, .swb**) і натиснути кнопку **Открыть**.

2.3. Елементи інтегрованого середовища розроблення програм VBA

Написання та налагодження програм і макросів проводиться в інтегрованому середовищі розроблення VBA, що входить до складу системи SolidWorks.

Розглянемо основні елементи середовища VBA (рис 2.2).

Головне меню (*1*) містить усі інструменти та налаштування. У головному меню є стандартні пункти, властиві багатьом Windows-програмам: **File** (Файл), **Edit** (Правка), **View** (Вид), **Tools** (Сервіс), **Help** (Допомога) та ін. Крім того меню містить пункти, за допомогою яких користувач може створювати, запускати і налагоджувати макрос: **Insert** (Вставка), **Run** (Запуск), **Debug** (Налагодження) та ін.

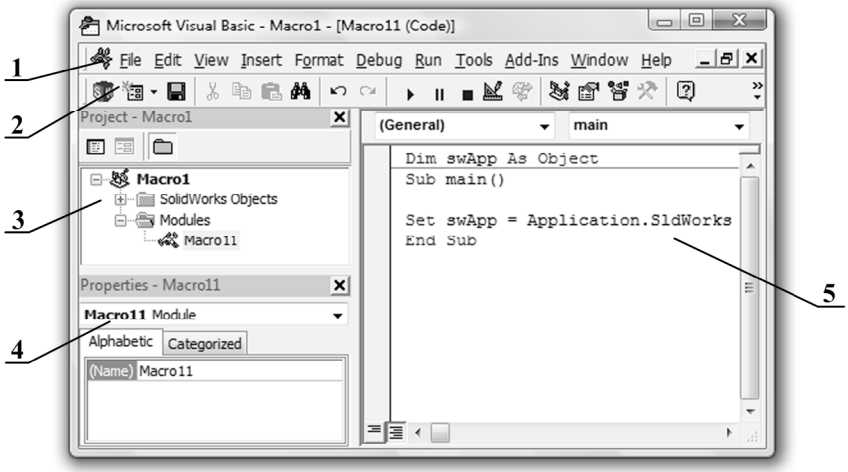


Рис. 2.2. Інтегроване середовище розроблення програм VBA [10]: *1* – головне меню; *2* – панелі інструментів; *3* – вікно коду з текстом макросу; *4* – менеджер проекту; *5* – менеджер властивостей

Панелі інструментів (*2*) середовища розроблення VBA: **Standard** (Стандартна), **Edit** (Правка), **Debug** (Налагодження) та **UserForm** (Форма користувача). За замовчуванням відображається тільки стандартна панель інструментів **Standard.**

Для того, щоб увімкнути або вимкнути панель інструментів, у головному меню потрібно обрати **View** (Вид)→**Toolbars** (Панелі інструментів) і поставити або прибрати відмітку біля потрібної панелі інструментів.

Розглянемо панелі інструментів середовища VBA.

Панель **Standard** (Стандарт) (рис. 2.3) є основною. З її допомогою можна виконувати широкий спектр дій. На ній частково дублюються інструменти з інших панелей. Вона зазвичай розташована під рядком меню, проте за допомогою мишки її можна перетягувати в інші місця вікна VBA.

Панель **Edit** (Правка) (рис. 2.4) призначено для роботи з текстом програми. Ця панель реалізує можливості простого текстового редактора: копіювання і переміщення тексту у буфер обміну, вставку тексту з буфера, пошук і заміну слів і фраз в тексті програми та ін.

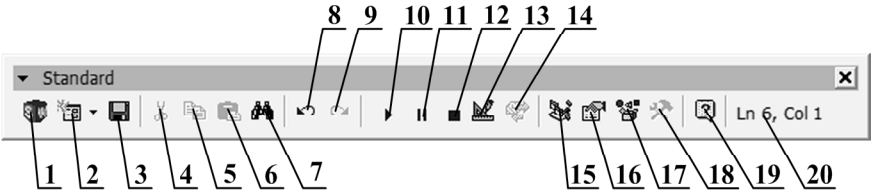


Рис. 2.3. Панель **Standard** [10]: *1* – перехід у головне вікно SolidWorks; *2* – створення форми користувача; *3* – збереження макросу; *4* – вирізання тексту; *5* – копіювання тексту; *6* – вставлення тексту; *7* – пошук у тексті; *8* – скасування дії; *9* – повторення дії; *10* – запускання макросу; *11* – переривання роботи макросу; *12* – припинення роботи макросу; *13* – увімкнення/вимкнення режиму дизайну; *14* – запускання усього проекту; *15* – увімкнення/вимкнення вікна проекту (поз. *3*, див. рис 2.2); *16* – увімкнення/вимкнення вікна властивостей форми й елементів керування (поз. *4*, див. рис 3.2); *17* – увімкнення/вимкнення вікна об'єктів; *18* – увімкнення/вимкнення панелі ToolBox; *19* – допомога VBA; *20* – поточна позиція курсора у тексті

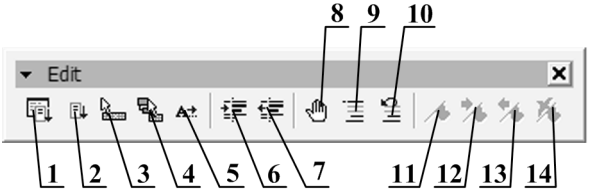


Рис. 2.4. Панель **Edit** [10]: *1* – відображення списку методів і функцій виділеного об'єкта; *2* – відображення списку констант; *3* – підказування вірного синтаксису виділеної команди, оператора тощо; *4* – відображення параметрів функції ; *5* – підказування під час уведення можливих варіантів завершення команди, оператора тощо; *6* – встановлення відступу; *7* – прибирання відступу; *8* – встановлення точки зупинення виконання програми; *9* – перетворення виділеного тексту у коментар; *10* – перетворення виділеного коментаря у текст; *11* – встановлення закладки; *12* – перехід на наступну закладку; *13* – перехід на попередню закладку; *14* – видалення усіх закладок

Панель **Debug** (Налагодження) (рис. 2.5) призначено для налагодження програми. Передбачено широкі можливості для налагодження: відстеження поточних значень змінних програми, покрокове виконання програми, при якому на кожному кроці виконується один оператор або його частина, тощо.

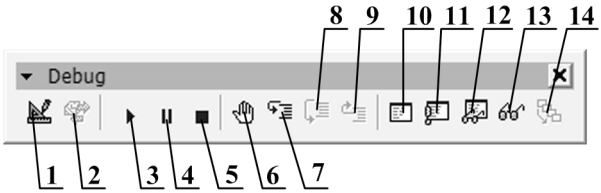


Рис. 2.5. Панель **Debug** [10]: *1* – увімкнення/вимкнення режиму дизайну; *2* – запускання усього проекту; *3* – запускання макросу; *4* – переривання роботи макросу; *5* – припинення роботи макросу; *6* – встановлення точки зупинення виконання програми; *7* – покрокове виконання програми із заходом у процедури та функції; *8* – покрокове виконання програми без заходу у процедури та функції; *9* – покрокове виконання програми, процедури або функції, починаючи від курсору і до кінця; *10* – увімкнення/вимкнення вікна відображення значень усіх оголошених змінних у поточній процедурі або функції; *11* – увімкнення/вимкнення вікна відображення результатів налагодження програми; *12* – увімкнення/вимкнення вікна відображення значень усіх виділених змінних; *13* – відображення значення виділеної змінної; *14* – відображення активних процедур і функцій під час переривання роботи макросу

Панель **UserForm** (Форма користувача) (рис. 2.6) призначена для допомоги при розміщенні на формі користувача елементів керування: вирівнювання елементів по горизонталі і вертикалі, групування елементів та ін.

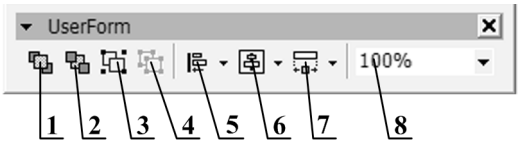


Рис. 2.6. Панель **UserForm** [10]: *1* – перенести на передній план; *2* – перенести на задній план; *3* – групування елементи; *4* – розгрупування елементи; *5* – режими вирівнювання групи елементів; *6* – вирівнювання елементів по горизонталі/по вертикалі на формі; *7* – вирівнювання розмірів; *8* – масштаб відображення форми користувача

2.4. Основні команди налагодження програм

Після набору тексту програми слідує етап налагодження, тобто виявлення і виправлення помилок в програмі. При цьому використовується вбудований налагоджувач.

Інструменти налагоджувача розташовано у головному меню у пункті **Debug** або на панелі інструментів **Debug**.

Розглянемо деякі інструменти налагоджувача.

Step Into (Крок із заходом) – покрокове виконання програми із заходом у процедури та функції. Ця команда також виконується при натисненні клавіші F8.

Run To Cursor (Виконати до поточної позиції) – виконання програми до курсору. Для установки курсору треба натиснути ліву кнопку миші у потрібному місці програми. Ця команда виконується також при натисненні клавіш Ctrl + F8.

Toggle Breakpoint (Точка зупинки) – установка або ліквідація точки зупинки виконання програми. Такою точкою відзначається рядок в програмі, перед яким виконання програми тимчасово припиняється. Установка або ліквідація точки зупинки робиться в тому місці, де знаходиться курсор. Ця команда також виконується при натисненні клавіші F9.

Для установки або ліквідації точки зупинки також можна натиснути ліву кнопку миші на сірій лівій межі вікна коду навпроти потрібного рядка.

Clear All Breakpoints (Зняти усі точки зупинки) – ліквідація усіх точок зупинки.

Add Watch (Додати контрольне значення) – візуалізація поточних значень обраних змінних під час виконання програми.

Крім вище означених інструментів, на панелі **Debug** розміщено три інструменти з пункту головного меню **Run**.

Розглянемо їх.

Run (Запуск) – запуск програми і перехід від однієї точки зупинки до іншої. Виконання програми означає послідовне виконання її операторів. Якщо точок зупинки немає, то програма виконується повністю.

Break (Пауза) – тимчасове переривання виконання програми. Повторне використання інструменту продовжує виконання програми.

При зупинках під час виконання програми стрілкою ліворуч і жовтим кольором виділяється оператор, який ще не вико-

наний. Значення тієї або іншої змінної можна побачити, підвівши до неї курсор мишки.

Reset (Зупинка) – повне припинення виконання програми.

2.5. Структура макросу

Текст програми відображається та набирається користувачем у вікні коду (*3*, див. рис.2.2).

Вікно коду (рис. 2.7) умовно поділено на дві частини: верхню, де оголошуються змінні (*3*), та нижню, де записується текст програми, процедур та функцій (*4*). Перший та останній рядки (оператори) макросу стандартні.

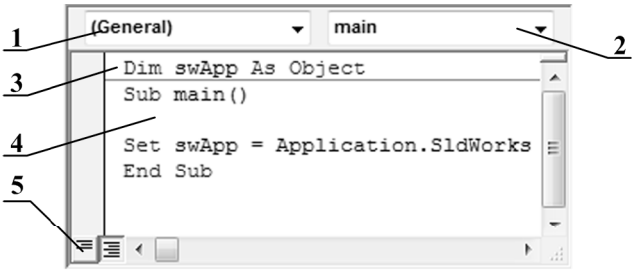


Рис. 2.7. Вікно набору коду: *1* – список програм, форм користувача й елементів управління; *2* – список оголошення змінних і процедур обробників подій; *3* – зона визначення змінних; *4* – зона процедур і функцій; *5* – кнопки режиму відображення тексту

Між першим і останнім операторами набираються інші оператори програми. При цьому можна користуватися звичайними командами редагування (як у текстовому редакторі MSWord), а також буфером обміну. Введення кожного рядка програми завершується натисканням кнопки **Enter.**

Ім'я макросу призначається користувачем. Воно повинне відповідати таким умовам:

– перший символ імені має бути буквою;
– ім'я може містити тільки букви, цифри і символ з'єднання _;
– ім'я не повинне містити більше 255 символів.

Приклади імен макросів

`MyProgram, My_Program, MyProgram1`.

2.6. Створення іконки для макросу

Послідовність створення іконки на панелі інструментів для швидкого виклику макросу, розробленого користувачем:
1. Створити макрос і зберегти його.
2. Обрати у головному меню SolidWorks пункт **Инструменты→ Настройка**.
3. У вікні **Настройк**а обрати закладку **Команды**.
4. У списку **Категории** обрати панель інструментів **Макрос**. Праворуч буде показано доступні інструменти. Один з них – **Создать кнопку для макроса**. Обрати його і, утримуючи ліву кнопку миші, перетягнути на панель інструментів або стрічку (рис. 2.8).

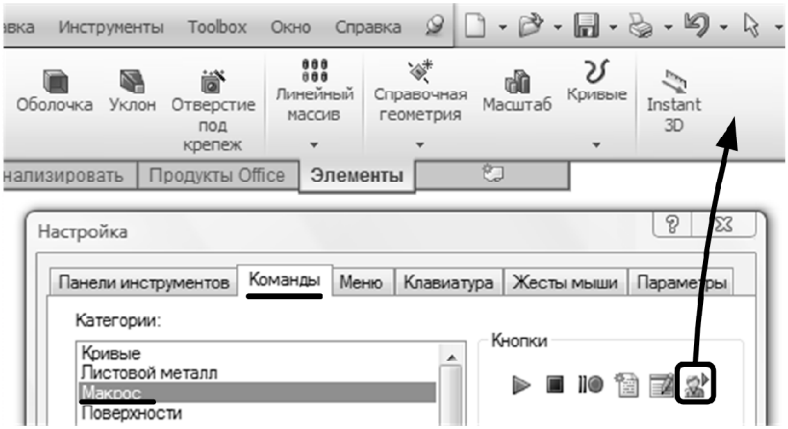


Рис. 2.8. Перетягування кнопки для макросу

5. Буде автоматично відкрито вікно **Настройка кнопки макроса** (рис 2.9).

У розділі **Результат** для завдання зображення кнопки макросу необхідно натиснути кнопку **Выбрать образ…** та обрати зображення з диска. У полі **Подсказка** записати коротку назву макросу. У полі **Спросить** записати коротку довідку, що пояснює призначення макросу. У розділі **Действие** у полі **Макрос** записати повну назву файлу макросу, включаючи шлях розташування на диску. Для пошуку файлу макросу на диску натиснути кнопку **…** .

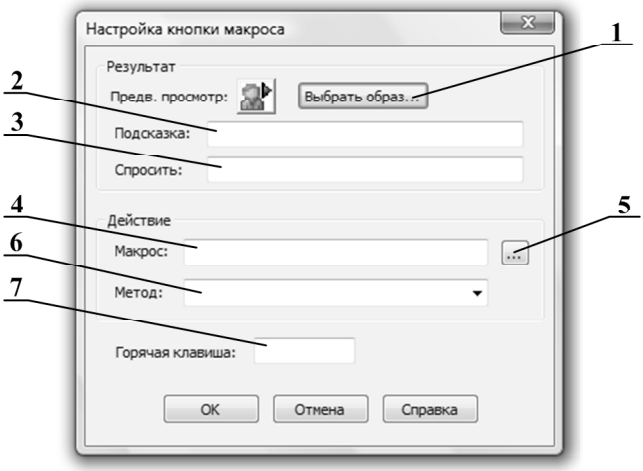


Рис. 2.9. Параметри кнопки: *1* – завдання зображення кнопки макросу; *2* – коротка назва макросу; *3* – коротка довідка, що пояснює призначення макросу; *4* – повна назва файла макросу, включаючи шлях розташування на диску; *5* – пошук файла макросу на диску; *6* – ім'я основної процедури макросу; *7* – поєднання клавіш для швидкого запуску макросу

При цьому у полі **Метод** буде записано ім'я основної процедури макросу. У полі **Горячая клавиша** можна задати поєднання клавіш для швидкого запуску макросу.

Вимоги до точкового зображення для кнопки макросу:
– розміри зображення – 16 х 16 пікселів;
– режим кольорів – 256 кольорів;
– колір фону – білий;
– формат файлу – **bmp**.

2.7. Приклад створення макросу для побудови моделі

Завдання: створити макрос, що будує тривимірну модель шайби за визначеним ескізом (рис. 2.10), налаштувати кнопку та розмістити її на панелі інструментів для швидкого запуску створеного макросу.

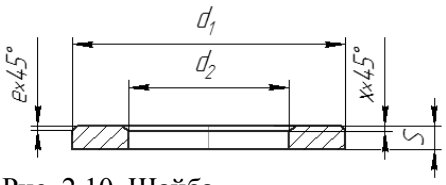


Рис. 2.10. Шайба

Розміри шайби:
d_1 = 20мм;
d_2 = 34мм;
S = 3 мм;
e = 1мм;
x = 1,5мм.

Послідовність дій при створенні макросу побудови шайби:
1. Для створення нового макросу на основі дій користувача у SolidWorks на панелі інструментів **Макрос** натиснути кнопку **Запись/Пауза макроса**.
2. Створити модель шайби.
2.1. Створити новий документ деталі SolidWorks. Для цього обрати у головному меню пункт **Файл→Новый**, у вікні **Новый документ** обрати **Деталь** та натиснути кнопку **Ок**. Буде створено новий документ деталі.
2.2. Запустити інструмент **Повернутая бобышка/основание** на стрічці на закладці **Элементы**.

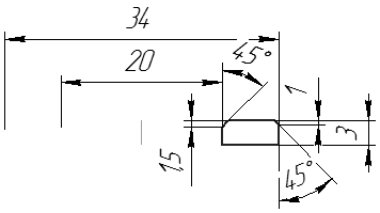


Рис. 2.11. Ескіз для побудови моделі шайби

2.3. Обрати площину **Спереди** для побудови ескізу шайби. Використовуючи інструменти зі стрічки з закладки **Эскиз**, побудувати ескіз перетину шайби згідно з вихідними даними. Оскільки модель буде побудовано обертанням половини ескізу перетину навколо осі шайби, то будується тільки половина її перетину (рис. 2.11).
2.4. Завершити побудову ескізу, натиснувши іконку **Выйти из эскиза и сохранить** у правому верхньому куті графічної області. Модель шайби побудовано (рис. 2.12).

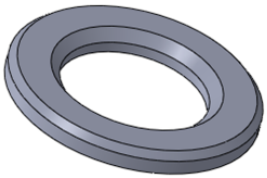


Рис. 2.12. Твердотільна модель шайби

3. Натиснути кнопку **Стоп** на панелі **Макрос**. Буде відкрито вікно **Сохранить** для збереження макросу. У діалоговому вікні ввести ім'я макросу у полі **Имя файла** і натиснути кнопку **Сохранить.**
4. Створити зображення для кнопки макросу. Графічний редактор для цього може бути будь-яким, що підтримує створення ра-

стрових графічних зображень у форматі **.bmp**. У даному прикладі далі розглядається використання стандартного графічного редактора **Paint**, що входить до складу операційної системи **Windows**.

4.1. Запустити графічний редактор **Paint**. У головному меню обрати пункт **Рисунок→Атрибуты**. У вікні **Атрибуты** встановити наступні параметри: у полях **Ширина** і **Высота** записати 16, **Единицы измерения→точки**, **Палитра→Цветная** (див. рис. 2.13).

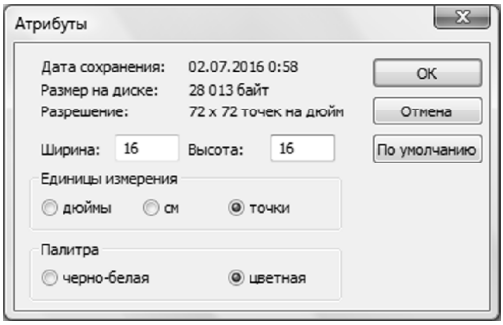


Рис. 2.13. Вікно атрибутів **MS Paint**

4.2. Використовуючи інструменти для малювання, створити зображення іконки для макросу.

4.3. У головному меню обрати пункт **Файл→Сохранить**. У діалоговому вікні у полі **Имя файла** ввести ім'я зображення, а у полі **Тип файла** обрати **256–цветный рисунок (*.bmp, *.dib)** і натиснути кнопку **Сохранить**.

5. Налаштування кнопки для макросу.

5.1. У головному меню обрати пункт **Инструменты→Настройка.** Буде відкрито вікно **Настройка**.

5.2. У вікні **Настройка** обрати закладку **Команды**, у списку **Категории** обрати панель **Макрос.**

5.3. У полі **Кнопки** оберіть кнопку **Создать кнопку для макроса** та, утримуючи ліву кнопку миші,перетягнути її на будь-яку панель інструментів. При цьому буде відкрито вікно **Настройки кнопки макроса** (рис. 2.8).

5.4. У вікні **Настройка кнопки макроса** зробити такі налаштування. У розділі **Результат** натиснути кнопку **Выбрать образ** та вказати вкажіть шлях на диску до створеного файлу зображен-

ня іконки для макросу. У полі **Подсказка** записати «**Побудова шайби**». У полі **Спросить** записати «**Побудова шайби 20х3**». У розділі **Действие** у полі **Макрос** записати повну назву файлу макросу, включаючи шлях розташування на диску. Для пошуку файлу макросу на диску натиснути кнопку **…** При цьому у полі **Метод** буде записано ім'я основної процедури макросу.

Контрольні запитання і завдання

1. Що називається макросом?
2. Яка послідовність дій при запису макросу?
3. Чим відрізняється запис макросу від створення?
4. Яким чином відбувається редагування макросу?
5. Що називається інтегрованим середовищем розробки VBA?
6. Охарактеризуйте чотири панелі інструментів середовища VBA.
7. Опишіть команди для налагодження макросів.
8. Призначення етапу налагодження програми.
9. Опишіть команди для запуску макросів з середовища VBA.
10. Яку структуру має макрос у середовищі VBA?
11. Які висуваються вимоги до імені макросу?
12. Опишіть порядок створення кнопки на панелі інструментів для запуску макросу користувача.
13. Створити макрос для побудови моделі шайби упорної швидкозйомної з внутрішнім діаметром 15 мм згідно ГОСТ 11648–75.

3. Основи мови програмування VBA

3.1. Основні поняття мови програмування VBA

3.1.1. Загальні синтаксичні принципи мови VBA

Основні вимоги до тексту макросів [3, 5]:

– регістр не має значення – оператори можуть бути прописані як у верхньому, так і в нижньому регістрі, або взагалі упереміш, наприклад, **dim**, **DIM**, **Dim**, різниці немає;
– максимальна довжина будь-якого імені складає 255 знаків;
– у кінці рядка з операторами не треба ставити коми;
– щоб не прописувати кожен оператор в окремому рядку, їх можна за допомогою двокрапки прописати в один рядок, наприклад: **MsgBox "Строка1": MsgBox "Строка2"**;
– якщо рядок з операторами занадто довгий, то його можна розділити на декілька рядків за допомогою символу підкреслення і знаку **&**, наприклад:

```
MsgBox "Це простий" _&
"рядок"
```

3.1.2. Коментарі

Коментар – це текст, що пояснює призначення і порядок виконуваних дій у програмі [3, 5]. Цей елемент програмування не є обов'язковим, але він дозволяє полегшити розуміння внутрішньої логіки програми. Коментар може розташовуватися як на окремому рядку, так і праворуч від виконуваного оператора.

Коментар позначається символом апострофа (**'**) або ключовим словом **Rem**. Усе, що записано лівіше за ці знаки, належить до коментарю і ігнорується програмою при її виконанні.

Приклад

```
Dim I As Integer        'оголошення змінної
I = 12
Rem присвоєння первинного значення
```

3.1.3. Типи даних, якими оперує VBA

Тип даних визначає, яким чином задана величина зберігається у пам'яті комп'ютера, скільки розрядів може містити її значення, тощо. VBA має кілька вбудованих типів даних, наведених у табл. 3.1 [3, 5].

Таблиця 3.1

Типи даних VBA

Назва типу	Тип даних	Розмір у байтах	Розрядність цифр	Діапазон значень
Логічний	`Boolean`	2	1	`True` або `False` (так – 1 ні – 0)
Цілий	`Integer`	2	5	Від –32768 до +32768
Дрібний	`Long`	4	10	Від –2147483648 до +2147483647
	`Single`	4	7	Від –3.402823E+38 до –1.401298E–45 і від +1.401298E–45 до +3.402823E+38
	`Double`	8	15	Від ± 1.79769313486E+308 до ± 4.94065645841E–324
Грошовий	`Currency`	8	19	Від –922337203685477.5808 до 922337203685477.5807
Дата	`Date`	8	-	Від 01.01.100 до 31.12.9999
Рядковий	`String`	1	+1	Будь-який символ від 0 до 65535 символів
Об'єкт	`Object`	4	-	Будь-який об'єкт
Масив	`Array`	Визначається кількістю і розміром елементів		
Універсальний	`Variant`	Тип визначається типом даних, який буде передано змінній		

3.1.4. Змінні

Змінні – це пойменовані області у пам'яті комп'ютера [3, 5]. Отримане значення – число або текст – має бути записане в пам'ять, а щоб використовувати його в подальших діях, необхідно викликати його з пам'яті. VBA створює пряму відповідність між областю (адресою) пам'яті і заданим ім'ям змінної.

Імена змінних складаються з алфавітних символів, цифр і спеціальних символів, але починатися повинні обов'язково з буквеного символу. Не допускається використання в імені пропусків, розділових знаків і наступних символів: **#**, **$**, **%**, **&**, **!**.

Тип змінних визначається типом даних, який повинен використовуватися при запису її значення в пам'ять. За замовчуванням використовується тип **Variant**. Тобто тип змінної визначається типом даних, який буде передано їй.

Оголошення змінних проводиться за допомогою оператора **Dim**. У VBA не можна робити оголошення списком. Обов'язково слід вказувати тип для кожної змінної, що знову вводиться, інакше вона буде віднесена до типу **Variant**.

Синтаксис оператора оголошення змінної:

Dim змінна [As тип]

де:

Dim – ключове слово, що свідчить про те, що оголошується змінна (dimension – розмір);

змінна – ім'я оголошуваної змінної;

As – ключове слово, використовуване у процесі задавання типу даних;

тип – тип даних для оголошуваної змінної або (що те ж саме) тип змінної.

Одним оператором **Dim** можна описати декілька змінних, перерахувавши їх через кому.

Приклади

```
Dim theName as String
Dim Cost as Currency, I as Integer
Dim MyPicture as Variant
Dim i As Byte, j As Integer, k As Integer
Dim K_i As Byte, Cila_Zminna_1 As Integer
```

Зона видимості змінних, оголошених за допомогою оператора **Dim**, обмежується тими модулями і процедурами, в яких ці

змінні оголошено. Щоб зробити змінну доступною усім процедурам в усіх модулях, що входять до складу програми, замість оператора **Dim** у головній програмі, слід вказати ключове слово **Public**.

Приклади

```
Public Var1 As Integer
Public Cost1 as Currency, A as Integer
```

3.1.5. Константи

Разом зі змінними у VBA використовуються константи.

Константи – це змінні, значення яких у програмі змінити не можна [3, 5].

Існує два різновиди констант – вбудовані та визначені користувачем. Вбудовані константи не вимагають оголошення. Імена вбудованих констант VBA починаються з префікса **vb**, наприклад, **vbFriday** (число `6`; `Friday` – п'ятниця). Визначені користувачем константи вимагають оголошення.

Синтаксис оголошення константи:

Const константа [As тип] = значення

де:

Const – ключове слово, яке показує, що оголошується константа;

константа – ім'я оголошуваної константи;

As – ключове слово, з якого починається процес задавання типу даних;

тип – тип даних для оголошуваної константи;

значення – значення, що привласнюється константі.

Обмеження на імена констант такі ж, як і на імена змінних.

За допомогою одного оператора **Const** можна оголосити декілька констант, перерахувавши їх через кому.

Приклади

```
Const pi As Double = 3.141592654
Const e As Double = 2.718281828
Const Message = "Завершення роботи"
Const Millennium As Date = #1 Jan 2000#
Const beta As Currency = 1/3
Const Min = 0, Max = 1000, Flag As Boolean = False
```

3.1.6. Масиви

Масивом – це послідовність або таблиця змінних одного типу, званих *елементами масиву* [3, 5]. У зверненні до елементу вказується ім'я масиву та один або декілька індексів.

Перш ніж використовувати масив, його слід описати (оголосити). Крім того, для кожного індексу мають бути визначені нижня і верхня межі, в рамках яких індекс може мінятися.

Існує два види масивів – статичні і динамічні.

Статичні масиви

При описі статичного одновимірного масиву задається нижня і верхня межі для індексу, що визначають кількість елементів масиву, причому задані межі не можуть бути змінені у програмі.

Синтаксис оголошення статичного масиву:

Dim масив (розмірність) As тип.

де:

Dim – ключове слово, що свідчить про те, що оголошується масив;

масив – ім'я оголошуваного масиву;

розмірність – межі для індексу, що визначають кількість елементів масиву;

As – ключове слово, використовуване при задаванні типу даних;

тип – тип даних для оголошуваного масиву.

Межами є цілі числа у круглих дужках. Між нижньою і верхньою межами ставиться ключове слово **To**.

Якщо в дужках вказано тільки одне ціле число, то це – верхня межа. При цьому нижня межа вважається рівною нулю.

Приклади

```
Dim Name_Client(100) As String
                        'одновимірний масив, в якому
                        '101 елемент, індекс набуває
                        'значень від 0 до 100
Dim Contacts(3, 75 To 90) As String
                        'двовимірний масив, що містить
                        'діапазон значень двох
                        'індексів. Перший набуває
                        'значень від 0 до 3, другий –
                        'від 75 до 90
```

Динамічні масиви

Динамічні масиви використовуються у тому випадку, коли кількість елементів масиву заздалегідь невідома, а визначається у процесі виконання програми. Після закінчення роботи з дина-

мічним масивом можна вивільнити пам'ять, яку він займає. Це важливо для завдань, що вимагають великого об'єму оперативної пам'яті.

Опис динамічного масиву здійснюється у два етапи.

Оголошується масив з використанням оператора **Dim**, але без вказівки розмірності. Ознакою масиву є дужки після його імені.

Синтаксис:

```
Dim масив () As тип.
```

У потрібному місці програми описується цей масив з указівкою розмірності за допомогою оператора **ReDim**.

Синтаксис:

```
ReDim масив (розмірність)
```

Як межі можна використовувати не лише цілі числа, але й арифметичні вирази. Важливо, щоб до моменту виконання оператора **ReDim** усі змінні у цих арифметичних виразах мали числові значення.

Приклад

```
Dim A() As Byte          'оголошення динамічного масиву
ReDim A(– 5 To 2)        'визначення розмірності масиву
                         'від –5 до 2
ReDim A(5)               'визначення розмірності масиву
                         'від 0 до 5
```

3.1.7. Типи даних, визначувані користувачем

У тому випадку, коли декілька змінних різних типів за смислом пов'язано між собою, їх доцільно об'єднати у так званий *запис* [3, 5]. Кожна змінна, що є складовою частиною запису, називається *полем*.

Для того, щоб оголосити свій тип даних, слід записати ключове слово **Type** і його ім'я. Потім указати імена і типи елементів даних, що входять до складу призначеного користувачем типу. Блок закінчується покажчиком закінчення створення типу **End Type**.

Синтаксис:

```
Type назва
  поле1 As тип1
```

```
  поле2 As тип2
  . . .  . . . .
  полеN  As типN
End Type
```

де:

назва – назва запису;

поле1, поле2, .., полеN – імена полів;

тип1, тип2, .., типN – типи даних для відповідних полів.

Розглянутий оператор створення запису розміщується перед першим рядком програми.

Приклад

Запис **MyContacts** описує сторінку телефонного довідника

```
Type MyContacts           'початок оголошення типу даних
  Name as String          'ім'я абонента
  Address as String       'адреса абонента
  Phone as Integer        'номер телефону абонента
  S_Pasport As String     'серія паспорта
  N_Pasport As Integer    'номер паспорта
End Type                  'кінець оголошення типу даних
```

Оголошення змінної цього типу робиться так само, як і звичайної змінної за допомогою оператора **Dim**.

Приклад

```
Dim MyClients(17) As MyContact
                          'визначення масиву записів
                          'MyContact
```

Звернення до поля запису здійснюється за допомогою імені змінної та імені поля після точки. Доступ до кожного елементу масиву **MyClients** здійснюється таким чином:

```
MyClients(i).Name = "Іванов П.А."
```

тут **i** – індекс масиву, який у цьому прикладі може набувати значення від 0 до 17.

Для заповнення декількох полів зручно використовувати оператор **With,** який має такий синтаксис:

```
With змінна
  поле1 = вираз1
  поле2 = вираз2
  . . . . . .
```

```
  полеN = виразN
End With
```

Приклад

```
With MyContact
  Name = "Захаркін Максим Сергійович"
  Address = "вул. Правди, буд.1, кв. 5"
  Phone = 5896753
  S_Pasport = "АМ"
  N_Pasport = 124439
End With
```

Оператор привласнення можна застосовувати як до полів, так і до записів загалом.

```
MyClients.Number = MyContact.Number
MyClients = MyContact
```

3.1.8. Оператори привласнення

Оператори привласнення складаються зі змінної, записуваної ліворуч від знака рівності, і формули або числового значення, записуваних праворуч [3, 5].

Приклади

```
x = 1.76
y = x
My_Var = x + y
My_Var_2 = My_var * 3
Str_A = "Термінове "
StrC = strA + "повідомлення"
```

3.1.9. Операції VBA

Операції у VBA призначені для виконання основних математичних дій – додавання, віднімання, множення, ділення та ін. Математичні операції VBA у порядку зменшення пріоритету наведено у табл. 3.2 [3, 5].

Установлений у VBA пріоритет операції дозволяє визначити, яка з них у введеному математичному виразі виконуватиметься першою, тобто задає послідовність математичних дій.

Таблиця 3.2

Операції VBA

Операція	Виконувана дія
`( )`	Дужки
`^`	Піднесення до степеня
`*`	Множення
`/`	Ділення
`\`	Цілочислове ділення
`Mod`	Визначення залишку від ділення
`+`	Додавання
`-`	Віднімання

Пріоритет можна змінити за допомогою виокремлення частини математичного виразу круглими дужками. При цьому вираз, поміщений у дужки, має найвищий пріоритет.

Приклади

```
Res = 24 * 8 + 3/0.5 - 0.5^2+(2.2 * 0.7 + 5.8)^1.5
Z = (X * Y/2 + 3/9) + Y^0.3
Dr = (Res * 0.7 + 1.6)/(3.24 + Z^0.5)
kx = (Res * Z)/(Res + Z)
IntR = kx\2.45
```

3.1.10. Математичні функції VBA

Операції дозволяють виконувати тільки прості математичні дії. Для складніших обчислень використовуються вбудовані математичні функції VBA, наведені у табл. 3.3 [3, 5].

Математичні функції VBA мають по одному аргументу (інколи два), використовуються для оброблення окремих чисел і не можуть застосовуватися до масивів чисел. Якщо у цьому є необхідність, то треба обробляти кожен елемент масиву окремо.

При виклику вбудованої функції слід указати у круглих дужках її аргумент. Виняток становить функція **`Randomize`**, яка включає генератор випадкових чисел і не має аргументів.

Таблиця 3.3

Математичні функції VBA

Функція	Виконувана дія
Atn(a)	Повернення арктангенса кута **a** у радіанах.
Sin(a)	Повернення синуса кута **a** у радіанах.
Cos(a)	Повернення косинуса кута **a** у радіанах.
Tan(a)	Повернення тангенса кута **a** у радіанах.
Exp(x)	Повернення значення e^x.
Log(a)	Повернення натурального логарифма числа **a**.
Sqr(a)	Повернення квадратного кореня числа **a**.
Randomize	Ініціація генератора випадкових чисел.
Rnd(a)	Повернення випадкового числа від 0 до **a**.
Abs(a)	Повернення значення модуля числа **a**.
Sgn(a)	Повернення знака числа **a** (+ чи –).
Fix(a)	Повернення округленого значення числа **a** відсіканням дробової частини.
Round(a,b)	Повернення округленого значення числа **a** до **b** чисел після десятинної коми.
Int(a)	Повернення округленого значення числа **a** до найближчого цілого.

Приклади

```
X = Sin(Alfa)          'визначення синуса кута
Y = Sqr(X)             'визначення квадратного кореня
Randomize              'ініціалізація генератора
                       'випадкових чисел
R = Rnd(10)            'випадкове число у інтервалі
                       'від 0 до 10
```

3.1.11. Функції для роботи з рядками

Рядок – змінна рядкового типу, оголошена за допомогою ключового слова **string**, яка являє собою послідовність символів [3, 5].

Функції для роботи з рядками дозволяють ефективно здійснювати пошук і обробку текстової інформації.

Таблиця 3.3

Основні рядкові функції VBA

Функція	Призначення
StrComp(P1, P2)	Порівнює рядки **P1** і **P2**.
Lcase(P)	Перетворює рядок **P** у нижній регістр.
Ucase(P)	Перетворює рядок **P** у верхній регістр.
Space(K)	Створює рядок пробілів відповідно до заданої кількості **K**.
String(K,C)	Створює рядок символів **C** відповідно до заданої кількості **K**.
Len(P)	Обчислює кількість символів рядка **P**.
Instr(P1,P2)	Шукає підрядок **P2** у рядку **P1**.
Lset(P)	Вирівнює рядок **P** по лівому краю.
Rset(P)	Вирівнює рядок **P** по правому краю.
Left(P, K)	Виділяє з рядка **P** символи кількістю **K**, що відлічуються ліворуч.
Right(P, K)	Виділяє з рядка **P** символи кількістю **K**, що відлічуються праворуч.
Mid(P,H,K)	Виділяє з рядка **P** символи кількістю **K**, що відлічуються, починаючи з символу із порядковим номером **H**.
Ltrim(P)	Видаляє пробіли на початку рядка **P**.
Rtrim(P)	Видаляє пробіли у кінці рядка **P**.
Trim(P)	Видаляє пробіли на початку та у кінці рядка **P**.
Str(I)	Перетворює число **I** у рядок.
Val(P)	Перетворює рядок **P** у число.
Format(I, F)	Перетворює число **I** у рядок за заданим форматом **F**.

Приклади

```
Dim strA As String, strB As String, strC As String
                              'оголошення рядкових змінних
strA = "Рядкова "
strB = "змінна"               'задавання первинних значень
strC = strA & strB
strC = strA + strB
strC = strA & "змінна"
```

```
strC = "Рядкова " + strB'поєднання рядків.
                         'Результатом усіх операцій є
                         'рядок strC = "Рядкова змінна"
strA = "   Рядкова"      'задавання первинного значення
strB = "змінна    "     'задавання первинного значення
strC = LTrim (strA)      'видалення пробілів ліворуч.
                         'Результатом операції є рядок
                         'strC = "Рядкова"
strC = RTrim (strB)      'видалення пробілів праворуч.
                         'Результатом операції є рядок
                         'strC = "змінна"
strA = "   Рядкова змінна   "
                         'задавання первинного значення
strC = Trim (strA)       'видалення пробілів ліворуч та
                         'праворуч. Результатом операції
                         'є рядок
                         'strC = "Рядкова змінна"
Dim A As Integer         'оголошення цілочислової змінної
A = 1234                 'задавання первинного значення
strC = str(A)            'перетворення числа на рядок.
                         'Результатом операції є рядок
                         'strC = "1234"
strC = "4321"            'задавання первинного значення
A = Val(strC)            'перетворення рядка на число.
                         'Результатом операції є число
                         'A = 4321
strA = "РяДОк"           'задавання первинного значення
strC = Lcase(strA)       'перетворення у нижній регістр.
                         'Результатом операції є рядок
                         'strC = "рядок"
strC = Ucase(strA)       'перетворення у верхній регістр.
                         'Результатом операції є рядок
                         'strC = "РЯДОК"
strA = "Павло Іванов"    'задавання первинного значення
strB = Replace(strA, "Іванов", "Гусев")
                         'заміна певних символів рядка.
                         'Результатом операції є рядок
                         'strC = "Павло Гусев"
Dim strA As String
Dim strB As String       'оголошення рядкових змінних
strA = "Моя рядкова змінна"
                         'задавання первинного значення
strB = Left("Моя рядкова змінна", 3)
                         'виділення 3-х символів
                         'ліворуч. Результатом операції
                         'є рядок strC = "Моя"
strB = Right(strA, 10)   'виділення 10-ти символів
```

```
                              'праворуч. Результатом операції
                              'є рядок strC = "змінна"
strB = Mid("Моя рядкова змінна", 5, 7)
                              'виділення 7-ми символів,
                              'починаючи з 5-го.
                              'Результатом операції
                              'є рядок strC = "рядкова"
strB = Mid(strA, 13)          'виділення рядка символів
                              'праворуч, починаючи з 13-го.
                              'Результатом операції
                              'є рядок strC = "змінна"
```

3.1.12. Логічні функції

Для розширення можливостей простих операторів логічного порівняння у VBA є можливість використання логічних функцій, які наведено у табл. 3.4 [3, 5].

Таблиця 3.4

Логічні функції VBA

Функція	Опис
Not	Заперечення
And	Логічне "І" (множення)
Or	Логічне "АБО" (додавання)
Xor	Заперечення "АБО"

Приклади

Результат логічного виразу **(1=1) And (2>1) => True**, оскільки обидві умови справедливі, тобто 1·0 = 0.

Результат логічного виразу **(1=1) Xor (1>2) => False**, оскільки перша умова вірна, друга – ні, тобто 1 + 0 = 1, а функція **Xor** заперечує результат.

3.1.13. Створення та застосування процедур і функцій

Процедури – це невеликі за обсягом підпрограми, що включаються до складу загальної прикладної програми [3, 5]. Процедура не повертає значень.

Синтаксис оголошення процедури:

```
Sub Ім'я_Процедури(Аргументи)
    . . . . . .
   Оператори тіла процедури
    . . . . . .
End Sub
```

де:

Ім'я_Процедури – ім'я процедури, до якого висуваються ті ж вимоги, що і до імені макросу;

Аргументи – імена параметрів із вказівкою їх типу, перераховані через кому;

Оператори тіла процедури – блок операторів процедури.

Функція – це процедура, яка повертає результуюче значення у своєму імені [3, 5].

Синтаксис оголошення функції:

```
Function Ім'я_функції (Аргументи) As тип
    . . . . . .
   Оператори тіла функції
    . . . . . .
   Ім'я_функції = значення
End Function
```

де:

Ім'я_функції – ім'я функції, до якого висуваються ті ж вимоги, що і до імені макросу;

Аргументи – імена параметрів із вказівкою їх типу, перераховані через кому;

As – ключове слово, використовуване при задаванні типу даних;

Тип – тип даних для значення, яке поверне функція.

Оператори тіла функції – блок операторів функції.

Ім'я_функції = значення – наприкінці розміщується оператор привласнення, у лівій частині якого знаходиться ім'я функції, а у правій – значення, що повертає функція.

Слід окремо зазначити, що для передавання функції або отримання від неї масивів слід застосовувати тип даних **Variant**. Цей тип даних дозволяє процедурам і функціях використовувати у якості аргументів такі складні типи даних, як масиви та записи.

Приклади

```
Sub My_Proc (X As Integer, Y As Integer)
                          'оголошення процедури, що
                          'будує коло радіусом
                          '100 пікселів на формі
                          'користувача. Аргументи: X та
                          'Y – координати центра кола.
   UserForm1.Circle(X,Y),100
                          'побудова кола на формі
                          'користувача радіусом
                          '100 пікселів
End Sub
```

Виклик процедури:

```
My_Proc (240, 230)
```

У результаті на формі користувача буде побудовано коло діаметром 200 пікселів.

```
Function My_Func (N As Integer) As Double
                          'оголошення функції, що
                          'повертає випадкове число типу
                          'Double у проміжку від 0 до N
  Dim A As Double         'оголошення змінної
  Randomize               'ініціація генератора
                          'випадкових чисел
  A = Rnd(N)              'повернення випадкового числа
                          'у проміжку від 0 до N
  My_Func = A             'повернення значення функції
End Function              'завершення функції
```

Виклик функції:

```
Dim R As Double
R = My_Func (15)
```

У результаті змінній **R** буде присвоєно випадкове число у проміжку від 0 до 15.

```
Function Rnd_V (N As Integer) As Variant
                          'оголошення функції, що
                          'повертає масив випадкових
                          'чисел типу Double у проміжку
                          'від 0 до N
Dim M (10) As Double      'оголошення масиву
  Randomize               'ініціація генератора
                          'випадкових чисел
For A = 0 To N            'оголошення масиву
  M(A) = Rnd(N)           'завантаження у масив
                          'випадкових чисел від 0 до N
Next A                    'завершення циклу
```

```
Rnd_V = M                    'повернення значення функції
End Function                 'завершення функції
```

Виклик функції:

```
Dim V (10) As Double
V = Rnd_V (15)
```

У результаті у масив **V** буде завантажено випадкові числа у проміжку від 0 до 15.

3.2. Основні оператори і функції VBA

3.2.1. Оператор переходу

Для зміни послідовності виконання операторів використовується оператор переходу [3, 5].

Синтаксис операторів переходу:

– безумовний перехід

```
GoTo мітка
```

– перехід з поверненням

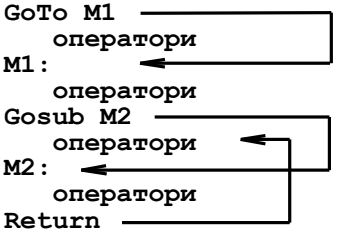
```
Gosub мітка
   оператори програми
Мітка:
   оператори програми
Return
```

де:

мітка – це послідовність букв і цифр, що починається з букви, або ціле число без знака, що завершується двокрапкою. Вимоги до назви міток такі самі, як і для імен змінних.

Заборонено використовувати оператор **GoTo** для переходу з коду по за конструкцією **For...Next** або **With...End With** на мітку, що знаходиться у цих конструкціях.

Приклади



3.2.2. Конструкція ухвалення рішень `If...Then`

Конструкція ухвалення рішень **`If...Then`** називається умовним оператором [3, 5].

Синтаксис умовного оператору:

– простий умовний оператор

```
If умова Then оператор
```

– умовний оператор з виконанням декількох операторів

```
If умова  Then
   Оператор1
   Оператор2
   .........
   ОператорN
End If
```

– умовний оператор з **`Else`**

```
If умова Then
   .........
   Оператори1
   ..........
      Else
   .........
   Оператори2
   .........
End If
```

– складна умова

```
If умова1 Then
    .........
    Оператори1
    .........
  Elself умова2 Then
    .........
    Оператори2
    .........
  ElseIf умова3 Then
    .........
    Оператори3
    .........
```

```
  Else
     .........
     Оператори N
     .........
End If
```

де:

умова – логічний вираз. Якщо значенням логічного виразу є `True`, то виконуються оператори, що знаходиться після ключового слова **Then**. Якщо значенням є `False`, то виконуються оператори, що знаходиться після ключового слова **Else.**

У логічних виразах, що використовуються для написання умови ухвалення рішення, застосовуються логічні функції, наведено у табл. 3.4 та оператори логічного порівняння, наведені у табл. 3.5. Використовуючи та комбінуючи їх, можна отримати досить складні логічні умови.

Таблиця 3.5

Оператори логічного порівняння

Оператор	Опис
=	Рівно.
<>	Не рівно.
<	Менше.
>	Більше.
<=	Менше або дорівнює.
>=	Більше або дорівнює.
Is	Ідентично.
Like	Порівняння рядків з використанням підстановлювальних символів.

Приклади

```
If (X = 9) Then X = X + 2
                        'якщо змінна X дорівнює 9, то
                        'значення змінної X
                        'збільшується на 2
If (X > 9 And X < 12) Then X = X + 1
                        'якщо змінна X знаходиться у
                        'проміжку від 9 до 12, то
                        'значення змінної X
                        'збільшується на 1
```

```
If (X > 9 And X < 12) Then
                          'якщо змінна X знаходиться у
                          'проміжку від 9 до 12, то
X = X + 3                 'значення змінної X
                          'збільшується на 3
Else                      'інакше
X = X + 2                 'значення змінної X
                          'збільшується на 2
End If                    'завершення блоку прийняття
                          'рішення
If  X < 2 Then            'якщо змінна X менше 2, то
X = X + 1 Elself X > 2 Then
                          'значення змінної X
                          'збільшується на 1, інакше,
                          'якщо змінна X більше 2, то
X = X - 1 Elself X = 2 Then
                          'значення змінної X
                          'зменшується на 1, інакше,
                          'якщо змінна X дорівнює 2, то
X = 1                     'змінній X присвоюється
                          'значення 1
End If                    'завершення блоку прийняття
                          'рішення
```

Розглянемо ще одну спрощену конструкцію використання умовного оператора:

If (умова, вираз1, вираз2)

де:

умова – логічний вираз;

вираз1, вираз2 – арифметичні, логічні вирази або рядки, які можна розглядати як вирази.

Функція **If** повертає у програму значення **виразу1** або **виразу2** залежно від того, яке значення приймає логічний вираз – `True` або `False`.

Приклад

```
strA = If(A Mod 2 = 0, "Парне", "Непарне")
                          'якщо залишок від ділення
                          'змінної A на 2 дорівнює 0,
                          'то рядковій змінній strA
                          'присвоюється значення
                          '"Парне", інакше рядковій
                          'змінній strA присвоюється
                          'значення "Непарне"
```

3.2.3. Конструкція ухвалення рішень `Select...Case`

Конструкція ухвалення рішень **`Select...Case`** називається оператором вибору [3, 5].

Синтаксис оператора вибору:

```
Select Case вираз
   Case значення1
   .........
     Оператори 1
   .........
   Case значення 2
     Оператори 2
   .........
  [Case  Else Оператори 3]
End Select
```

де:

умова – логічний вираз;

вираз – змінна, арифметичний, логічний вирази або рядки, які можна розглядати як вирази.

Приклад

```
Select Case 2*x+1          ‘якщо значення виразу 2*x+1
Case 1                     ‘дорівнює 1, то
      x = x + 1            ‘значення змінної X
                           ‘збільшується на 1
Case 2, 3, 4               ‘якщо значення виразу дорівнює
                           ‘2, 3, 4, то
      x = 10               ‘змінній X присвоюється
                           ‘значення 10
Case Else x = 20           ‘інакше змінній X присвоюється
                           ‘значення 10
End Select                 ‘завершення блоку прийняття
                           ‘рішення
```

3.2.4. Цикл `For...Next`

Цикл **`For...Next`** використовується у тому випадку, коли кількість виконань заданого блоку операторів відома заздалегідь [3, 5].

Синтаксис:

```
For лічильник = початок To кінець [Step крок]
   .........
```

```
    Оператори
    .........
Next [лічильник]
```

де:

лічильник – змінна, що використовується у ролі лічильника;

початок, **кінець** – числа або вирази, що визначають значення змінної-лічильника;

крок – крок змінювання змінної-лічильника.

Приклад

Змінна-лічильник **I** послідовно приймає значення 1, 2.5, 4, 5.5, 7 з кроком збільшення, що дорівнює 1.5. Таким чином цикл виконується 5 разів:

```
For I = 1 To 7 Step 1.5  'заголовок циклу, змінна I -
                         'лічильник циклу, приймає
                         'значення від 1 до 7 з кроком
                         '1.5.
  F = F * I              'визначення значення змінної F
Next I                   'кінець циклу
```

При негативному значенні величини **крок** цикл працює, як при позитивному значенні, але перевіряється умова **лічильник < кінець**.

Приклад

Змінна-лічильник **I** послідовно приймає значення 6, 5, 4, 3, 2, 1. Цикл виконується 6 разів.

```
For I = 6 To 1 Step – 1 'заголовок циклу, змінна I -
                        'лічильник циклу, приймає
                        'значення від 6 до 1 з кроком – 1
  F = F * I             'визначення значення змінної F
Next I                  'кінець циклу
```

У разі відсутності ключового слова **Step** крок змінювання лічильника вважається рівним одиниці.

Для дострокового завершення циклу **For...Next** використовується оператор **Exit For.**

Приклад

Змінна-лічильник **I** послідовно приймає значення 1, 2, 3,…,13 з кроком збільшення, що дорівнює 1. Цикл повторюєть-

ся 13 разів. При чому, якщо різниця **F – I = 0**, то відбуватиметься дострокове завершення циклу, щоб уникнути у подальшому ситуації ділення на 0.

```
For I = 1 To 13          'заголовок циклу, змінна I -
                         'лічильник циклу, приймає
                         'значення від 1 до 13
  If (F – I) = 0 Then Exit For
                         'якщо значення виразу (F – I)
                         'дорівнює 0, то відбувається
                         'дострокове завершення циклу
  F = F/(F – I)          'визначення значення змінної F
Next I                   'кінець циклу
```

3.2.5. Цикл While...Wend

Цикл **While...Wend** застосовується у тому випадку, коли число виконань циклу заздалегідь невідомо [3, 5].

Синтаксис:

```
While  умова
   .........
   Оператори
   .........
Wend
```

де:

умова – логічний вираз;

Робота циклу **While...Wend** починається з розрахунку значення логічного виразу **умова**. Якщо **умова =** False, то робота циклу закінчується, тобто здійснюється перехід на оператор, розташований після ключового слова **Wend**. Якщо **умова =** True, то виконуються оператори циклу. Після цього знову розраховується значення логічного виразу і цикл повторюється знову.

Приклад

```
While I <= 6             'умова роботи циклу – цикл
                         'повторюється, поки I менше
                         'або дорівнює 6
  F = F * I              'визначення значення змінної F
  I = I + 1              'визначення значення змінної I
Wend                     'кінець циклу
```

Для циклу **While...Wend** оператора дострокового виходу з циклу не існує.

3.2.6. Цикл `Do...Loop`

Цикл **`Do...Loop`** застосовується, коли число виконань операторів циклу заздалегідь невідоме. Існує чотири різновиди цієї конструкції [3, 5]:

- **`Do While...Loop`**
- **`Do Until...Loop`**
- **`Do...Loop While`**
- **`Do...Loop Until`**

Цикл **`Do While...Loop`**

Синтаксис:

```
Do While умова
   .......
   Оператори
   .......
Loop
```

Робота циклу **`Do While...Loop`** повністю аналогічна до розглянутого раніше циклу **`While...Wend.`**

Приклад

```
Do While I <= 6          'умова роботи циклу – цикл
                         'повторюється, поки I менше
                         'або дорівнює 6
  F = F * I              'визначення значення змінної F
  I = I + 1              'визначення значення змінної I
Loop                     'кінець циклу
```

Цикл **`Do Until...Loop`**

Синтаксис :

```
Do Until умова
   .......
   Оператори
   .......
Loop
```

Якщо **`умова`** = `False`, то виконуються оператори циклу. Після цього знову перевіряється умова. Якщо **`умова`** = `True`, то робота циклу завершується, тобто здійснюється перехід на оператор, розташований після ключового слова **`Loop`**.

Приклад

```
Do Until x>= 55          'умова роботи циклу – цикл
                         'повторюється, поки X менше 55
  x = x + h              'визначення значення змінної X
Loop                     'кінець циклу
```

***Цикл* Do...Loop While**

Синтаксис

```
Do
   .......
   Оператори
   .......
Loop While умова
```

Робота циклу **Do...Loop While** починається з виконання операторів циклу, після цього перевіряється **умова**. Якщо **умова** = `False`, то робота циклу завершується. Якщо **умова** = `True`, то знову виконуються оператори циклу.

Приклад

```
Do                       'початок циклу
  x = x + h              'визначення значення змінної X
Loop While x < 55        'кінець циклу з умовою роботи
                         'циклу – цикл виконується,
                         'поки значення змінної X менше
                         '55
```

***Цикл* Do...Loop Until**

Синтаксис

```
Do
   .......
   Оператори
   .......
Loop Until умова
```

Робота циклу **Do...Loop Until** починається з виконання операторів циклу. Після цього перевіряється **умова**. Якщо **умова** = `True`, то робота циклу закінчується. Якщо **умова** = `False`, то знову виконуються оператори циклу.

Приклад

```
Do                       'початок циклу
  x = x + h              'визначення значення змінної X
Loop Until x < 55        'кінець циклу з умовою роботи
                         'поки значення змінної X
                         'більше або дорівнює 55
```

У циклах **Do While...Loop** та **Do Until...Loop** існує ситуація, коли оператори циклу не виконуються жодного разу, оскільки умова завершення циклу перевіряється перед виконанням цих операторів.

У циклах **Do...Loop While** та **Do...Loop Until** оператори циклу виконуються хоча б один раз, оскільки умова завершення циклу перевіряється після виконання цих операторів.

Для дострокового завершення циклу **Do...Loop** використовується оператор **Exit Do**. Він використовується таким же чином, як і **Exit For** для циклу **For...Next**.

3.2.7. Робота з файлами

Файлом називається область на будь-якому електронному носії інформації (жорсткий диск, компакт-диск та ін.), що містить однотипну інформацію і має назву [3, 5].

Усі файли у VBA поділяються на файли послідовного доступу (текстові файли) та файли довільного доступу (файли даних).

Основними способами виводу і читання даних у VBA є:

– запис або читання даних з файлів послідовного доступу;
– запис або читання даних з файлів довільного доступу;
– запис або читання даних з робочого простору відкритого для редагування документу програм, що входять до складу пакету MS Office.

Розглянемо роботу з файлами послідовного доступу, як з найбільш простими і зручними у використанні

Перевагою таких файлів є те, що їх можна редагувати за допомогою звичайних текстових процесорів типу MS Word, WordPad та ін., не використовуючи інших спеціальних програм.

Основним недоліком збереження даних у файлах послідовного доступу є те, що числові дані зберігаються у формі символів. Крім того, такі файли завантажуються у пам'ять цілком, не розбиваючись на фрагменти. Це знижує ефективність обробления даних та ускладнює роботу з файлами великих розмірів.

Робота з файлами послідовного доступу. Послідовність роботи з такими файлами наведено на рис 3.1.

Для відкриття текстового файлу використовується оператор **Open**.

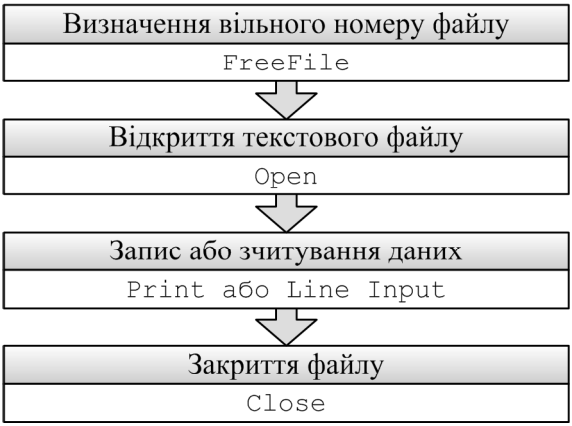


Рис. 3.1. Порядок роботи за файлами послідовного доступу

Синтаксис:

Open ім'я For призначення As номер

де:

ім'я – повне ім'я файлу (рядок, що містить ім'я файлу з шляхом і розширенням);

номер – номер файлу;

призначення – це одне з ключових слів **Input**, **Output** або **Append**;

Input – означає, що файл відкривається для зчитування з нього інформації;

Output – файл відкривається для запису інформації;

Append – файл відкривається для додавання інформації у кінець файлу.

Для визначення номера файлу використовується функція **FreeFile**, яка повертає незайнятий номер файлу. Функція повинна знаходитися перед оператором **Open**.

Синтаксис:

номер = FreeFile

Після закінчення роботи з файлом його слід закрити за допомогою оператора **Close.**

Синтаксис:

Close номер

Для додавання у файл нових рядків використовується оператор **Print**.

Синтаксис :

Print #номер, рядкова_змінна

де:

рядкова_змінна – рядкова змінна, яка завантажується у файл;

Для зчитування інформації з файлу використовується оператор **Line Input.**

Синтаксис:

Line Input #номер, рядкова_змінна

де:

рядкова_змінна – рядкова змінна, з якої завантажується рядок у файл;

Для створення нового каталогу на диску використовується оператор **MkDir.**

Синтаксис

MkDir(назва)

де:

назва – назва каталогу з розміщення на диску.

Приклад:

```
MkDir("d:\usr\texts")    'створення нового каталогу
                         'Texts на диску d
```

Для видалення файлів використовується оператор **Kill**, для видалення каталогів – оператор **RmDir**.

Синтаксис

Kill(ім'я)

де:

ім'я – повне ім'я файлу (рядок, що містить ім'я файлу з шляхом і розширенням);

Синтаксис

RmDir(назва)

де:

назва – назва каталогу, розміщеного на диску.

Приклад

```
Kill("d:\usr\texts\a.txt" )
                         'видалення файлу a.txt
                         'з каталогів \usr\texts\
                         'на диску d
RmDir("d:\usr\texts")    'видалення каталогу Texts на
                         'диску d
```

Функція **LOF** повертає кількість символів у файлі, причому символи «повернення» і «новий рядок» (якими закінчується кожен рядок) також враховуються.

Синтаксис:

кількість_символів = LOF #номер

де:

номер – номер файлу;

кількість_символів – ціла змінна, у яку завантажується кількість символів у файлі.

Функція **EOF** визначає кінець файлу під час послідовного зчитування і повертає `True` при досягненні його кінця.

Синтаксис

кінець_файлу = EOF#номер

де:

кінець_файлу – логічна змінна, у яку завантажується ознака кінця файлу.

Приклад

У прикладі створюється новий текстовий файл і у нього записується декілька рядків.

```
Dim FNamel As String
Dim s1 As String,s1 As String
Dim FNuml As Integer     'оголошення змінних
MkDir("d:\usr\texts")    'створення каталогу texts
FNamel = "d:\usr\texts\a.txt"
                         'повне ім'я файлау
FNuml = FreeFile         'визначення вільного номера
                         'файлу
Open FNamel For Output As FNuml
                         'відкриття файлу для запису
                         'інформації
s1 = "Перший рядок для запису у файл"
s2 = "Другий рядок для запису у файл"
Print #FNuml, s1         'запис рядка у файл
Print #FNuml, s2         'запис рядка у файл
Print #FNuml,"третій рядок для запису у файл"
                         'запис рядка у файл
Close FNuml              'закриття файлу a.txt
```

У прикладі відкривається файл для зчитування інформації і проводиться завантаження інформації з файлу у змінну.

```
Dim FNamel As String
Dim s1 As String,s1 As String
Dim FNuml As Integer     'оголошення змінних
```

```
FNamel = "d:\usr\texts\a.txt"
                          'повне ім'я файлу
FNuml = FreeFile          'визначення вільного номера
                          'файлу
Open FNamel For Input As FNuml
                          'відкриття файлу для зчитування
                          'інформації
LineInput #FNuml, s1      'читання рядка з файлу
LineInput #FNuml, s2      'читання рядка з файлу
Close FNuml               'закриття файлу a.txt
```

Контрольні запитання і завдання

1. Перерахуйте вимоги до тексту макросів.
2. Охарактеризуйте типи даних у мові програмування VBA.
3. Що таке змінні і як вони оголошуються?
4. Що таке константи і як вони оголошуються?
5. Що таке масиви?
6. Як оголошуються статичні масиви?
7. Як оголошуються динамічні масиви?
8. Чим відрізняються статичні та динамічні масиви?
9. Як оголошуються типи даних, визначувані користувачем?
10. Які правила має оголошення коментарів?
11. Як працюють оператори привласнення?
12. Які операції існують у мові VBA?
13. Перерахуйте основні математичні функції мови VBA.
14. Які існують функції для роботи з рядками у мові VBA?
15. Охарактеризуйте логічні функції мови VBA.
16. Дайте визначення процедури.
17. Які особливості синтаксису створення процедури та її виклику з програми?
18. Дайте визначення функції.
19. Які особливості синтаксису створення функції та її виклику з програми?
20. Які існують оператори переходу?
21. Охарактеризуйте роботу умовного оператора **If...Then**?
22. Які існують варіанти використання умовного оператора **If...Then**?
23. Опишіть використання оператора вибору **Select...Case**.
24. Яка область використання та синтаксис оператора циклу **For...Next**?

25. Яка область використання та синтаксис оператора циклу **`While...Wend`**?
26. Охарактеризуйте роботу оператора циклу **`Do...Loop`**.
27. Які існують варіанти використання оператора циклу **`Do... Loop`**?
28. Назвіть синтаксис використання різних варіантів оператора циклу **`Do...Loop`**.
29. Порівняйте різні варіанти оператора циклу **`Do...Loop`**.
30. Що називається файлом послідовного доступу?
31. Яка послідовність роботи з текстовими файлами?
32. Назвіть режими роботи з файлами послідовного доступу.
33. Назвіть послідовнсть запису рядка у файл послідовного доступу.
34. Назвіть послідовність читання рядка з файлу послідовного доступу.

4. Візуальне програмування. Створення і використання форм

4.1. Візуальне програмування

Візуальне програмування – програмування, що передбачає створення додатків за допомогою наочних засобів, тобто це спосіб створення програми для ЕОМ шляхом маніпулювання графічними об'єктами замість написання її тексту [3, 5]. При цьому програміст показує, що повинно вийти у результаті, а текст програми генерується автоматично.

Visual Basic належить до групи програмних засобів, які забезпечують користувача середовищем для візуального розроблення програм.

У систему програмування Visual Basic входить текстовий редактор для написання текстів програм. Програміст пише початкові тексти програм формалізованою мовою, яка є послідовністю команд або операторів.

Розроблення інтерфейсу програми виконується за допомогою конструктора форм. Щоб програма виконувалася, вихідні тексти перекладають на машинну мову. Це робить компілятор, який також входить у систему програмування Visual Basic.

Головна перевага візуального програмування полягає у тому, що у режимі проектування можна настроїти форму, розташувати необхідні елементи інтерфейсу майбутньої програми (кнопки, мітки, списки тощо.), не витрачаючи часу на написання програмного коду цього оформлення.

4.2. Стандартні діалогові вікна Windows

Windows надає багато стандартних діалогових вікон, які використовують, щоб запросити введення даних від користувача або відобразити інформацію. Використання діалогових вікон – рекомендований засіб для прикладної програми, щоб отримати введення даних.

Діалогове вікно (dialogbox) – тимчасове вікно, яке створює прикладна програма, щоб вивести інформацію або отримати дані, що вводяться користувачем. Діалогове вікно зазвичай міс-

тить один або більшу кількість елементів управління, за допомогою яких користувач отримує інформацію або вводить дані.

Розрізняють два види діалогових вікон: вікна введення інформації і вікна виведення інформації (вікна повідомлень).

4.2.1. Діалогові вікна повідомлень

Діалогові вікна у формі повідомлень створюються за допомогою функції **MsgBox()** [3, 5]. Такі вікна містять текст повідомлення, відповідно до змісту якого користувач повинен обрати один з декількох (не більше трьох) варіантів дій, пропонованих програмою. Вікно відображає текст повідомлення, а потім чекає, поки користувач натисне одну з командних кнопок.

Синтаксис використання функції:

MsgBox(Повідомлення, Кнопки, Титульний_рядок)

де:

Повідомлення – текст повідомлення, який є строковою змінною. Цей аргумент є обов'язковим;

Кнопки – цифровий код, відповідно до якого у діалоговому вікні міститимуться різні набори командних кнопок і застосовуватимуться різні види візуального оформлення. Цей аргумент є необов'язковим;

Титульний_Рядок – містить текст, який буде відображений у титульному рядку діалогового вікна. Аргумент також є необов'язковим.

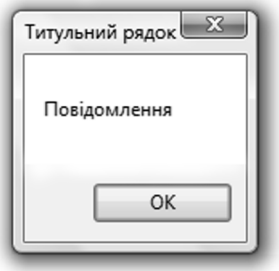


Рис. 4.1. Просте діалогове вікно повідомлення

Приклад

Просте діалогове вікно, що створюється функцією **MsgBox()** (рис. 4.1), містить текст повідомлення, титульний рядок і командну кнопку «**OK**». Користувач повинен натиснути цю кнопку для підтвердження прийому повідомлення і закриття діалогового вікна. вікна.

```
MsgBox "Повідомлення", ,"Титульний рядок"
```

Додавання командних кнопок і деяке візуальне перетворення вікна повідомлення можливе, якщо задавати необов'язкові аргументи самостійно. Для цього використовуються коди активності за замовчуванням (табл. 4.1).

Таблиця 4.1

Коди завдання командних кнопок і піктограм

Код	Ескіз	Опис
Коди командних кнопок		
0	OK	Кнопка **OK**
1	OK Отмена	Кнопки **OK** і **Cancel**
2	Прервать Повтор Пропустить	Кнопки **Abort**, **Retry** та **Ignore**
3	Да Нет Отмена	Кнопки **Yes**, **No** та **Cancel**
4	Да Нет	Кнопки **Yes**, **No**
5	Повтор Отмена	Кнопки **Retry**, **Cancel**
Коди активності за замовчуванням		
0	Да Нет Отмена	Активна перша кнопка
256	Да Нет Отмена	Активна друга кнопка
512	Да Нет Отмена	Активна третя кнопка
Коди піктограм		
16		Помилка
32		Підтвердження
48		Увага
64		Інформація

Якщо передбачається, що діалогове вікно міститиме більше за одну кнопку, слід визначитися, яка з них уважатиметься активною, тобто буде виділятися, підказуючи бажаний варіант розвитку дій.

Остаточний код кнопки є сумою усіх використаних кодів наборів кнопок, їх активності та кодів піктограм.

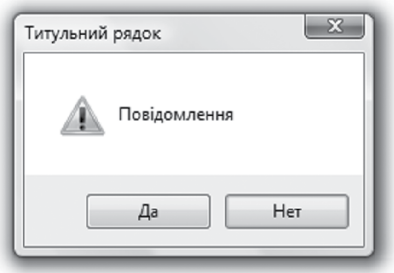


Рис. 4.2. Вікно повідомлення

Приклад

Вікно повідомлення (рис. 4.2) містить кнопки **Да** (**Yes**), **Нет** (**No**) (код 4), піктограму **Увага** (код 48). З двох кнопок активна друга кнопка (код 256). Таким чином, фінальний код: 4 + 48 + 256 = 308.

```
MsgBox "Повідомлення",308,"Титульний рядок"
```

Коли у діалоговому вікні міститься більше за одну кнопку, то після того, як користувач натиснув на одну з них, у програму має бути переданий код цієї кнопки (табл. 4.2).

Таблиця 4.2

Значення, що повертаються функцією `MsgBox()`

Код	Командна кнопка
1	**OK**
2	**Cancel**
3	**Abort**
4	**Retry**
5	**Ignore**
6	**Yes**
7	**No**

Приклад

```
Dim Ret As String
Ret = MsgBox "Повідомлення",308,"Титульний рядок"
```

У наведеному прикладі після того як користувач натиснув одну з двох кнопок, функція `MsgBox()` повертає код натиснутої кнопки.

4.2.2. Діалогові вікна введення інформації

Діалогові вікна для введення інформації створюються за допомогою функції **InputBox()** [3, 5]. Такі вікна містять поле для введення і редагування тексту, командні кнопки **ОК** і **Cancel**.

Синтаксис використання функції:

InputBox(Повідомлення,Титульний_Рядок,Значення)

де:

Повідомлення – текст запиту на введення, який є рядковою змінною. Цей аргумент є обов'язковим;

Титульний_Рядок – містить текст, який буде відображено у титульному рядку діалогового вікна. Аргумент є необов'язковим;

Значення – текст, який буде знаходитися у полі введення тексту при відкритті діалогового вікна як підказка. Аргумент є необов'язковим.

Будь-який текст, уведений у текстове поле діалогового вікна, буде повернений функцією **InputBox()** після того, як користувач натисне командну кнопку **ОК**. Якщо ж користувач натисне командну кнопку **Cancel**, то буде повернений порожній рядок, тобто "".

Уведені користувачем значення, що повертає функція, є рядковими змінними.

Приклад

Вікно введення (рис. 4.3), що створюється функцією **InputBox()**, містить текст повідомлення "Уведіть ПІБ користувача", титульний рядок "ПІБ користувача" і текст при відкритті діалогового вікна як підказка "Іванов Іван Іванович".

Також вікно містить кнопки **ОК** і **Cancel**. Користувач повинен увести потрібне значення і натиснути одну з кнопок для закриття діалогового вікна. Функція **InputBox()** повертає змінній **Ret** рядок, уведений користувачем. Якщо користувач нічого не ввів, функція повертає порожню строку "".

```
Dim Ret As String
Ret = InputBox("Уведіть ПІБ користувача","ПІБ користувача","Іванов Іван Іванович")
```

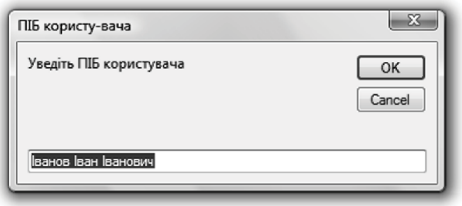


Рис. 4.3. Діалогове вікно введення інформації

4.3. Створення і використання форм

4.3.1. Поняття форми

Інтерфейсом користувача програми називається засіб спілкування користувача з програмою.

Для створення інтерфейсу, VBA надає готові об'єкти, що мають певні властивості, які можна змінювати та налаштовувати на виконання конкретного завдання.

Для створення засобів спілкування з користувачем у програму включаються форми, які є типовими вікнами Windows.

Форма – це головний елемент програми, на якому розташовуються елементи управління [3, 5]. По суті, форма – це вікно, наповнене елементами управління.

Форма (рис. 4.4) має рядок заголовка з кнопками управління і системним меню. Порожня поверхня форми (сірий фон, покритий сіткою) заповнюється у процесі проектування.

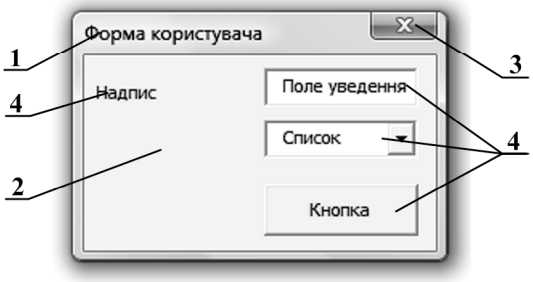


Рис. 4.4. Приклад форм користувача: *1* – кнопка закриття; *2* – заголовок форми; *3* – робоча область; *4* – елементи управління

Кожна форма і кожен елемент управління має своє унікальне ім'я, за яким до них можна звертатися. Імена за замовчуванням даються системою при створенні об'єкта і складаються зі слова, що означає тип об'єкта і цифри – порядкового номера. Наприклад, перша форма у проекті має ім'я **`UserForm1`**.

Ім'я об'єкта повинне:

– починатися з букви;
– не містити точки;
– бути не більше 255 символів;
– не збігатися з операторами і ключовими словами VBA;
– бути унікальним у межах зони видимості.

Для створення форми користувача застосовується конструктор форм (рис. 4.5). Для додання форми користувача у проект необхідно вибрати інструмент **`UserForm`** (*7*) з панелі інструментів **Standard**. Буде запущено конструктор форм і створена порожня форма користувача (*6*).

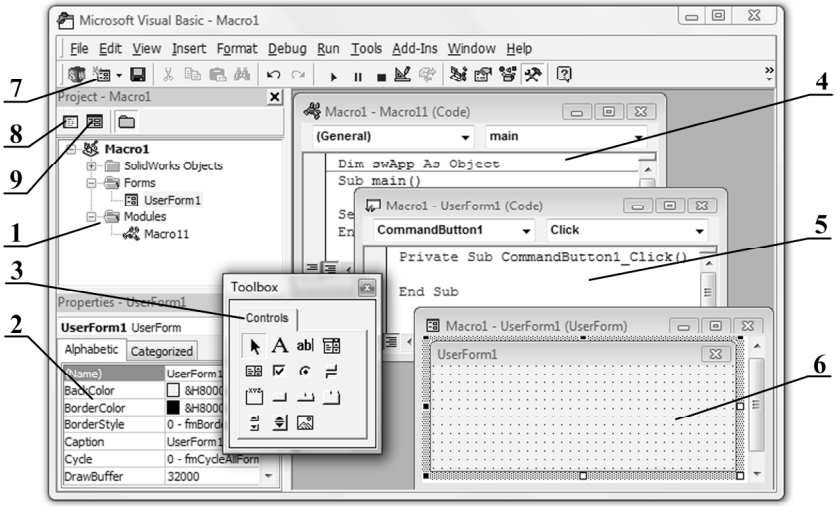


Рис. 4.5. Режим конструктора форм: *1* – менеджер проекту; *2* – менеджер властивостей; *3* – панель елементів управління; *4* – вікно коду головної програми; *5* – вікно коду подій форми та елементів управління; *6* – форма користувача; *7* – інструмент **`UserForm`**; *8* – швидкий перхід у вікно подій форми та елементів управління; *9* – швидкий перхід у вікно форми користувача

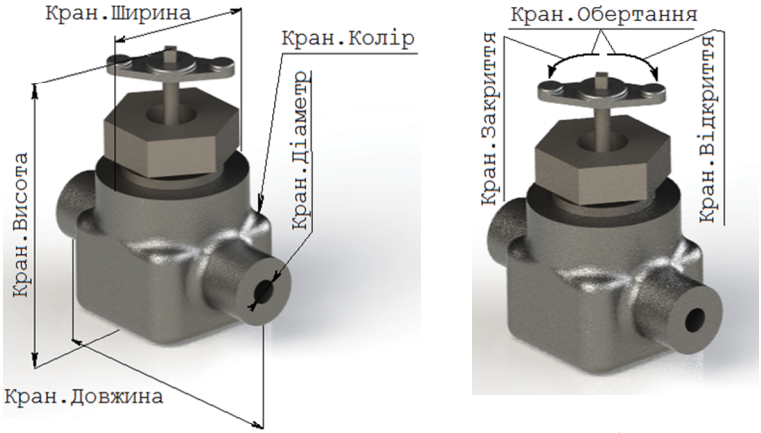
Кожна форма та елемент управління є об'єктом і мають властивості, методи та події.

Властивості – це параметри форми або елемента.

Методи – це операції, які можна виконувати над формою або елементом.

Події – це дії, які повинна виконати програма при взаємодії користувача з формою або елементом керування.

Розглянемо поняття властивостей, методів і подій на прикладі крана (рис 4.6).



а *б*

Рис. 4.6. Властивості, методи, події об'єкта: *а* – властивості; *б* – методи та подія

Властивості крана включають видимі атрибути типу висоти, ширини, довжини, діаметра патрубків, кольору. Інші властивості описують його стан (увімкнений або вимкнений) або атрибути, які невидимі, наприклад, матеріал деталей.

За визначенням, усі крани мають ці властивості, а параметри налаштування цих властивостей можуть відрізняти один кран від іншого.

Кран також має методи властивостей або дії, які він міг би виконати. Це були б метод **`Відкриття`** (дія обертання штока за годинниковою стрілкою і відкриття отвору), метод **`Закриття`** (дія обертання штока за протигодинниковою стрілкою і закриття отвору). Отже, усі крани мають ці методи.

Крани також мають задані відповіді на деякі зовнішні події. Наприклад, кран відповів би на подію **Обертання**, відкривши або закривши отвір.

Якби можна було запрограмувати кран, то код установлення властивостей крану VBA міг би мати такий вид:

```
Кран.Довжина = 120
Кран.Ширина = 60
Кран.Висота = 90
Кран.Колір = сірий
Кран.Матеріал = сталь 45
```

Розглянемо синтаксис цього коду. Об'єкт (**Кран**) супроводжується властивістю (**Довжина**), яка супроводжується присвоєнням значення (**= 120**). Так можна змінити довжину крана. Властивості можуть бути також встановлені у вікні властивостей під час розроблення програми.

Методи крана могли б викликатися приблизно так:

```
Кран.Відкриття
Кран.Закриття
```

Синтаксис методу подібний до синтаксису властивості. Об'єкт (**Кран**) супроводжується методом (**Відкриття**). Деякі методи матимуть один або декілька аргументів, щоб потім описати дію, яку буде виконано.

Кран міг би відповісти на подію у такий спосіб:

```
Sub Кран.Обертання( Напрям )
  If Напрям = за годинниковою стрілкою Then
    Кран.Відкриття
    Кран.Відкритий = Так
  Else
    Кран.Закриття
    Кран.Відкритий = Ні
  End If
End Sub
```

У цьому випадку код описує роботу крана, коли відбувається подія **Обертання**. При цьому викликається метод **Відкриття**, або **Закриття** залежно від напряму обертання. Оскільки змінюється стан крана, властивість **Відкритий** встановлюється `Так` або `Ні`.

Таким чином, можна вирішити, які властивості потрібно змінити, які викликані методи або події будуть відповідати тому, щоб досягти бажаного виду і роботи програми.

4.3.2. Властивості форми

Форма та елементи керування мають багато властивостей [3, 5]. Розглянемо деякі властивості форм (табл. 4.3).

Таблиція 4.3

Основні властивості форми

Властивість	Тип	Призначення
1	2	3
BackColor	long	Колір фону форми. Колір визначається шістнадцятковим числом або спеціальною константою VBA (див. табл. 4.4).
BorderStyle	integer	Тип межі, що обрамляє форму. 0 – немає рамки, кнопок розгорнути / відновити, згорнути і вихід у правій частині заголовка, а також віконного меню; 1 – форму можна згорнути і розгорнути, але не можна змінити розмір форми та перетягати її екраном за краї. Доступні кнопки розгорнути/відновити, згорнути і вихід у правій частині заголовка.
Caption	string	Текст, який відображається у заголовку форми.
ControlBox	boolean	Відповідає за присутність на формі віконного меню. `True` – увімкнути іконку віконного меню; `False` – вимкнути іконку віконного меню.
FillColor	long	Колір заливки форми (див. табл. 4.4).
FillStyle	integer	Тип заливки форми кольором. 0 – суцільне заповнення; 1 – немає заповнення; 2 – горизонтальне штрихування; 3 – вертикальне штрихування; 4 – штрихування по діагоналі зліва направо; 5 – штрихування по діагоналі справа наліво; 6 – горизонтально-вертикальне штрихування; 7 – штрихування по діагоналі в обох напрямах.

Продовження табл. 4.3

1	2	3
Font	variant	Тип шрифту. Обирається зі списку шрифтів системи. Визначається декількома параметрами.
Height	integer	Висота форми у пікселах.
Left	integer	Відстань від лівого краю форми до лівого краю екрана. Використовується у поєднанні з властивістю **Top** і визначає розташування форми на екрані.
Name	string	Унікальне ім'я форми або будь-якого елементу керування. За замовчуванням, якщо додавати нові форми у проект, їм привласнюються імена, що починаються з UserForm + "номер" (UserForm1, UserForm2, UserForm3).
Picture	Variant	Зображення, що виводиться на форму з графічного файлу.
StartUpPosition	integer	Розташування форми при завантаженні. 0 – положення форми на екрані, задається властивостями **Left** і **Top**; 1 – форма розташовується у центрі робочого столу, дочірня – у центрі батьківської; 2 – форма розташовується у центрі екрана; 3 – положення форми задається системою, виходячи з кількості відкритих вікон і розташування їх на екрані.
Top	integer	Відстань від верхнього краю форми до верхнього краю екрана. Використовується у поєднанні з властивістю **Left** і визначає розташування форми на екрані.
Visible	boolean	Видимість форми. True – форма видима; False – форма невидима.
Width	integer	Ширина форми у пікселах.
WindowState	integer	Стан форми при запуску. 0 – звичайний стан форми; 1 – при запуску форма буде згорнута; 2 – при запуску форма буде розгорнута на весь екран.

Таблиця 4.4

Константи кольору

Колір	Шістнадцяткове число	Константа VBA
Чорний	&0000000	vbBlack
Синій	&HFF0000	vbBlue
Бірюзовий	&HFFFF00	vbCyan
Зелений	&HFF0000	vbGreen
Пурпурний	&HFF00FF	vbMagenta
Червоний	&HFF0000	vbRed
Білий	&HFFFFFF	vbWhite
Жовтий	&HFFFF00	vbYellow

4.3.3. Методи форми

Розглянемо деякі методи роботи з формою [3, 5].

Circle дозволяє намалювати еліпс, коло, дугу або сектор на формі. При цьому властивість форми **AutoRedraw** потрібно встановити у True.

Синтаксис:

```
UserForm.Circle [Step] (X, Y), радіус, [Колір], [Початок], [Кінець], [Відношення]
```

У круглих дужках вказані обов'язкові параметри, у квадратних – необов'язкові.

Приклади

```
UserForm1.Circle (1000,1000), 500, vbRed
                          'будується коло червоного
                          'кольору з радіусом 500
UserForm1.Circle (1000, 1000), 500, vbRed,,, 0.5
                          'будується еліпс з меншою віссю
                          'по вертикалі
UserForm1.Circle(1000,1000),500,vbRed,,3.14/2,3.14
                          'будується дуга
```

Cls – очищення форми. Очищає графічний зміст вікна програми.

Hide – видаляє форму з екрана, але не вивантажує з пам'яті.

Line – дозволяє намалювати лінію або прямокутник. Властивість форми **AutoRedraw** потрібно встановити у `True`.

Синтаксис:

```
UserForm1.Line [Step] (X1, Y1) [Step] – (X2, Y2), [Колір], [F]
```

У круглих дужках вказані обов'язкові параметри, у квадратних – необов'язкові.

Приклади

```
UserForm1.Line (1000, 1000) –(2000, 2000), vbBlue
                        'малює лінію синього кольору
UserForm1.Line (500, 500) –(4000, 2000)
                        'малює лінію тим кольором, який
                        'встановлений у властивості
                        'ForeColor форми
UserForm1.Line (1000, 1000) –(2000, 2000),vbRed,BF
                        'малює квадрат, залитий
                        'червоним кольором
```

Move – дозволяє перемістити форму.

Синтаксис:

```
UserForm1.Move [Лівий край], [Верхній край], [Ширина], [Висота]
```

У круглих дужках вказані обов'язкові параметри, у квадратних – необов'язкові.

Приклади

```
UserForm1.Move 2000, 2000, 5000, 5000
                        'переміщає форму, змінюючи її
                        'розмір
UserForm1.Move 100, 7000
                        'переміщає форму, не змінюючи
                        'її розмірів
```

Point – повертає значення кольору вибраної точки. Якщо координати випадуть за межі форми, повертає – `1`.

Синтаксис:

```
Ret = UserForm1.Point (X, Y)
```

Приклад

```
PointColor = Form1.Point(2, 2)
```

Print – друкує текст на формі. Властивість форми **AutoRedraw** потрібно встановити у `True`.

Приклад

```
UserForm1.Print "Текст на формі".
```

Pset – малює точки вказаного кольору у заданому місці. За товщину об'єкта відповідає властивість розмір пензля **DrawWidth**.

Синтаксис:

```
UserForm1.PSet [Step] (X, Y)[,Колір]
```

Приклад

```
UserForm1.DrawWidth = 10
                        'встановлення розміру пензля у
                        '10 пікселів
UserForm1.PSet (100, 100), vbRed
                        'малює точку червоного кольору
                        'розміром 10 пікселів
```

Refresh – оновлює форму.

Show – завантажує або показує форму на екрані, якщо вона не завантажена у пам'ять, то метод дозволяє завантажити її.

Unload – вивантажує форму і видаляє її з пам'яті.

4.3.4. Події форми

Після встановлення властивостей форми або елементів управління, переходять до написання коду, пов'язаного з ними. Особливе місце у цьому коді займають процедури оброблення подій. У кожної форми та елементу управління є цілий набір подій, що обробляють ті або інші дії користувача або системи.

Кожна подія форми або елемента управління являє собою процедуру, що запускається при встановленій дії користувача або системи.

Код оброблення події записується у вікні модуля (рис 4.7). Для створення процедури оброблення подій потрібно у вікні коду обрати зі списку (*2*) форму або потрібний елемент управління, потім зі списку подій (*3*) обрати потрібну подію.

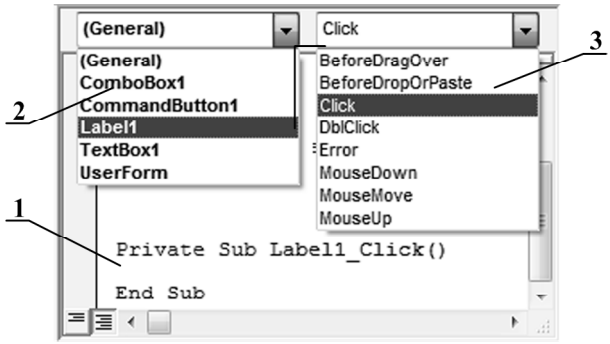


Рис. 4.7. Вікно коду оброблення подій: *1* – код процедури оброблення події; *2* – список елементів управління; *3* – список можливих подій форми або елементів управління

Розглянемо деякі події форми [3, 5].

`Activate` – відбувається, коли форма стає активною, тобто отримує фокус. Спрацьовує лише при перемиканні між формами проекту. Відбувається кожного разу при активації форми.

`Click` – відбувається при натисненні на ліву або праву кнопку миші, коли курсор знаходиться над вільним місцем форми.

`DblClick` – відбувається при подвійному натисненні лівої або правої кнопки миші, коли курсор знаходиться над вільним місцем форми. Причому подія **`Click`** також відбувається.

`Deactivate` – подія відбувається у тому випадку, коли форма, протилежно події **`Activate`**, перестає бути активною. Виникає при втраті фокусу. Спрацьовує лише при перемиканні між формами проекту.

`DragDrop` – відбувається при завершенні операції перетягання у той момент, коли елемент скидається на форму.

`DragOver` – відбувається, коли об'єкт-джерело знаходиться над об'єктом-одержувачем, але кнопка ще не відпущена.

`GotFocus` – відбувається при отриманні формою фокусу при натисканні клавіші **Tab** або за допомогою методу **`SetFocus`**.

`Initialize` – відбувається при створенні форми, причому відбувається один раз до завантаження форми, під час конфігурації.

`KeyDown` – відбувається, коли форма має фокус і користувач натискає клавішу на клавіатурі.

KeyPress – відбувається при натисненні й утримуванні клавіші. Причому момент утримання відбувається дуже швидко, майже миттєво. За допомогою цієї події можна отримати код натиснутої клавіші.

KeyUp – відбувається, коли форма має фокус і користувач відпускає клавішу на клавіатурі.

Load – завантаження форми у пам'ять до її появи на екрані. Цю подію має тільки форма. Форма може бути вивантажена і наново завантажена у ході виконання додатка скільки завгодно раз. Отже, подія може виникати за бажанням розробника кілька разів.

LostFocus – відбувається, коли форма втрачає фокус.

MouseDown – відбувається під час натиснення лівої або правої кнопки миші на формі.

MouseMove – відбувається при переміщенні курсору миші над формою.

MouseUp – відбувається лише тоді, коли користувач після натиснення на формі відпускає клавішу миші**.**

Події **MouseDown**, **MouseMove** і **MouseUp** мають однакові параметри:

```
Form_MouseDown (Button As Integer, Shift As Integer,
X As Single, Y As Single)
Form_MouseMove (Button As Integer, Shift As Integer,
X As Single, Y As Single)
Form_MouseUp (Button As Integer, Shift As Integer,
X As Single, Y As Single)
```

де:

Button – містить номер натиснутої кнопки миші:
1 – натиснуто ліву кнопку миші;
2 – натиснуто праву кнопку миші;
4 – натиснуто середню кнопку миші.

Shift – містить інформацію про натиснення кнопок **ALT**, **CTRL** і **SHIFT** при натисканні кнопки миші:
1 – натиснуто кнопку **SHIFT**;
2 – натиснуто кнопку **CTRL**;
4 – натиснуто кнопку **ALT**.

X, Y – координати покажчика миші, де сталася подія.

QueryUnload – відбувається перед подією **Unload** перед закриттям форми. Параметр **UnloadMode** містить значення, з якої причини відбувається закриття форми:

0 – після натиснення кнопки **Закрити** у віконному меню, у заголовку форми;
1 – у коді додатка викликана подія **`Unload`**;
2 – після завершення Windows;
3 – після натиснення **CTRL**+**ALT**+**DEL**;
4 – після закриття головної форми, закривається дочірня.

`Resize` – відбувається при зміні розмірів форми. Можна використовувати для масштабування елементів управління при зміні розміру форми.

`Terminate` – виконується самою останньою подією, після **`Unload`**. Цю подію має тільки форма.

`Unload` – подія, що відбувається при закритті та вивантаженні форми і видалення її з пам'яті. Цю подію має тільки форма.

4.4. Базові компоненти форми

Компоненти форми є елементами управління. Доступні компоненти форми знаходяться на панелі **Toolbox**.

Усі компоненти мають властивості, методи і події, схожі з властивостями, методами і подіями форм.

Коротко розглянемо базові компоненти (рис. 4.8, 4.9) [3, 5].

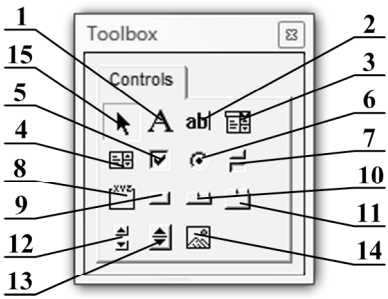


Рис. 4.8. Елементи управляння на панелі **Toolbox**: *1* – надпис **`Label`**; *2* – поле для введення **`TextBox`**; *3* – віпидаючий список **`ComboBox`**; *4* – список **`ListBox`**; *5* – позначка **`CheckBox`**; *6* – позначка **`OptionBox`**; *7* – кнопка **`ToggleButton`**; *8* – рамка **`Frame`**; *9* – кнопка **`CommandButton`**; *10* – сторінки **`TabStrip`**; *11* – сторінки **`MultiPage`**; *12* – стрічка прокручування **`ScrollBar`**; *13* – кнопки прокручування **`SpinButton`**; *14* – рисунок **`Image`**; *15* – режим вибору об'єктів

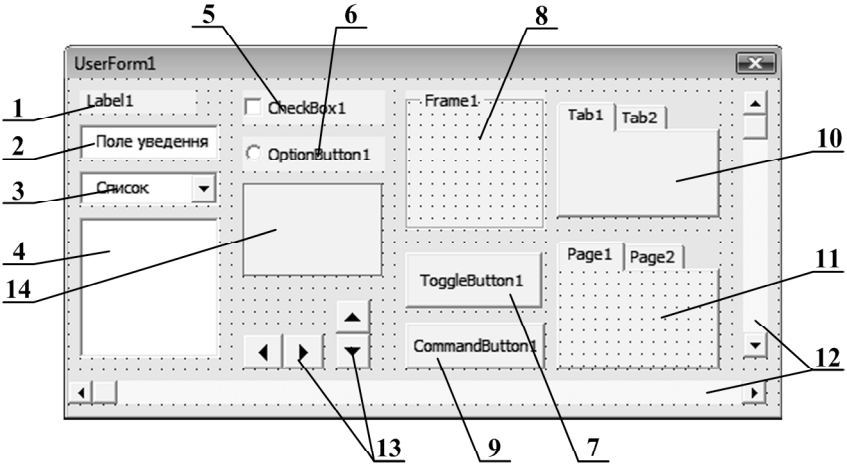


Рис. 4.9. Зовнішній вигляд елементів управління на формі користувача: *1* – надпис **Label**; *2* – поле для введення **TextBox**; *3* – випадаючий список **ComboBox**; *4* – список **ListBox**; *5* – позначка **CheckBox**; *6* – позначка **OptionBox**; *7* – кнопка **ToggleButton**; *8* – рамка **Frame**; *9* – кнопка **CommandButton**; *10* – сторінки **TabStrip**; *11* – сторінки **MultiPage**; *12* – стрічка прокручування **ScrollBar**; *13* – кнопки прокручування **SpinButton**; *14* – рисунок **Image**

4.4.1. Елемент управління Label

Label – пояснювальний надпис на формі, який не може бути змінений користувачем, але текстом напису можна управляти програмно. Зазвичай використовується для створення пояснювальних надписів для елементів управління, які не мають властивості **Caption** або для розміщенні на формі описового чи довідкового тексту. Основні властивості елементу управління **Label** наведено у табл. 4.5.

За необхідності цей елемент можна перетворити на спрощену кнопку, створивши оброблювач події **Click** натискання миші на надпис.

Приклад

```
Label1.Caption = "Текст пояснення"
                          'задавання рядка надпису
```

Таблиця 4.5

Основні властивості елементу управління `Label`

Властивість	Тип	Призначення
`Alignment`	`integer`	Визначає вирівнювання тексту надпису: 0 – за замовчуванням; 1 – текст по центру; 2 – текст праворуч.
`AutoSize`	`boolean`	`True` – встановлюється режим автоматичної зміни розміру поля так, щоб увесь текст, що вводиться, поміщався у ньому; `False` – встановлюється фіксований розмір поля.
`Caption`	`string`	Текст, що відображається у написі довжиною максимум 1024 байт.
`ForeColor`	`long`	Колір тексту надпису
`Multiline`	`boolean`	`True` – встановлюється багаторядковий режим уведення тексту у полі; `False` – однорядковий режим.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.
`WordWrap`	`boolean`	`True` – встановлюється режим автоматичного перенесення; `False` – текст надпису не переноситься.

4.4.2. Елемент управління `TextBox`

`TextBox` – поле для введення/виведення інформації. Використовується в основному для введення користувачем тексту, доступного для редагування.

Основні властивості елементу управління **`TextBox`** наведено у табл. 4.6.

Елемент управління має додаткову подію:
`Change` – відбувається при введенні або зміні тексту в полі.

Приклад

```
TextBox1.Text = "Надпис"  'виведення тексту у полі
                          'введення
S1=TextBox1.Text          'завантаження у змінну тексту з
                          'поля
```

Таблиця 4.6

Основні властивості елементу управління `TextBox`

Властивість	Тип	Призначення
`Alignment`	`integer`	Визначає вирівнювання тексту надпису. 0 – за замовчуванням; 1 – текст по центру; 2 – текст праворуч.
`AutoSize`	`boolean`	`True` – встановлюється режим автоматичної зміни розміру поля так, щоб увесь текст, що вводиться, поміщався у ньому; `False` – встановлюється фіксований розмір поля.
`BackColor`	`long`	Визначає колір фону введення.
`BorderColor`	`long`	Визначає колір рамки поля введення.
`Enabled`	`boolean`	`True` – редагування тексту у полі введення дозволене; `False` – редагування тексту у полі введення заборонене.
`ForeColor`	`long`	Колір тексту надпису.
`MaxLenght`	`integer`	Встановлює максимально допустиму кількість символів, що вводяться у поле. Якщо ця властивість дорівнює 0, то немає обмежень на кількість символів, що вводиться.
`Multiline`	`boolean`	`True` – встановлюється багаторядковий режим уведення тексту у полі; `False` – однорядковий режим.
`ScrollBars`	`integer`	Встановлює режим відображення у полі смуг прокрутки: 0 – не відображати смуги прокрутки; 1 – відображати горизонтальну смугу прокрутки; 2 – відображати вертикальну смугу прокрутки; 3 – відображати горизонтальну і вертикальну смуги прокрутки.
`Text`	`integer`	Текст, що введено відображається у полі напису довжиною максимум 2048 символів. Якщо властивість **`MultiLine =`** `True`, то поле може містити до 32Кб тексту.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.
`WordWrap`	`boolean`	`True` – встановлюється режим автоматичного перенесення; `False` – текст надпису не переноситься.

4.4.3. Елемент управління ComboBox

ComboBox – комбінований список на формі, який дає змогу обирати готові значення зі списку або вводити власні дані, як у елементі управління **TextBox**. У нього відсутній режим виділення декількох елементів списку.

Основні властивості елементу управління **ComboBox** наведено у табл. 4.7.

Таблиця 4.7

Основні властивості елементу управління ComboBox

Властивість	Тип	Призначення
AddItem	string	Додавання рядка тексту до списку.
BackColor	long	Визначає колір фону введення.
BorderColor	long	Визначає колір рамки поля введення.
BorderStyle	integer	Визначає тип рамки поля; 0 – без лінії обрамлення; 1 – рамка з одинарною лінією.
Clear	–	Повне очищення списку з видаленням усіх записів.
Enabled	boolean	True – дозволене редагування тексту в полі введення; False – забороняє редагування тексту в полі введення.
ForeColor	long	Колір тексту надпису.
ListRows	integer	Кількість елементів, які відображаються у списку, що розкривається.
MatchRequired	boolean	True – відключення можливості введення і додавання нових записів; False – включення можливості введення і додавання нових записів.
MaxLenght	integer	Встановлює максимально допустиму кількість символів, що вводяться у поле. Якщо ця властивість дорівнює 0, то немає обмежень на кількість символів, що вводиться.
Text	string	Текст, що обрано зі списку.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

Елемент управління **ComboBox** має додаткову подію: **Change** – відбувається при вибору іншого запису зі списку або введенні тексту.

Приклади

```
ComboBox1.AddItem = "Рядок 1"
ComboBox1.AddItem = "Рядок 2"
ComboBox1.AddItem = "Рядок 3"
ComboBox1.AddItem = "Рядок 4"
                          'завантаження записів до списку
ComboBox1.Text = "Рядок 1"
                          'завантаження запису за
                          'замовчуванням
S1 = ComboBox1.Text       'завантаження обраного рядка у
                          'змінну
```

4.4.4. Елемент управління ListBox

ListBox – список, що дозволяє користувачеві обрати значення зі списку даних. Якщо число елементів у списку перевищує число пунктів, яке список може вивести на екран, то у нього автоматично додається смуга прокрутки.

Основні властивості елементу управління **ListBox** наведено у табл. 4.8.

Таблиця 4.8

Основні властивості елементу управління ListBox

Властивість	Тип	Призначення
1	2	3
AddItem str [,i]	string	Додавання запису до списку. **str** – рядковий вираз для додавання у список; **i** (integer) – визначає, у яке місце списку повинен бути вставлений новий пункт. Якщо **i =** 0, запис встановлюється у першу позицію списку. Якщо значення **i** не вказано, запис вставляється у кінець списку.
BackColor	long	Визначає колір фону введення.
BorderColor	long	Визначає колір рамки поля введення.
BorderStyle	integer	Визначає тип рамки поля: 0 – без лінії обрамлення; 1 – рамка з одинарною лінією.

Продовження табл. 4.8

1	2	3
Clear	–	Повне очищення списку з видаленням усіх записів.
Enabled	boolean	True – обирати записи у списку дозволено; False – обирати записи у списку заборонено.
ForeColor	long	Колір тексту списку.
List (i)	integer	Забезпечує доступ (запис або зчитування) до окремих пунктів у списку.
ListCount	integer	Повертає кількість пунктів у списку.
ListIndex	integer	Повертає або встановлює позицію обраного пункту списку. Якщо обраних елементів немає, то значення властивості **ListIndex** = –1. Перший пункт списку має значення **ListIndex** = 0, а значення властивості **ListCount** завжди більше на одиницю від найбільшого значення **ListIndex**.
ListStyle	integer	Тип виокремлення елементу списку: fmListStyleOption – виокремлений елемент виділяється відміткою; fmListStylePlain – виокремленний елемент виділяється кольором.
RemoveItem i	integer	**i** – визначає номер запису, що потрібно видалити.
Sorted	boolean	True – включення сортування списку за алфавітом. Сортування не враховує регістр символів. Використання у цьому випадку методу **AddItem** з аргументом індексу може привести до непередбачуваних, несортованих результатів; False – відключення сортування списку за алфавітом.
Text	string	Містить пункт, вибраний у даний момент зі списку.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

Приклад

```
ListBox1.AddItem "Рядок1"
ListBox1.AddItem "Рядок2"
ListBox1.AddItem "Рядок3"
                              'додавання рядків у список
ListBox1.RemoveItem 1         'видалення 2-го рядка зі списку
```

4.4.5. Елемент управління `CommandButton`

`CommandButton` – це звичайна кнопка управління, яка застосовується для виконання команд. При натисненні на кнопку виконується процедура, визначена подією **`Click`**.

Основні властивості елементу управління **`CommandButton`** наведено у табл. 4.9.

Таблиця 4.9

Основні властивості елементу управління `CommandButton`

Властивість	Тип	Призначення
`BackColor`	`long`	Визначає колір фону введення.
`Caption`	`string`	Текст, що відображається на кнопці.
`DisabledPicture`	`variant`	Зображення, яке буде на кнопці, поки вона не доступна. Для розташування зображення необхідно встановити властивість **`Style =`** `Graphical`.
`DownPicture`	`variant`	Зображення, яке буде на кнопці, поки вона натиснута. Для розташування зображення необхідно встановити властивість **`Style =`** `Graphical`.
`Enabled`	`boolean`	`True` – кнопку розблоковано; `False` – кнопку заблоковано.
`ForeColor`	`long`	Колір тексту списку.
`Picture`	`variant`	Зображення, яке розташовуватиметься на кнопці у графічному стані. Для розташування зображення, необхідно встановити властивість **`Style =`** `Graphical`.
`Style`	`integer`	Стиль відображення кнопки: 0 – `Standart` – звичайний; 1 – `Graphical` – графічний.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.

Основні події елементу **`CommandButton`**:

`Click` – основна подія кнопки, яка відбувається при натисненні на неї.

`KeyDown` – відбувається, коли кнопка має фокус і користувач натискає клавішу на клавіатурі.

KeyPress – відбувається, коли користувач натиснув і відпустив клавішу, можна отримати код натиснутої клавіші.

KeyUp – відбувається, коли кнопка має фокус і користувач відпускає клавішу на клавіатурі.

MouseDown – відбувається під час натиснення мишею на кнопку.

MouseUp – відбувається лише тоді, коли користувач після натиснення, відпускає клавішу миші. Краще використовувати замість події **Click**, оскільки після натиснення користувач може передумати і захоче скасувати дію.

Події **MouseDown**, **MouseMove** і **MouseUp** мають однакові параметри:

```
Command1_MouseDown(Button As Integer,
 Shift As Integer,
 X As Single,
 Y As Single)
Command1_MouseMove(Button As Integer,
 Shift As Integer,
 X As Single,
 Y As Single)
Command1_MouseUp(Button As Integer,
 Shift As Integer,
 X As Single,
 Y As Single)
```

де:

Button – містить номер клавіші миші:
1 – ліва кнопка миші;
4 – середня кнопка миші;
2 – права кнопка миші;

Shift – містить інформацію про стан клавіш **ALT**, **CTRL** і **SHIFT**:
4 – натиснуто кнопку **Alt**;
2 – натиснуто кнопку **Ctrl**;
1 – натиснуто кнопку **Shift**;

X,Y – координати покажчика миші, де сталася подія.

4.4.6. Елемент управління CheckBox

CheckBox – це прапорець на формі, який можна відзначати (ставити або знімати позначку). Використовується для вибору варіантів, які не є взаємовиключними. Може мати різні стани – по-

значено, не позначено і недоступно. У випадку недоступного стану елемент управління відображається як позначений, але є неактивним.

Основні властивості елементу управління **CheckBox** наведено у табл. 4.10.

Таблиця 4.10

Основні властивості елементу управління CheckBox

Властивість	Тип	Призначення
Caption	string	Текст, що відображатиметься біля позначки праворуч.
DisabledPicture	variant	Зображення, яке буде на позначці, поки вона не є доступною. Для розташування зображення необхідно встановити властивість **Style =** Graphical.
DownPicture	variant	Зображення, яке буде на позначці, поки вона натиснута. Для розташування зображення необхідно встановити властивість **Style =** Graphical.
ForeColor	long	Колір тексту надпису.
Picture	variant	Зображення, яке буде на позначці у графічному стані. Для розташування зображення необхідно встановити властивість **Style =** Graphical.
Style	integer	Стиль відображення познаки: 0 – Standart – звичайний; 1 – Graphical – графічний.
Value	integer	Повертає стан вибору: 0 – не позначено; 1 – позначено; 2 – недоступно.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

Основні події елементу **CheckBox:**

Click – основна подія позначки, яка відбувається при натисненні на неї.

Change – відбувається при зміні стану позначки.

4.4.7. Елемент управління OptionButton

OptionButton – кнопка-перемикач, яка дозволяє обрати один з декількох взаємовиключних варіантів. Використовується групами, коли потрібно обрати одночасно тільки одну кнопку з групи. Кнопки-перемикачі об'єднуються разом у групу за допомогою рамки **Frame.**

Основні властивості елементу управління **OptionButton** наведено у табл. 4.11.

Таблиця 4.11

Основні властивості елементу управління OptionButton

Властивість	Тип	Призначення
Caption	string	Текст, що відображатиметься біля позначки праворуч.
Enabled	boolean	True – елемент активовано; False – елемент деактивовано.
ForeColor	long	Колір тексту надпису.
Value	boolean	True – позначку встановлено; False – позначку знято.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

Основні події елементу **OptionButton:**
Click – основна подія кнопки, яка відбувається при натисненні на неї;
Change – відбувається при зміні стану кнопки.

4.4.8. Елемент управління ToggleButton

ToggleButton – вимикач, який виглядає як кнопка, що при першому натисненні стає натиснутою, а при повторному натисненні відключається. Використовується для вибору варіантів, які не є взаємовиключними. Може мати різні стани – натиснуте, відключене і недоступне. У випадку недоступного стану елемент управління відображається як позначений, але недоступний.

Основні властивості елементу управління **ToggleButton** наведено у табл. 4.12.

Таблиця 4.12

Основні властивості елементу управління `ToggleButton`

Властивість	Тип	Призначення
`Caption`	`string`	Текст, що відображатиметься біля відмітки з права.
`Enabled`	`boolean`	`True` – елемент активовано; `False` – елемент деактивовано.
`ForeColor`	`long`	Колір тексту надпису.
`Value`	`integer`	Повертає стан вибору: 0 – не позначено; 1 – позначено; 2 – недоступно.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.

Основні події елементу **`ToggleButton`:**

`Click` – основна подія кнопки, яка відбувається при натисненні на неї;

`Change` – відбувається при зміні стану кнопки.

Приклад

```
Private Sub ToggleButton1_Change()
                              'оголошення обробника події
  If ToggleButton1.Value = True Then TextBox1.Text =
"Увімкнено" Else TextBox1.Text = "Вимкнено"
                              'якщо кнопку натиснуто, то у
                              'полі для введення виводиться
                              'надпис "Увімкнено", інакше у
                              'полі для введення виводиться
                              'надпис "Вимкнено"
End Sub                       'завершення обробника події
```

4.4.9. Елемент управління `SpinButton`

`SpinButton` – лічильник, що дозволяє користувачеві збільшувати і зменшувати заданий параметр до набуття необхідного значення. Має вигляд елементу управління **`ScrollBar`** з двома кнопками, але без смуги прокрутки. Зазвичай використовується при роботі з невеликими значеннями.

Основні властивості елементу управління **`SpinButton`** наведено у табл. 4.13.

Таблиця 4.13

Основні властивості елементу управління `SpinButton`

Властивість	Тип	Призначення
`Enabled`	`boolean`	`True` – елемент активовано; `False` – елемент деактивовано.
`Max` **`Min`**	`integer`	Максимальне і мінімальне значення лічильника. Значення є цілими числами в діапазоні від –32767 до +32767. Задавати числові значення можна як у прямому, так і у зворотному порядку, наприклад, **`Min =`** `100`, а **`Max =`** `1`.
`Orientation`	`integer`	Задає орієнтацію: 1 – за замовчуванням, визначає орієнтацію згідно з параметрами форми користувача; –1 – горизонтальна орієнтація; 0 – вертикальна орієнтація.
`SmallChange`	`integer`	Визначає крок зміни значення. За замовчуванням дорівнює 1. Значення кроку є цілим числом з діапазону від –32767 до +32767.
`Value`	`integer`	Зберігає обране у поточний момент значення з діапазону, заданого властивостями **`Min`** і **`Max`**.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.

Елемент управління має додаткову подію:
`Change` – відбувається при натисненні на кнопку.

Приклад

```
Private Sub SpinButton1_Change()
                              'оголошення обробника події
TextBox1.Text = SpinButton.Value
                             'відображення поточного
                             'значення у полі для введення
End Sub                      'завершення обробника події
```

4.4.10. Елемент управління `Frame`

`Frame` – рамка із заголовком на формі. Призначена для візуального і логічного поєднання елементів управління. Здебільшого цей елемент управління використовується пасивно для

групування інших елементи управління, тому необхідності обробляти його події зазвичай не виникає.

Основні властивості елементу управління **Frame** наведено у табл. 4.14.

Таблиця 4.14

Основні властивості елементу управління Frame

Властивість	Тип	Призначення
BackColor	long	Визначає колір фону введення.
BorderColor	long	Визначає колір рамки поля введення.
BorderStyle	integer	Визначає тип рамки поля: 0 – без лінії обрамлення; 1 – рамка з одинарною лінією.
Caption	string	Текст, що відображається в заголовку довжиною максимум 1024 байт.
ForeColor	long	Колір тексту надпису.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

4.4.11. Елемент управління TabStrip

TabStrip – набір вкладок на формі. Складається із зони, в якій розміщуються інші елементи управління. Використовується для створення діалогових вкладок, що відображають одні й ті ж дані для різних категорій, тобто усі сторінки містять однаковий набір елементів управління.

Основні властивості елементу управління **TabStrip** наведено у табл. 4.15.

Для управління вкладками при їх створенні потрібно запустити контекстне меню правою кнопкою миші. Контекстне меню містить наступні пункти:

NewPage – додати нову вкладку до наявних;
DeletePage – видалити обрану вкладку;
Rename – перейменувати вкладку;
Move – перемістити обрану вкладку у заданому напрямі.

Елемент управління має додаткову подію:
Change – відбувається при переході між вкладками.

Таблиця 4.15

Основні властивості елементу управління `TabStrip`

Властивість	Тип	Призначення
`BackColor`	`long`	Визначає колір фону введення
`ForeColor`	`long`	Колір тексту надпису.
`MultiRow`	`boolean`	`True` – якщо вкладки не вміщатимуться в один рядок на поверхні форми, то вони розташовуватимуться у декілька рядків; `False` – якщо вкладки не вміщатимуться в один рядок на поверхні форми, то вони розташовуватимуться на одній прямій з додаванням спеціальної прокрутки для переходу до вкладок, що не доступні.
`SelectedItem`	`integer`	Індекс обраної вкладки
`Style`	`integer`	Стиль відображення вкладок: 0 – закладки; 1 – кнопки; 2 – без відображення вкладок.
`TabOrientation`	`integer`	Визначає розташування вкладок: 0 – зверху; 1 – знизу; 2 – ліворуч; 3 – праворуч.
`Value`	`integer`	Містить номер обраної вкладки, причому нумерація починається з нуля.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.

4.4.12. Елемент управління `Page`

`Page` – набір сторінок, кожна з яких містить власний заголовок і власний набір елементів управління. Використовується у тому випадку, якщо елементів управління дуже багато і всі вони не уміщаються на формі. На відміну від елементу управління **`TabStrip`**, у елементі **`Page`** усі сторінки містять різний набір елементів управління.

Основні властивості елементу управління **`Page`** наведено у табл. 4.16.

Таблиця 4.16

Основні властивості елементу управління `Page`

Властивість	Тип	Призначення
`BackColor`	`long`	Визначає колір фону введення.
`ForeColor`	`long`	Колір тексту надпису.
`MultiRow`	`boolean`	`True` – якщо вкладки не вміщатимуться в один рядок на поверхні форми, то вони розташовуватимуться у декілька рядків; `False` – якщо вкладки не вміщатимуться в один рядок на поверхні форми, то вони розташовуватимуться на одній прямій з додаванням спеціальної прокрутки для переходу до недоступних вкладок.
`SelectedItem`	`integer`	Містить інформацію про обрану вкладку.
`Style`	`integer`	Стиль відображення вкладок: 0 – закладки; 1 – кнопки; 2 – без відображення вкладок.
`TabOrientation`	`integer`	Визначає розташування вкладок: 0 – зверху; 1 – знизу; 2 – ліворуч; 3 – праворуч.
`Value`	`integer`	Містить номер обраної вкладки, причому нумерація починається з нуля.
`Visible`	`boolean`	`True` – елемент відображається під час виконання програми; `False` – елемент сховано під час виконання програми.

Для управління вкладками при їх створенні потрібно запустити контекстне меню правою кнопкою миші. Контекстне меню містить наступні пункти:

`NewPage` – додати нову вкладку до наявних;
`DeletePage` – видалити обрану вкладку;
`Rename` – перейменувати вкладку;
`Move` – перемістити обрану вкладку у заданому напрямі.

Елемент управління має додаткову подію:

`Change` – відбувається при переході між вкладками.

4.4.13. Елемент управління ScrollBar

ScrollBar – смуга прокрутки. Смуги прокрутки забезпечують простоту переміщення довгим списком або великим масивом інформації, горизонтально або вертикально переміщуючи огляд в межах застосування або елементу управління. Використовується в елементах управління, коли вміст не уміщається у цьому елементі.

Основні властивості елементу управління **ScrollBar** наведено у табл. 4.17.

Таблиця 4.17

Основні властивості елементу управління ScrollBar

Властивість	Тип	Призначення
Enabled	boolean	True – елемент активовано; False – елемент деактивовано.
LargeChange	integer	Визначає крок зміни повзунка при натисненні на полосі прокрутки. За замовчуванням дорівнює 1, значення кроку є цілим числом з діапазону від –32767 до +32767.
Max **Min**	integer	Максимальне і мінімальне значення лічильника. Значення є цілими числами в діапазоні від –32767 до +32767. Задавати числові значення можна як у прямому, так і у зворотному порядку, наприклад, **Min =** 100, а **Max =** 1.
Orientation	integer	Задає орієнтацію: 1 – за замовчуванням, визначає орієнтацію згідно з параметрами форми користувача; –1 – горизонтальна орієнтація; 0 – вертикальна орієнтація.
SmallChange	integer	Визначає крок зміни повзунка при натисненні на кнопки прокрутки. За замовчуванням дорівнює 1, значення кроку є цілим числом з діапазону від –32767 до +32767.
Value	integer	Зберігає поточне положення повзунка на полосі прокрутки з діапазону, заданого властивостями **Min** і **Max**.
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.

Елемент управління має додаткові події:
Change – відбувається після переміщення повзунка після того, як повзунок відпущено, або коли натиснуто кнопки прокручування або полоса прокручування.
Scroll – відбувається під час переміщення повзунка.

4.4.14. Елемент управління Image

Image – розміщує на формі графічне зображення у форматах: *.bmp, *.cur, *.gif, *.ico, *.jpg, *.wmf. Може використовуватися як замінник кнопок або для створення простих анімацій.

Основні властивості елементу управління **Image** наведено у табл. 4.18.

Таблиця 4.18

Основні властивості елементу управління Image

Властивість	Тип	Призначення
Visible	boolean	True – елемент відображається під час виконання програми; False – елемент сховано під час виконання програми.
AutoSize	boolean	True – встановлюється режим автоматичної зміни розміру рисунку так, щоб усе зображення поміщалося у полі; False – встановлюється фіксований розмір поля.
Picture = LoadPicture(Ім'я Файла**)**	string	Завантаження графічного файлу за допомогою методу **LoadPicture**. Ім'я_Файла – повне ім'я графічного файлу
PictureSizeMode	integer	Режим масштабування рисунка: 0 – частини рисунку, що не вміщуються у полі, обрізаються; 1 – рисунок масштабується так, щоб він займав усю площу поля; 3 – рисунок масштабується зі збереженням відносних розмірів так, щоб він повністю вміщувався у полі.
PictureAlignment	integer	Положення рисунку у середині поля: 0 – у верхньому лівому куті; 1 – у верхньому правому куті; 2 – по центру; 3 – у нижньому лівому куті; 4 – у нижньому правому куті.
PictureTiling	boolean	True – поле покривається мозаїкою із зображень; False – на полі відображається один рисунок

Елемент управління має додаткову подію:
Click – подія, яка відбувається при натисненні на рисунок.

Приклади

```
Image1.Picture = LoadPiture("c:\Windows\
малюнок.bmp")              'завантаження файлу зображення
                           'з диску
Image1.Picture = LoadPicture
                           'очищення елементу управління
                           'Image від графіки
```

4.5. Розроблення інтерфейсу програми

Загальна схема макросу, що використовує форму користувача наведено на рис. 4.10.

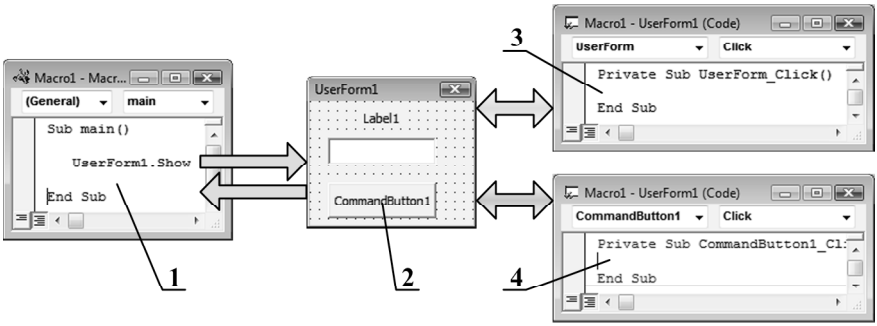


Рис. 4.10. Схема макросу, що використовує форму користувача: *1* – головна програма; *2* – форма користувача з елементами управління; *3* – обробники подій форми користувача; *4* – обробники подій елементів управління

Для запуску форми користувача з головної програми застосовується метод **Show**.

Для створення програм з гарним інтерфейсом і дизайном необхідно дотримуватися основ композиції, використання кольорів та простоти інтерфейсу.

Композиція є важливою складовою інтерфейсу і визначає, наскільки легко буде працювати з програмою. Композиція у програмуванні включає розташування елементів управління,

ефективне використання вільного простору і візуальний зв'язок між компонентами.

Розташування елементів управління

У більшості програм елементи управляння мають різне функціональне значення. Їх необхідно розташувати так, щоб важливіші елементи були помітні користувачеві у першу чергу, а менш важливі – потім.

У більшості мов текст розташовується зліва направо , зверху вниз. Принцип роботи з інформацією таким чином використовується і при роботі з комп'ютером: погляд користувача спершу упирається у верхній лівий кут монітора, тому важливіші елементи повинні знаходиться там. Навпаки, кнопки **OK** і **Cancel** повинні розташовуватися у нижній частині екрана: користувач не натискає їх до тих пір, поки не закінчить працювати з іншими даними у вікні.

Візуальний зв'язок між компонентами

Створюючи інтерфейс, розробник повинен прагнути до того, щоб користувач побачив візуальний зв'язок між окремими компонентами там, де це потрібно. Наприклад, тривимірні ефекти на кнопках підкреслюють їх призначення. Але якщо зробити їх пласкими, то користувач може не здогадатися, що це кнопки. Якщо прибрати елемент об'ємності з поля для введення тексту, то його легко можна прийняти за нередагований елемент.

Використання вільного простору

За допомогою вільного простору можна відокремлювати один елемент від іншого у вікні й надавати особливого акценту деяким з них. Не варто розташовувати дуже багато елементів в одному вікні – це призводить до відчуття хаотичності інтерфейсу і створює враження недопрацьованої, дешевої програми.

Простота інтерфейсу

Одним з найважливіших принципів створення інтерфейсу є прагнення до простоти. Якщо інтерфейс виглядає складним, то програма сприйматиметься користувачем важко. Крім того, з естетичної точки зору простий, ясний дизайн завжди виглядає краще.

Стандартна помилка при створенні інтерфейсу – змоделювати зовнішній вигляд програми до подібних реальних об'єктів.

4.6. Приклад розроблення інтерфейсу програми

Завдання: створити форму користувача та макрос, що її завантажує. На формі розмістити елементи управління (рис. 4.11) та організувати зв'язок між ними, а саме:
а) рамка **Frame**, що містить шість кнопок **CommandButton** для зміни кольору форми користувача на такі кольори: білий, чорний, бірюзовий, жовтий, зелений, синій.
б) поле для введення тексту **TextBox** та надпис **Label** пов'язані між собою таким чином: коли у полі для введення змінюють текст, напис дублює цей текст.
в) випадаючий список **ComboBox**, який містить декілька букв латинського алфавіту, та надпис **Label** пов'язані між собою так: коли користувач обирає пункт у випадаючому списку, надпис дублює обраний текст.
г) невеликий рисунок.
д) кнопку завершення роботи модуля з надписом «**Закрити**».

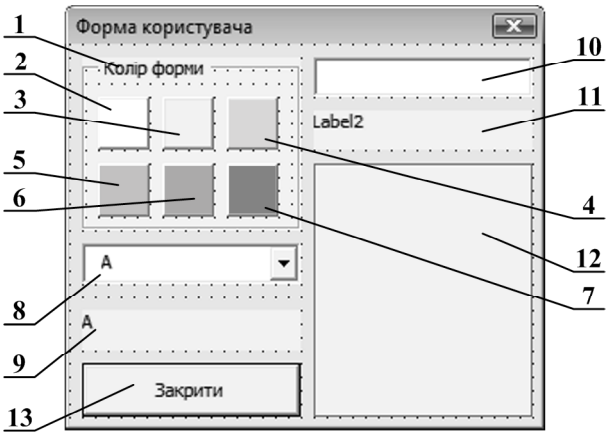


Рис. 4.11. Форма користувача до завдання: *1* – рамка **Frame1** ; *2*–*7* – кнопки зміни кольору форми **CommandButton1**–**CommandButton6**; *8* – випадаючий список **ComboBox1**; *9* – надпис **Label1**, пов'язаний з випадаючим списком **ComboBox1**; *10* – поле для введення **TextBox1**; *11* – надпис **Label2**, пов'язаний з полем для введення **TextBox1**; *12* – зображення **Image1**; *13* – кнопка **CommandButton7** з надписом «**Закрити**»

Послідовність дій при створенні форми користувача:
1. Запустити SolidWorks. Увімкнути панель інструментів **Макрос**.
2. Для створення нового макросу на цій панелі запустити інструмент **Создать макрос**. Задати ім'я нового макросу. Буде відкрито вікно вбудованого середовища програмування VBA.
3. У головній програмі макросу додати команду **UserForm1.Show** для запуску форми користувача:

```
Sub main()
  UserForm1.Show          'запускання форми користувача
End Sub
```

4. Створити форму користувача. Для цього у головному меню середовища VBA обрати пункт головного меню **Insert→UserForm**.
5. Розмістити рамку **Frame1** для кнопок зміни кольору. У менеджері властивостей цього елементу як значення властивості **Caption** записати "Колір форми".
6. Розмістити у рамці **Frame** шість кнопок **CommandButton**.
6.1. Змінити розмір кнопок та розмістити їх у рамці **Frame**, як показано на рис. 4.11. Видалити надпис на кнопці. Для цього у менеджері властивостей кнопок видалити надпис властивості **Caption**.
6.2. Встановити колір кнопки. Для цього у менеджері властивостей кнопки у значення властивості **BackColor** записати колір згідно із завданням на основі табл. 4.4.
6.3. Створити обробник події кнопки при натисканні. Для цього подвійним натисканням лівої кнопки миші на кнопці відкрити вікно коду обробника події. У обробнику події слід передати колір кнопки кольору форми користувача, скориставшись відповідними властивостями компонентів:

```
Private Sub CommandButton1_Click()
                        'оголошення обробника події
  UserForm1.BackColor = CommandButton1.BackColor
                        'колір форми такий самий, як
                        'колір кнопки
End Sub                 'завершення обробника події
```

6.4. У такий же спосіб налаштувати інші 5 кнопок для зміни кольору форми користувача, застосовуючи значення кольорів чорного, бірюзового, жовтого, зеленого і синього згідно з табл. 4.4.
7. Розмістити на формі компоненти **TextBox1** та **Label1**. Зв'язати їх між собою так, щоб коли у полі для введення змінюють текст, надпис **Label** дублював цей текст. Це робиться за допомогою ви-

користання обробника подій зміни вмісту надпису **Change**.

7.1. Створити обробник події зміни вмісту текстового поля **TextBox1**. Для цього подвійним натисканням лівої кнопки миші на полі відкрити вікно коду обробника події. У обробнику події слід передати введений користувачем текст у полі **TextBox1** надпису **Label1**, скориставшись відповідними властивостями компонентів:

```
Private Sub TextBox1_Change()
                          'оголошення обробника події
  Label1.Caption = TextBox1.Text
                          'введений текст у полі
                          'відображається у надпису
End Sub                   'завершення обробника події
```

8. Розмістити на формі компоненти **ComboBox1** та **Label2**. Зв'язати їх між собою у такий спосіб, щоб коли у списку змінюють обраний рядок, надпис дублював цей текст. Це робиться за допомогою використання обробника подій зміни вмісту компонента форми **Change**.

8.1. Завантажити у випадаючий список букви латинського алфавіту. Для того щоб букви у список завантажувались відразу після запуску форми, потрібно скористатися обробником події форми **Activate**, що запускається відразу після створення форми користувача.

Для створення відповідної процедури оброблення події потрібно у вікні коду з випадаючого списку (*2*, рис. 4.8) обрати форму, з списку подій форми (*3*, рис 4.8) обрати **Activate**. У обробнику події слід завантажити у випадаючий список букви латинського алфавіту:

```
Private Sub UserForm_Activate()
                          'оголошення обробника події
 ComboBox1.AddItem ("À")  'додавання до списку букви A
 ComboBox1.AddItem ("B")  'додавання до списку букви B
 ComboBox1.AddItem ("C")  'додавання до списку букви C
 ComboBox1.AddItem ("D")  'додавання до списку букви D
 ComboBox1.Text = "À"     'за замовчуванням обрано
                          'букву A
End Sub                   'завершення обробника події
```

8.2. Створити обробник події зміни вмісту компонента для випадаючого списку **ComboBox1**. Для цього подвійним натисканням лівої кнопки миші на полі відкрити вікно коду обробника події. У обробнику події слід передати обраний користувачем

текст у списку **ComboBox1** надпису **Label2**, скориставшись відповідними властивостями компонентів:

```
Private Sub ComboBox1_Change()
                          'оголошення обробника події
  Label2.Caption = ComboBox1.Text
                          'обраний текст у списку
                          'відображається у надпису
End Sub                   'завершення обробника події
```

9. Розмістити на формі компонент **Image1**. Завантажити у компонент рисунок для відображення. Для цього у менеджері властивостей компонента **Image1** обрати властивість **Picture** та завантажити файл рисунку у форматі *.**bmp** або *.**jpg**.

10. Розмістити на формі компонент **CommandButton7**.

10.1. Змінити надпис на кнопці. Для цього у менеджері властивостей кнопки як значення властивості **Caption** записати "Закрити".

10.2. Створити обробник події кнопки при натисканні. Для цього подвійним натисканням лівої кнопки миші на кнопці відкрити вікно коду обробника події. У обробнику події слід використати метод **Unload**, що закриває форму та повертає управління головній програмі:

```
Private Sub CommandButton7_Click()
                          'оголошення обробника події
  Unload UserForm1        'закриття форми користувача
End Sub                   'завершення обробника події
```

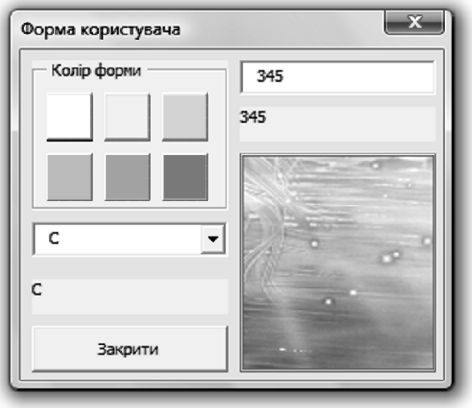


Рис. 4.12. Результат роботи форми користувача

Контрольні запитання і завдання

1. Дайте визначення візуального програмування.
2. Назвіть переваги візуального програмування.
3. Що називається діалоговим вікном Windows?
4. Охарактеризуйте стандартні діалогові вікна повідомлень Windows.
5. Охарактеризуйте стандартні діалогові вікна Windows для введення інформації.
6. Що називається інтерфейсом користувача?
7. Дайте визначення форми користувача.
8. Які характерні елементи має форма?
9. Яким чином у макрос додається форма користувача?
10. Дайте визначення властивості форми.
11. Дайте визначення методів форми.
12. Дайте визначення подій форми.
13. Назвіть декілька основних властивостей форми.
14. Назвіть декілька основних методів форми.
15. Назвіть декілька основних подій форми.
16. Що таке елемент управління?
17. Назвіть елементи управління для виведення інформації на форму.
18. Назвіть елементи управління введення інформації користувачем.
19. Назвіть основні вимоги до розроблення інтерфейсу користувача.
20. Створіть спрощений калькулятор на основі форми користувача та елементів управління.

5. Приклади програм

5.1. Робота з масивами

Робота з масивами є одним з найпоширеніших завдань програмування. Умовно можна окреслити такі основні завдання, що вирішує робота з масивами:

– пошук елементів масиву за визначеними умовами;
– різноманітні математичні операції з елементами масиву, наприклад, визначення середньоарифметичних та середньогеометричних значень елементів масиву;
– визначення мінімального або максимального елементу масиву;
– сортування елементів масиву за зростанням або спаданням.

Для розв'язання цих завдань потрібно організувати перебирання елементів масиву. Це робиться за допомогою циклів, а оскільки число елементів масиву заздалегідь відомо, то при роботі з масивами для перебору елементів часто використовується цикл **For...Next**.

Завдання 1. Нехай задано масив M(5, 5) типу `Double`. Розробити функцію для визначення кількості елементів масиву, які менші за число **K**.

```
Function M_K (M As Variant, K As Double) As Integer
                          'аргументи функції: заданий
                          'масив, число K. Функція
                          'повертає кількість елементів
                          'масиву, які менші за число K
Dim N As Integer, A As Integer, B As Integer
                          'оголошення змінних
N = 0                     'присвоєння первинного
                          'значення змінній-лічильнику
For A = 0 To 5            'організація циклу
                          'перебирання по стовпцях
  For B = 0 To 5          'організація циклу
                          'перебирання по рядках
    If M(A, B)< K Then N = N + 1
                          'пошук елементів масиву, які
                          'менші за число K. Якщо такий
                          'елемент знайдено, то змінна-
                          'лічильник збільшується на
                          'одиницю
```

```
  Next B                    'завершення циклу
Next A                      'завершення циклу
M_K = N                     'повернення функції результату
End Function                'завершення функції
```

Завдання 2. Нехай задано масив V(10) типу `Double`. Розробити функцію для визначення середнього арифметичного значення усіх його елементів.

```
Function S_A (V As Variant) As Double
                            'аргумент функції: заданий
                            'масив. Функція повертає
                            'середнє арифметичне значення
                            'усіх елементів масиву
Dim Sp As Double            'змінна для середнього
                            'арифметичного
Sp = 0                      'присвоєння первинного значення
For a = 0 To 10             'організація циклу для
                            'визначення середнього
                            'арифметичного значення
   Sp = Sp+V(a)             'визначення суми усіх елементів
                            'масиву
Next a                      'завершення циклу
Sp = Sp / 11                'визначення середнього
                            'арифметичного
S_A = Sp                    'повернення функції результату
End Function                'завершення функції
```

Завдання 3. Нехай задано масив V(10) типу `Double`. Розробити функцію для визначення значення максимального елементів масиву.

```
Function Max_E (V As Variant) As Double
                            'аргумент функції: заданий
                            'масив. Функція повертає
                            'значення максимального елементу
                            'масиву
Dim Max As Double           'змінна для максимального числа
Max = V(0)                  'присвоєння первинного значення
For a = 0 To 10             'організація циклу для перевірки
                            'усіх елементів масиву
  If Max < V(a) Then Max = V(a)
                            'визначення максимального
                            'значення
Next a                      'завершення циклу
Max_E = Max                 'повернення функції результату
End Function                'завершення функції
```

Завдання 4. Нехай задано одновимірний масив V(10) типу `Double`. Розробити функцію для сортування елементів масиву за зростанням їх значень.

```
Function Sort_Max (V As Variant) As Variant
                          'аргумент функції: заданий
                          'масив. Функція повертає
                          'розсортований за зростанням
                          'масив
Dim N As Double           'допоміжна змінна
For a = 0 To 10           'оголошення циклу
  For b = 0 To 9          'оголошення циклу
    If V(b) > V(b+1) then
                          'якщо поточний елемент масиву
                          'більший за наступний, то
                          'потрібно змінити місцями ці
                          'елементи. Для цього...
    N = V(b+1)            'змінній N присвоїти значення
                          'наступного елементу масиву
    V(b+1) = V(b)         'наступному елементу масиву
                          'присвоїти значення поточного
                          'елементу масиву
    V(b)= N               'поточному елементу масиву
                          'присвоїти значення змінної N
    End If                'завершення умови
  Next b                  'завершення циклу
Next a                    'завершення циклу
Sort_Max = V              'повернення функції результату
End Function              'завершення функції
```

5.2. Округлення чисел

Одна з найчастіших операцій, що трапляється при проектуванні виробів, – це округлення. ***Округлення*** – математична операція, що дозволяє зменшити кількість знаків у числі через заміну числа його наближеним значенням з певною точністю.

Розглянемо декілька випадків округлення чисел.

Округлення до цілого. Для округлення числа до цілого використовуються функції **`Round`**, **`Fix`** та **`Int`.**

Приклад роботи цих функцій:

```
Round(2.3,0) → 2     Int(2.3) → 2     Fix(2.3) → 2
Round(2.5,0) → 2     Int(2.5) → 2     Fix(2.5) → 2
Round(2.6,0) → 3     Int(2.6) → 2     Fix(2.6) → 2
```

Округлення до 5. Для округлення чисел до 5 необхідно розробити функцію, яка працює таким чином:

```
Round_5(2.1) → 2.0
Round_5(2.4) → 2.5
Round_5(2.7) → 3.0
```

Приклад такої функції округлення чисел до 5 наведено нижче.

```
Function Round_5(N As Double) As Integer
                          'оголошення функції. Аргумент
                          'N – число для округлення.
                          'Функція повертає округлене до
                          '5 число.
Dim s As String, ld As String,d As Integer
                          'оголошення допоміжних змінних
N = Int(N)                'округлення числа до цілого
ld = Mid(Str(N), Len(Str(N)), 1)
                          'перетворення числа для
                          'округлення у рядок і отримання
                          'останньої цифри числа
Select Case ld            'оголошення блоку прийняття
                          'рішень
    Case "0", "1", "2", "3"
                          'якщо остання цифра числа є
                          '0, 1, 2 або 3, то...
       d = 0              'останньою цифрою після
                          'округлення буде 0
    Case "4", "5", "6"    'якщо остання цифра числа є 4,5
                          'або 6, то...
       d = 5              'останньою цифрою після
                          'округлення буде 5
    Case "7", "8", "9"    'якщо остання цифра числа є 7,8
                          'або 9, то...
       d = 10             'останньою цифрою після
                          'округлення буде 10
End Select                'завершення блоку прийняття
                          'рішення
N = (N-Val(ld)) + d       'визначення округленого числа
Round_5 = N               'повернення значення функції
End Function              'завершення функції
```

Функція працює так. Спочатку число для округлення округлюється до цілого та виділяється його остання цифра. Залежно від значення цієї цифри визначається додаткове число **d**. Наприкінці від числа для округлення віднімається його остання цифра і додається додаткове число **d**.

Округлення до 10. Для округлення чисел до 10 необхідно розробити функцію, яка працює так:

```
Round_10(12.1) →10
Round_10(28.4) →30
Round_10(25.0) →20
```

Приклад такої функції округлення чисел до 10 наведено нижче.

```
Function Round_10(N As Double) As Integer
                          'оголошення функції. Аргумент
                          'N – число для округлення.
                          'Функція повертає округлене до
                          '10 число
  N = Round(N / 10) * 10
                          'визначення округленого числа
  Round_10 = N            'повернення значення функції
End Function              'завершення функції
```

Функція працює так. Спочатку число ділиться на 10, потім округлюється до цілого, після цього множиться на 10.

Приведення розмірів до рядів переважних чисел. На основі округлення розроблено ряди переважних чисел, що широко застосовуються у техніці.

Ряди переважних чисел – це упорядкована послідовність чисел, призначена для уніфікації значень технічних параметрів.

Застосування цих рядів дозволяє:

– уніфікувати посадочні розміри деталей (як наслідок, наприклад, у серійному виробництві скорочується кількість типорозмірів деталей, необхідних для комплектації різних виробів);
– використовувати типовий сортамент і заготовки (прутки, труби, кола, дріт та ін.);
– використовувати типовий інструмент (свердла, фрези тощо).

Ряди переважних чисел створюються на основі числових послідовностей. Найбільш поширені геометричні прогресії зі знаменником $q = \sqrt[n]{10}$, де n = 5, 10, 20, 40 – ступінь кореня. Ці стандартні ряди переважних чисел за ГОСТ 8032–84 позначаються R5, R10, R20, R40. Кожен ряд містить у кожному десятковому інтервалі відповідно 5, 10, 20 і 40 різних чисел (див. табл. 5.1). Ряд з меншою кількістю чисел є кращим по відношенню до ряду з більшою кількістю чисел.

Таблиця 5.1

Ряд переважних чисел від 1 до 10

Основні ряди			
R5	R10	R20	R40
1,00	1,00	1,00	1,00
			1,06
		1,12	1,12
			1,18
	1,25	1,25	1,25
			1,32
		1,40	1,40
			1,50
1,60	1,60	1,60	1,60
			1,70
		1,80	1,80
			1,90
	2,00	2,00	2,00
			2,12
		2,24	2,24
			2,36
2,50	2,50	2,50	2,50
			2,65
		2,80	2,80
			3,00
	3,15	3,15	3,15

Основні ряди			
R5	R10	R20	R40
			3,35
		3,55	3,55
			3,75
4,00		4,00	4,00
			4,25
		4,50	4,50
			4,75
	5,00	5,00	5,00
			5,30
		5,60	5,60
			6,00
6,30	6,30	6,30	6,30
			6,70
		7,10	7,10
			7,50
	8,00	8,00	8,00
			8,50
		9,00	9,00
			9,50
10,00	10,00	10,00	10,00

Базовим є ряд чисел від 1 до 10. Переважні числа інших десятинних порядків отримують множенням або діленням на 10, 100 та ін.

Функція приведення введених значень до ряду переважних чисел працює так. Спочатку введене значення для округлення приводиться до проміжку від 1 до 10 за допомогою множення або ділення на 10. Якщо значення для округлення більше за 10,

то воно ділиться на 10, а якщо менше за 1, то воно множиться на 10. Кількість множень та ділень фіксується для того, щоб наприкінці значення привести до свого первинного числового проміжку.

Наступним кроком є визначення різниці між цим значенням та числами з обраного користувачем переважного ряду. Остаточно округлююче число обирається за мінімальною різницею.

Наприкінці обране округлююче число приводиться до первинного числового проміжку.

```
Function Normalize(N as Double, R As Integer) as Double
                          'оголошення функції. Аргумент
                          'N – число для приведення до
                          'нормального ряду; R – номер
                          'ряду, до якого приводиться
                          'число 5, 10, 20 або 40.
                          'Функція повертає приведене до
                          'заданого нормального ряду
                          'число
Dim N_R(40) As Single     'оголошення масиву для таблиці
Dim Np As Double, Min As Double, Res As Double
Dim A As Integer, P As Integer
                          'оголошення допоміжних змінних
   '--Завантаження таблиці переважних чисел у масив--
N_R(0)= 1.00:N_R(1)= 1.06:N_R(2)= 1.12:N_R(3)= 1.18
N_R(4)= 1.25:N_R(5)= 1.32:N_R(6)= 1.40:N_R(7)= 1.50
N_R(8)= 1.60:N_R(9)= 1.70:N_R(10)= 1.80:N_R(11)= 1.90
N_R(12)= 2.00:N_R(13)= 2.12:N_R(14)= 2.24:N_R(15)= 2.36
N_R(16)= 2.50:N_R(17)= 2.65:N_R(18)= 2.80:N_R(19)= 3.00
N_R(20)= 3.15:N_R(21)= 3.35:N_R(22)= 3.55:N_R(23)= 3.75
N_R(24)= 4.00:N_R(25)= 4.25:N_R(26)= 4.50:N_R(27)= 4.75
N_R(28)= 5.00:N_R(29)= 5.30:N_R(30)= 5.60:N_R(31)= 6.00
N_R(32)= 6.30:N_R(33)= 6.70:N_R(34)= 7.10:N_R(35)= 7.50
N_R(36)= 8.00:N_R(37)= 8.50:N_R(38)= 9.00:N_R(39)= 9.50
N_R(40)=10.00
    '--Приведення числа до інтервалу від 1 до 10--
Np = N                    'збереження введеного числа
P = 0                     'початкове значення змінної
                          'показника ступеня
If N > 10 Then            'якщо число для округлення
                          'більше за 10, то...
  While Np > 10           'поки число для округлення
                          'більше за 10, виконувати...
    P = P + 1             'збільшення значення показника
                          'ступеня
    Np = Np / (10)        'зменшення числа у 10 разів
```

```
  Wend                    'завершення циклу
  ElseIf N< 1 Then      'інакше, якщо число для
                          'округлення менше за 10, то...
  While Np < 1            'поки число для округлення
                          'менше за 10, виконувати...
    P = P- 1              'зменшення значення показника
                          'ступеня
    Np = Np * (10)        'збільшення числа у 10 разів
  Wend                    'завершення циклу
End If                    'завершення умови
   '-- Приведення числа до ряду переважних чисел --
Res = 0                   'початкове значення змінної
                          'результату
Min = Np                  'початкове значення змінної
                          'мінімальної різниці між
                          'числом для округлення і
                          'числом з переважного ряду
For A = 0 To R            'організація циклу перебирання
                          'елементів таблиці переважного
                          'ряду. Цикл виконується стільки
                          'разів, скільки елементів у
                          'ряду переважних чисел R
  If Abs(Np - N_R(A * 40/R))<= Min Then
                          'якщо різниця між числом для
                          'округлення і числом з
                          'переважного ряду менше або
                          'дорівнює за значення змінної
                          'мінімальної різниці, то...
  Min = Abs(Np - N_R(A * 40/R))
                          'змінній мінімальної різниці
                          'присвоюється значення різниці
  Res = N_R(A * 40/R)     'визначення результуючого
                          'значення
  End If                  'завершення умови
Next A                    'завершення циклу
Normalize = Round(Res * 10 ^ P,2)
                          'повернення значення функції
End Function              'завершення функції
```

5.3. Робота з файлами

При розробленні макросів часто виникає потреба завантаження і оброблення таблиць даних. Як файли даних, що містять таблиці значень, зручно використовувати файли формату CSV.

Формат CSV (від англ. Comma-Separated Values – значення, розділені комами) – текстовий формат, призначений для пред-

ставлення табличних даних. Кожен рядок файлу – це один рядок таблиці. Значення окремих колонок розділяються символом (delimiter) – крапка з комою (;). Рядки розділяються парою символів `CR LF` (у DOS і Windows ця пара генерується натисненням клавіші **Enter**). Файли такого формату відкриті і можуть створюватися й редагуватися у звичайних текстових редакторах. Причому цей формат підтримують багато програм: Microsoft Excel, MathCAD, SolidWorks Simulation, SolidWorks FlowSimulation та ін.

5.3.1. Функція визначення кількості рядків у текстовому файлі

При визначенні кількості рядків у текстовому файлі потрібно організувати перебирання цих рядків, як для масиву. Це робиться за допомогою циклів, однак оскільки число рядків заздалегідь не відомо, то при роботі з файлами для перебору рядків використовуються цикли **`While...Wend`** або **`Do...Loop`**.

```
Function FileLen(FName As String) As Integer
                          'оголошення функції. Аргумент
                          'FName – ім'я файлу. Функція
                          'повертає кількість рядків.
Dim N As Integer, FNum As Integer, S As String
                          'оголошення змінних: N –
                          'лічильник рядків, S – змінна
                          'для читання рядків з файлу,
                          'FNum – номер файлу
  N = 0                   'початкове значення лічильника
                          'рядків
FNum = FreeFile           'на дання номеру файла
Open FName For Input As FNum
                          'відкриття файлу для читання
While Not EOF(FNum)       'організація циклу читання
                          'файлу. Цикл працює, доки не
                          'досягнуто кінець файлу, тобто
                          '(EOF(FNum) = True)
Line Input #FNum, S       'читання рядка з файлу
    N = N + 1             'збільшення лічильника на 1
Wend                      'кінець циклу
Close FNum                'закриття файлу
FileLen = N               'повернення функції значення
                          'кількості рядків у файлі
End Function              'кінець функції
```

5.3.2. Функція читання визначеного рядка з текстового файлу

Функція читання визначеного користувачем рядка з текстового файлу працює так. Спочатку відкривається вказаний текстовий файл для читання. Далі за допомогою циклу з нього зчитуються рядки до вказаного рядка. Потім зчитується вказаний рядок і його значення повертається функцією.

```
Function ReadFile(FName As String, N As Integer) As String
                        'оголошення функції. Аргумент
                        'FName – ім'я файлу, N – номер
                        'рядка, який потрібно прочитати
                        'з файлу. Функція повертає
                        'прочитаний рядок.
  Dim FNum As Integer, S As String
                        'оголошення змінних: S – змінна
                        'для читання рядків з файлу,
                        'FNum – номер файлу
  FNum = FreeFile       'надання номеру файлу
  Open FName For Input As FNum
                        'відкриття файлу для читання
For A = 1 To N-1        'організація читання рядків з
                        'файлу до потрібного рядка
  Line Input #FNum, S   'читання рядка з файлу
Next A                  'завершення циклу
Line Input #FNum, S     'читання заданого рядка з файлу
Close FNum              'закриття файлу
ReadFile = S            'повернення функції значення
                        'прочитаного рядка
End Function            'кінець функції
```

5.3.3. Функція визначення кількості стовпців таблиці у CSV– файлі

Кількість стовпців визначається з першого рядка файлу за кількістю символів **";"**, що є роздільниками стовпців. Для роботи потрібно, щоб десятинним розділювачем була крапка.

```
Function CSV_X(FName As String) As Integer
                        'оголошення функції. Аргумент
                        'FName – ім'я файлу. Функція
                        'повертає кількість стовпців
                        'таблиці у CSV-файлі.
```

```
Dim N As Integer, FNum As Integer, S As String,
A As Integer              'оголошення змінних:N –
                          'лічильник рядків, S – змінна
                          'для читання рядків з
                          'файлу, FNum – номер файлу, A –
                          'лічильник циклу
N = 0                     'початкове значення лічильника
                          'стовпців
FNum = FreeFile           'надання номера файлу
Open FName For Input As FNum
                          'відкриття файлу
Line Input #FNum, S       'читання рядка з файлу
For A = 1 To Len(S)       'організація циклу перебирання
                          'символів у рядку. Цикл
                          'виконується стільки разів,
                          'скільки символів у рядку.
                          'Довжину рядка визначено
                          'функцією Len(S)
  If Mid(S, A, 1) = ";" Then N = N + 1
                          'виділення з рядка S одного
                          'символу, причому при роботі
                          'циклу послідовно виділяються
                          'усі символи рядка. Якщо
                          'виділений символ = ";",
                          'лічильник стовпців збільшується
                          'на 1
Next A                    'кінець циклу
Close FNuml               'закриття файлу
CSV_X = N + 1             'повернення функції значення
                          'кількості стовпців у файлі
End Function              'кінець функції
```

5.3.4. Процедура завантаження таблиці з CSV–файлу в динамічний масив

Процедура спочатку визначає кількість рядків і стовпців у CSV-файлі за допомогою функцій, розглянутих вище. Потім вона послідовно зчитує рядки з файлу і поділяє їх на стовпці, використовуючі символ роздільника стовпців.

```
Public DB() As Double     'оголошення глобального
                          '(доступного усім процедурам і
                          'функціям) динамічного масиву.
                          'Оголошення записується у
                          'заголовку макросу (перед
                          'усіма функціями і
                          'процедурами)
```

```
Sub CSVLoad (FName As String)
                        'оголошення процедури. Аргумент
                        'FName – ім'я CSV-файлу.
Dim X As Integer, Y as Integer,S As String,
S1 As String,A As Integer, FNum As Integer,
N As Integer            'оголошення змінних
Y = FileLen (FName)     'визначення кількості рядків у
                        'файлі. Функцію розглянуто
                        'вище
X = CSV_X (FName)       'визначення кількості стовпців
                        'таблиці у CSV-файлі. Функцію
                        'розглянуто вище
ReDim DB(X-1,Y-1)       'визначення розмірності масиву,
                        'яка відповідає розмірності
                        'таблиці
FNum = FreeFile         'надання номера файлу
Open FName For Input As FNum
                        'відкриття файлу
For I = 0 To Y-1        'організація циклу читання
                        'рядків файлу
  Line Input #FNum, S   'читання рядка з файлу
  N = 0                 'змінна N очищується
  S1 = ""               'змінна S1 очищується
  For A = 1 To Len(S)   'організація циклу перебирання
                        'символів у рядку. Цикл
                        'виконується стільки разів,
                        'скільки символів у рядку.
                        'Довжину рядка визначено
                        'функцією Len(S)
    If Mid(S, A, 1) = ";" Then
                        'виділення з рядка S одного
                        'символу, причому якщо
                        'виділений символ ";",то...
     N = N + 1          'лічильник стовпців
                        'збільшується на 1
     DB(N-1,I) = Val(S1)'значення змінної S1
                        'перетворюється з рядка на
                        'число і завантажується у масив
     S1 = ""            'змінна S1 очищується
    Else                'якщо виділений символ не ";",
                        'то...
     S1 = S1 + Mid(S, A, 1)
                        'виділення з рядка S одного
                        'символу і додавання його до
                        'змінної S1
    End If              'кінець умови
  Next A                'кінець циклу читання символів
                        'з рядка
```

```
  DB(X – 1, i) = Val(S1)'значення змінної S1
                        'перетворюється з рядка на
                        'число і завантажується у масив
Next I                  'кінець циклу читання рядка з
                        'файлу
Close FNuml             'закриття файлу
End Sub                 'кінець процедури
```

5.4. Робота з елементами управління

5.4.1. Завантаження даних з текстового файлу в елемент управління ComboBox

Дані з текстового файлу послідовно зчитуються у вигляді рядків і додаються у список елементу управління **ComboBox**. Зчитування відбуватиметься доки не буде досягнуто кінець файлу.

```
FNamel = "d:\data.txt"  'повне ім'я файлу
FNuml = FreeFile        'надання номера файлу
Open FNamel For Input As FNuml
                        'відкриття файлу
While Not EOF(FNuml)    'організація циклу читання
                        'файлу. Цикл працює, доки не
                        'досягнуто кінець файлу
  Line Input #FNum, S   'читання рядка з файлу
  ComboBox5.AddItemS    'завантаження рядка у елемент
                        'управління
Wend                    'кінець циклу
Close FNuml             'закриття файлу
```

5.4.2. Створення проектної підсистеми для округлення чисел

Розглянемо проектну інваріантну підсистему для округлення введеного користувачем числа до цілого, до 5, до 10 та до рядів переважних чисел.

Проектування форми користувача. На формі користувача розташовуються елементи управляння згідно з рис. 5.1.

Для виведення результатів округлення на формі користувача розташовано таблицю з трьома стовпцями зі списків **ListBox.**

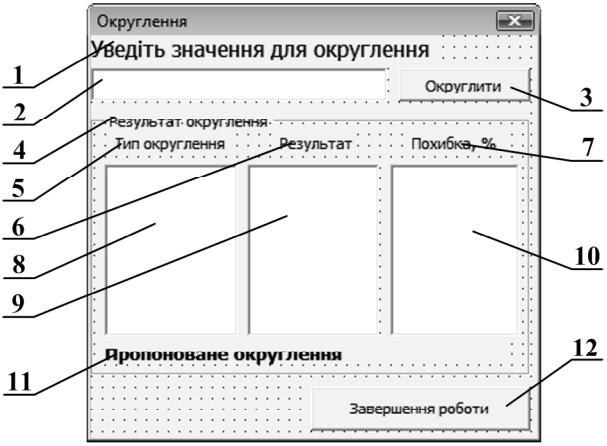


Рис. 5.1. Форма користувача та елементи управління: *1* – надпис **Label1**; *2* – поле для введення **TextBox1**; *3* – кнопка **CommandButton1** з надписом «**Округлити**»; *4* – рамка **Frame1**; *5–7* – надписи **Label2**–**Label4**; *8–10* – списки **ListBox1**–**ListBox3**; *11* – надпис **Label5**; *12* – кнопка **CommandButton2** з надписом «**Завершення роботи**»

У першому стовпці розміщується тип округлення, у другому – результат округлення, у третьому – значення похибки у %.

Налаштування форми та елементів управління. При розробленні форми користувача у менеджері властивостей установлюються наступні параметри:

– для форми `UserForm1`:

```
UserForm1.Caption = "Округлення"
```

– для надпису **Label1**:

```
Label1.Caption = "Уведіть число для округлення"
Label1.Font.Size = 12
```

– для надпису **Label2**:

```
Label2.Caption = "Тип округлення"
```

– для надпису **Label3**:

```
Label3.Caption = "Результат"
```

– для надпису **Label4**:

```
Label4.Caption = "Похибка, %"
```

– для надпису **Label5**:

```
Label5.Caption = "Пропоноване округлення"
```

– для рамки **Frame1**:

```
Frame1.Caption = "Результат округлення"
```

– для кнопки **CommandButton1**:

```
CommandButton1.Caption = "Округлити"
```

– для кнопки **CommandButton2**:

```
CommandButton2.Caption = "Завершення роботи"
```

Проектна підсистема містить головну програму та 2 обробники подій форми та елементів керування:

– обробник події натиснення кнопки **CommandButton1_Click()** для розрахунку округлення числа, уведеного користувачем;

– обробник події натиснення кнопки **CommandButton2_Click()** для завершення роботи.

Головна програма

```
Sub main()
   UserForm1.Show       'запуск форми користувача
End Sub
```

Обробник події натиснення кнопки **CommandButton1**

```
Private Sub CommandButton1_Click()
Dim N As Double
Dim Nr(1, 6) As Double  'масив для таблиці округлених
                        'чисел та похибок округлення
N = Val(TextBox1.Text)  'завантаження рядка, уведеного
                        'користувачем та переведення
                        'його у число
If N = 0 Tnen MsgBox "Не вірно введені початкові дані",
48, "Помилка"           'перевірка на коректність
                        'уведених даних користувачем.
                        'Якщо введено нульове
                        'значення, то запускається
                        'вікно повідомлення про
                        'помилкове введення числа для
                        'округлення (рис. 7.2.),
  Else                  'інакше робота процедури
                        'продовжується
      'Завантаження у перший стовпець таблиці назв
                  'способів округлення
ListBox1.Clear          'очищення списку 1
ListBox1.AddItem ("До цілого")
ListBox1.AddItem ("До 5")
ListBox1.AddItem ("До 10")
ListBox1.AddItem ("До ряду R5")
ListBox1.AddItem ("До ряду R10")
ListBox1.AddItem ("До ряду R20")
```

```
ListBox1.AddItem ("До ряду R40")
   'Визначення значень округлених чисел та запис їх у
                                'масив
Nr(0, 0) = Round(N, 0)  'округлення до цілого
Nr(0, 1) = Round_5(N)   'округлення до 5
Nr(0, 2) = Round_10(N)  'округлення до 10
Nr(0, 3) = Normalize(N, 5)
                        'округлення до ряду переважних
                        'чисел R5
Nr(0, 4) = Round(Normalize(N, 10), 2)
                        'округлення до ряду переважних
                        'чисел R10
Nr(0, 5) = Round(Normalize(N, 20), 2)
                        'округлення до ряду переважних
                        'чисел R20
Nr(0, 6) = Round(Normalize(N, 40), 2)
                        'округлення до ряду переважних
                        'чисел R40
   'Завантаження у другий стовпець округлених значень
ListBox2.Clear          'очищення списку ListBox2
For A = 0 To 6          'організація циклу завантаження
                        'округлених чисел у колонку
                        'таблиці Результат
  ListBox2.AddItem (Str(Nr(0, A)))
                        'завантаження округлених чисел
                        'у список ListBox2
Next A                  'завершення циклу
            'Визначення похибок округлення
Nr(1, 0) = Round((Abs(Nr(0, 0) - N) * 100) / N, 4)
Nr(1, 1) = Round((Abs(Nr(0, 1) - N) * 100) / N, 4)
Nr(1, 2) = Round((Abs(Nr(0, 2) - N) * 100) / N, 4)
Nr(1, 3) = Round((Abs(Nr(0, 3) - N) * 100) / N, 4)
Nr(1, 4) = Round((Abs(Nr(0, 4) - N) * 100) / N, 4)
Nr(1, 5) = Round((Abs(Nr(0, 5) - N) * 100) / N, 4)
Nr(1, 6) = Round((Abs(Nr(0, 6) - N) * 100) / N, 4)
   'Завантаження у третій стовпець похибок округлення
ListBox3.Clear          'очищення списку ListBox3
For A = 0 To 6          'організація циклу завантаження
                        'похибок округлення у колонку
                        'таблиці Похибка
  ListBox3.AddItem (Str(Nr(1, A)))
                        'завантаження похибок
                        'округлення у список ListBox3
Next A                  'завершення циклу
     'Визначення округлення з мінімальною похибкою
Dim Min As Double       'змінна для мінімального числа
Min = Nr(1, 0)          'присвоєння первинного значення
For A = 0 To 6          'організація циклу для перевірки
                        'усіх елементів масиву
```

```
 If Mix > Nr(1, A) Then 'якщо значення змінної для
                        'мінімального числа більше за
                        'поточний елемент масиву, то,
   Mix = Nr(1, A)       'змінна для мінімального числа
                        'дорівнює поточному елементу
                        'масиву
   N1 = A               'змінна N1 дорівнює індексу
                        'поточного елементу масиву
   End If               'завершення умовного блоку
Next A                  'завершення циклу
Label5.Caption = "Пропоноване округлення " + Str(Nr(0, n1))
                        'пропоноване користувачеві
                        'округлення з мінімальною
                        'похибкою
End If                  'завершення умовного блоку
End Sub                 'завершення процедури
```

Обробник події натиснення кнопки **CommandButton2**.

```
Private Sub CommandButton2_Click()
   Unload UserForm1     'закриття форми користувача
End Sub                 'завершення процедури
```

Проектна інваріантна підсистема працює так. Вихідні дані, що формує користувач, зокрема число для округлення, перевіряються на коректність. Якщо введено нульове значення, то запускається вікно повідомлення про помилкове введення числа для округлення (рис. 5.2), інакше робота процедури продовжується.

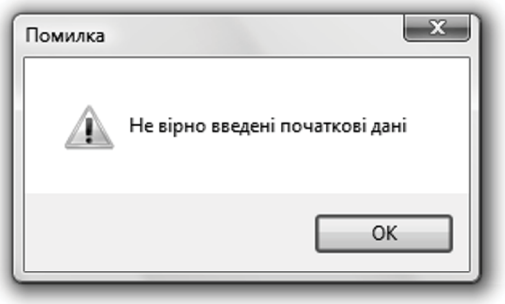


Рис. 5.2. Вікно повідомлення

У підсистемі використовуються функції, розглянуті вище, а саме: округлення чисел до 5 (**Round_5**), округлення чисел до 10 (**Round_10**), приведення чисел до переважного ряду (**Normalize**). За допомогою цих функцій заповнюється табли-

ця: визначаються округлені значення до цілого, до 5, до 10 та до рядів переважних чисел. Визначаються похибки округлення для кожного з варіантів. Як обмеження використовуються умови завдання оптимізації – мінімізація похибки між уведеним та округленим числами.

Блок розрахункового модуля містить процедуру з пошуку оптимального рішення, тобто мінімальної похибки між уведеним та округленим числами. За мінімальною похибкою знаходиться число, пропоноване користувачеві, як результат округлення.

Приклад роботи процедури наведено на рис. 5.3. Користувач увів значення 347,83. Рекомендоване округлене число з найменшою похибкою буде 348. Це округлення до найближчого цілого числа. Похибка становитиме 0,5714%.

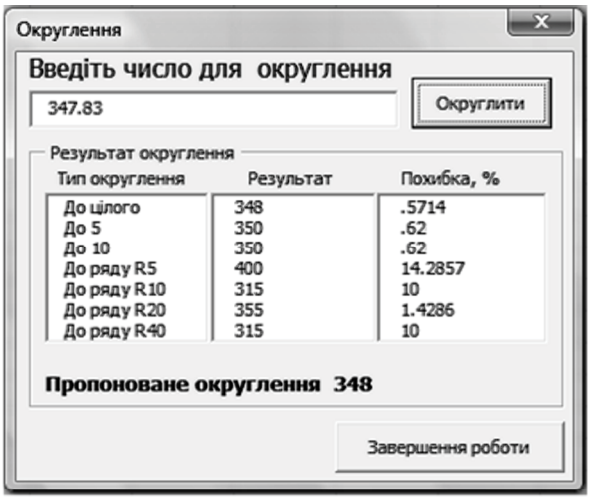


Рис. 5.3. Приклад роботи процедури

Контрольні завдання

1. Нехай задано масив M(15, 15) типу `Double`. Створіть функцію, що визначає кількість від'ємних елементів масиву.
2. Нехай задано масив V(9, 9) типу `Double`. Створіть функцію, що визначає середнє геометричне значення усіх його елементів.

3. Нехай задано одновимірний масив V(50) типу `Double`. Створіть функцію, що визначає мінімальне значення елементів масиву.
4. Нехай задано одновимірний масив V(3 to 40) типу `Double`. Створіть функцію, що розсортовує числа у масиві за спаданням.
5. Організуйте завантаження даних з текстового файлу в елемент управління **`ListBox`**. Передбачити можливість сортування списку за зростанням чи спаданням.
6. Побудуйте блок-схему частини програми для пошуку кількості елементів масиву, які менше за число K (див. завдання 1, п. 5.1).
7. Побудуйте блок-схему частини програми для визначення середнього арифметичного усіх елементів масиву (див. завдання 2, п.5.1).
8. Побудуйте блок-схему частини програми для визначення максимального значення елементів масиву (див. завдання 3, п. 5.1).
9. Побудуйте блок-схему частини програми для розсортування елементів масиву за зростанням їх значень (див. завдання 4, п. 5.1).
10. Побудуйте блок-схему функції округлення чисел до 5 (див. п. 5.2).
11. Побудуйте блок-схему функції округлення чисел до 10 (див. п. 5.2).
12. Побудуйте блок-схему функції приведення розмірів до рядів переважних чисел (див. п. 5.2).
13. Побудуйте блок-схему функції визначення рядків у текстовому файлі (див. п. 5.2.1).
14. Побудуйте блок-схему функції читання визначеного рядка з текстового файлу (див. п. 5.2.2).
15. Побудуйте блок-схему функції визначення кількості стовпців таблиці у CSV-файлі (див. п. 5.2.3).
16. Побудуйте блок-схему функції завантаження таблиці з CSV-файлу в динамічний масив (див. п. 5.2.4).
17. Побудуйте блок-схему частини програми для завантаження даних з текстового файлу в елемент управління **`ComboBox`** (див. п. 5.3.1).

6. Створення макросів для роботи з документами SolidWorks

6.1. Рекомендації щодо роботи з документами за допомогою SolidWorks API

Програмний інтерфейс SolidWorks API для роботи з основними документами використовує об'єктно-орієнтовану модель. Це сотні методів та властивостей, які застосовуються до об'єктів та забезпечують прямий доступ до функцій SolidWorks. Відтак SolidWorks API розглядає кожну модель, складання та креслення як окремий об'єкт, що містить властивості і методи (аналогічно до елементів управління на формі користувача).

Методи побудови елементів та об'єктів при створенні нових моделей у документах засобами SolidWroks API є складними і мають багато аргументів. Тому рекомендованим способом роботи з ними є попередній запис дій користувача з побудови моделей у макрос, а потім редагування, оптимізація та налагодження отриманого програмного коду (рис. 6.1).

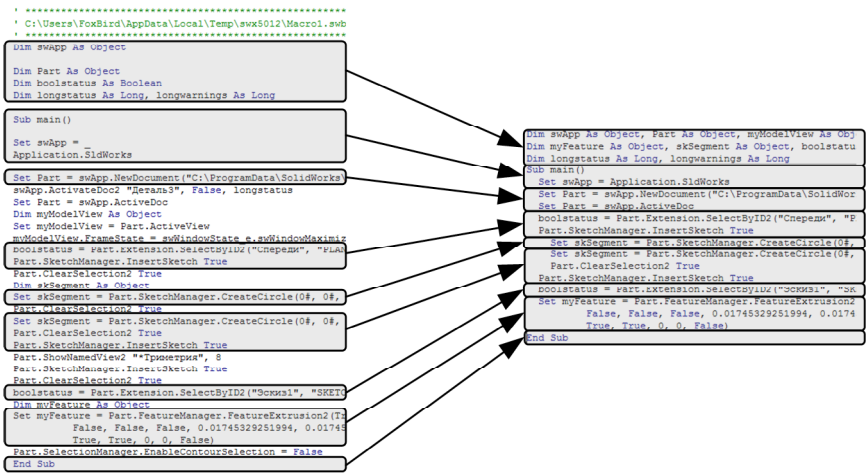


Рис. 6.1. Оптимізація коду

Автоматично записаний код у макросі не є оптимальним, оскільки містить багато зайвих (а іноді навіть суперечливих) команд і потребує редагування та оптимізації. Саме тому наведені далі синтаксиси методів побудови елементів та об'єктів є більшою мірою довідковим матеріалом. Запам'ятовувати слід загальні послідовності використання цих методів, наведених у вигляді схем.

Розглянемо базові методи для роботи з основними документами.

6.2. Відкриття наявних документів

Послідовність відкриття наявних документів SolidWorks наведено на рис 6.2.

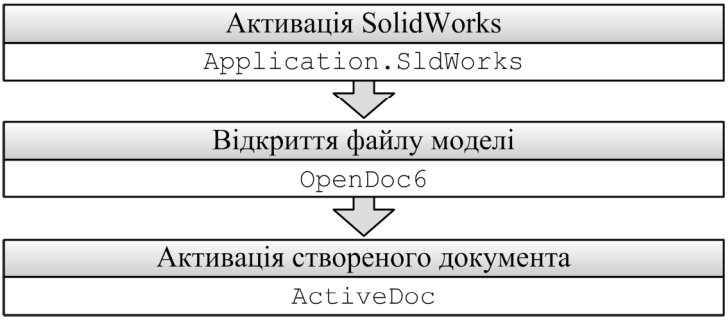


Рис. 6.2. Послідовність відкриття наявних документів SolidWorks

Метод **`Application`** є основним при розробці макросу для роботи з документами SolidWorks. Адже від створює головний батьківський об'єкт. Цей метод для активації SolidWorks не має аргументів.

Метод **`ActiveDoc`** для активації створеного документа не мають аргументів.

Метод **`OpenDoc6`** є універсальним для створення основних типів документів SolidWorks.

Синтаксис методу **`OpenDoc6`**:

```
newDoc = SldWorks.OpenDoc6(FN, T, O, C, E, W)
```

Аргументи методу **`OpenDoc6`** наведено в табл. 6.1.

Таблиця 6.1

Аргументи методу `OpenDoc6`

Аргументи	Тип	Призначення
FN	string	Шлях та ім'я файлу моделі для відкриття.
T	integer	Тип документа, що відкривається, визначається системною змінною **swDocumentTypes_e** [10]: 0 – жодного документа не відкрито; 1 – відкрито документ **Деталь**; 2 – документ **Сборка**; 3 – відкрито документ **Чертеж**.
O	integer	Режим, у якому відкривається документ, визначається системною змінною **swOpenDocOptions_e** [10]: 1 – тихе відкриття документа; 2 – відкриття тільки для читання; 4 – відкриття складання у режимі **Просмотр большой сборки**; 8 – відкриття документа креслення у роз'єднаному режимі з документом моделі; 16 – відкриття документа креслення і пов'язаного з ним документа моделі; 32 – при відкритті документа застосувати недавно використану конфігурацію; 64 – скасувати параметри за замовчування при відкритті складання у спрощеному режимі; 128 – відкриття складання у спрощеному режимі; 256 – при відкритті складання приховані компоненти теж завантажуються.
C	string	Назва конфігурації моделі (якщо їх декілька), у якій відкривається модель.
E	integer	Код помилки, що визначається системною змінною **swFileLoadError_e** [10].
W	integer	Код попередження, що визначається системною змінною **swFileLoadWarning_**e [10].

Приклад

```
Set swApp = Application.SldWorks
                          'ініціалізація пакета SolidWorks
Set Part = swApp.OpenDoc6("C:\Деталь.prtdot", 1, 0, "",
longstatus,longwarnings) 'відкриття наявного документа
                          'Деталь.prtdot
Set Part = swApp.ActiveDoc
                          'активація відкритого документа
```

6.3. Створення нового документа

Послідовність створення нового документа SolidWorks наведено на рис. 6.3.

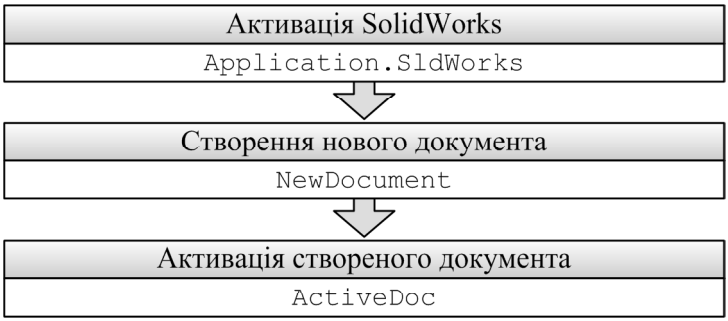


Рис. 6.3. Послідовність створення нового документа SolidWorks

Синтаксис методу **NewDocument**:

```
newDoc = SldWorks.NewDocument(tN, pS, w, h)
```

Аргументи методу **NewDocument** наведено в табл. 6.2.

Таблиця 6.2

Аргументи методу **NewDocument**

Аргументи	Тип	Призначення
tN	string	Шлях та ім'я файлу моделі, що створюється.
pS	double	Розмір (формат) аркуша креслення, що визначається системною змінною **swDwgPaperSizes_e** [10] (тільки для креслень): 0 – A горизонтальний; 7 – A4 вертикальний; 1 – A вертикальний; 8 – A3 горизонтальний; 2 – B горизонтальний; 9 – A2 горизонтальний; 3 – C горизонтальний; 10 – A1 горизонтальний; 4 – D горизонтальний; 11 – A0 горизонтальний; 5 – E горизонтальний; 12 – формат, визначений користувачем. 6 – A4 горизонтальний;
w	double	Висота аркуша креслення, якщо системна змінна **swDwgPaperSizes_e = 12** (тільки для креслень).
h	double	Ширина аркуша креслення, якщо системна змінна **swDwgPaperSizes_e = 12** (тільки для креслень).

Приклади

Створення нового документа деталі.

```
Set swApp = Application.SldWorks
                        'ініціалізація пакета SolidWorks
Set Part = swApp.NewDocument("C:\ProgramData\SolidWorks\
SolidWorks2011\templates\Деталь.prtdot", 0, 0#, 0#)
                        'створення нового документа з
                        'шаблону Деталь.prtdot
Set Part = swApp.ActiveDoc
                        'активація створеного документа
```

Створення нового документа складання.

```
Set swApp = Application.SldWorks
                        'ініціалізація пакету SolidWorks
Set Part = swApp.NewDocument("C:\ProgramData\SolidWorks\
SolidWorks2011\templates\Сборка.prtdot", 0, 0#, 0#)
                        'створення нового документа з
                        'шаблону Сборка.prtdot
Set Part = swApp.ActiveDoc
                        'активація створеного документа
```

Створення нового документа креслення.

```
Set swApp = Application.SldWorks
                        'ініціалізація пакету SolidWorks
SetPart = swApp.NewDocument("C:\ProgramData\SolidWorks\
SolidWorks2011\templates\Чертеж.prtdot", 12, 0.2794,
0.4318)                 'створення нового документа з
                        'шаблону Чертеж.prtdot, формат
                        'аркуша 2, ширина – 0.2794 м,
                        'висота – 0.4318 м.
Set Part = swApp.ActiveDoc
                        'активація створеного документа
```

Для визначення шаблону основного надпису та масштабу аркуша креслення виживається метод для редагування параметрів аркуша **SetupSheet5**. Послідовність редагування аркуша креслення не має ніяких попередніх дій. Метод використовується після відкриття (або створення) та активації креслення. Спроба застосувати цей метод при роботі з деталлю або складанням спричинить помилку.

Синтаксис:

```
Retval = Part.SetupSheet5( N, PS, TI, S1, S2, FA,
TN, W, H, PWN)
```

Аргументи методу **SetupSheet5** наведено в табл. 6.3.

Таблиця 6.3

Аргументи методу **SetupSheet5**

Аргумент	Тип	Призначення
N	string	Ім'я аркуша
PS	integer	Розмір (формат) аркуша креслення, що визначається системною змінною **swDwgPaperSizes_e** [10]: 0 – A горизонтальний; 7 – A4 вертикальний; 1 – A вертикальний; 8 – A3 горизонтальний; 2 – B горизонтальний; 9 – A2 горизонтальний; 3 – C горизонтальний; 10 – A1 горизонтальний; 4 – D горизонтальний; 11 – A0 горизонтальний; 5 – E горизонтальний; 12 – формат, визначений 6 – A4 горизонтальний; користувачем.
TI	integer	Формат шаблону основного надпису аркуша, визначений системною змінною **swDwgTemplates_e** [10]: 0 – A горизонтальний; 1 – A вертикальний; 2 – B горизонтальний; 3 – C горизонтальний; 4 – D горизонтальний; 5 – E горизонтальний; 6 – A4 горизонтальний; 7 – A4 вертикальний; 8 – A3 горизонтальний; 9 – A2 горизонтальний; 10 – A1 горизонтальний; 11 – A0 горизонтальний; 12 – формат, визначений користувачем; 13 – без основного надпису.
S1	double	Масштаб збільшення.
S2	double	Масштаб зменшення.
FA	boolean	True – використання напряму проекції за замовчуванням, False – змінити напрям проекції.
TN	string	Ім'я файлу шаблону основного надпису, якщо аргумент **TI =** 12.
W	double	Висота аркуша креслення, якщо аргумент **TI =** 12 або 13.
H	double	Ширина аркуша креслення, якщо аргумент **TI =** 12 або 13.
PWN	string	Ім'я виду моделі, з якого імпортуються значення параметрів.
RMN	boolean	True – видалити змінені примітки, False – не видаляти.

Приклад

```
boolstatus = Part.SetupSheet5("Лист1", 12, 12, 1, 1, True,
"a3-gost.slddrt", 0.42, 0.2794, "По умолчанию", True)
                          'встановлення шаблону основного
                          'надпису для аркуша А3 ГОСТ з
                          'файлу a3-gost.slddrt,
                          'визначеного користувачем.
```

При створенні нового документа система автоматично присвоює йому ім'я, що складається з назви типу документа та порядкового номера. Інколи при розробці програми виникає потреба знати це присвоєне автоматично ім'я. Для цього застосовується функція **GetTitle**.

Синтаксис:

```
value = instance.GetTitle()
```

Функція не має аргументів та повертає ім'я активного у даний час документа.

Приклад

```
Dim AT As string
AT = Part.GetTitle        'визначення назви документа
```

6.4. Збереження документа

Збереження основного документа у файл (пункти головного меню **Сохранить** і **Сохранить как**) виконується за допомогою методу **SaveAs**. Слід пам'ятати, що збереження відкритого документа у файл з тим самим ім'ям призведе до заміни вмісту цього файлу.

Синтаксис

```
instance.SaveAs NewName
```

де:

NewName – шлях та ім'я файлу для збереження.

Приклад

```
Part.SaveAs "D:\Деталь1.SLDPRT"
                          'збереження документа у файлі
                          'Деталь1.SLDPRT
```

6.5. Способи роботи з документами SolidWorks

Способи роботи з деталями, складаннями і кресленнями САПР SolidWorks наведено на рис. 6.4.

Рис. 6.4. Способи роботи з документами SolidWorks

Існують два способи роботи з документами – це зміна наявного документа або створення нового. Кожен із зазначених способів має свої недоліки те переваги.

Перевагою зміни наявного документа є простота та швидкість розроблення програми.

Основним недоліком такого способу є те, що зміна наявного документа унеможливлює істотну зміну геометрії. Змінювати можна лише розміри елементів побудови, рівняння та конфігурації. Отже, такий макрос орієнтовано на побудову тільки однієї деталі з незначною зміною її геометрії.

Основною перевагою створення нового документа є можливість істотної зміни геометрії моделі.

6.6. Зміна існуючих моделей

6.6.1. Зміна розмірів в існуючій моделі

Послідовність зміни розміру в документі SolidWorks наведено на рис. 6.5.

При використанні способу зміни розмірів модель повинна бути побудована раніше. Як зазначалося вище, можна змінювати значення розмірів, але неможливо змінювати порядок операцій побудови або їх тип.

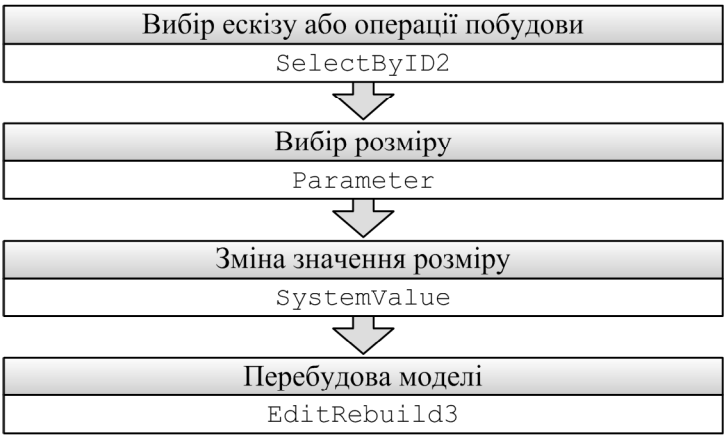


Рис. 6.5. Послідовність зміни розміру

Після зміни розміру модель потрібно зберегти, як правило, під новим ім'ям.

Для зміни значення розміру у моделі необхідно знати повне ім'я розміру, тобто в якому ескізі або операції побудови знаходиться необхідний розмір, його тип і номер.

Розглянемо методи, які використовуються при заміні розміру в документах SolidWorks.

Для вибору будь-якого об'єкта використовується універсальний метод **SelectByID2.** Метод дозволяє обирати будь-які об'єкти середовища SolidWorks. Вибір здійснюється двома способами: за унікальним ім'ям об'єкта або вказанням точки на ньому.

Синтаксис:

```
retval = Part.SelectByID2 (N,T,X,Y,Z,A,M,C,SO)
```

Аргументи методу **SelectByID2** наведено в табл. 6.4.

Приклади

```
boolstatus = Part.Extension.SelectByID2(«Эскиз1”,
“SKETCH”, 0, 0, 0, False, 0, Nothing, 0)
                          ‘вибір ескізу
boolstatus = Part.Extension.SelectByID2(«Бобышка-
вытянуть1”, “BODYFEATURE”, 0, 0, 0, False, 0, Nothing, 0)
                          ‘вибір операції побудови
```

Таблиця 6.4

Аргументи методу **SelectByID2**

<table>
<tr><th>Аргументи</th><th>Тип</th><th colspan="3">Призначення</th></tr>
<tr><td>N</td><td>string</td><td colspan="3">Ім’я об’єкта, що обирається або порожній рядок.</td></tr>
<tr><td>T</td><td>string</td><td colspan="3">Тип об’єкта, згідно зі змінною swSelectType_e [10]:
“EDGE” – кромка;
“FACE” – грань;
“VERTEX” – вершина;
“PLANE” – площина;
“AXIS” – вісь;
“SKETCH” – ескіз;
“SKETCHSEGMENT” – елемент ескізу;
“SKETCHPOINT” – точка у ескізі;
“DRAWINGVIEW” – вид на кресленні;
“NOTE” – примітка;
“DIMENSION” – розмір;
“SECTIONLINE” – лінія розрізу;
“COMPONENT” – деталь складання;
“MATE” – спряження деталей у складанні;
“BODYFEATURE” – елемент побудови.</td></tr>
<tr><td>X, Y, Z</td><td>double</td><td colspan="3">Позиція вибору X, Y, Z.</td></tr>
<tr><td rowspan="5">A</td><td rowspan="5">boolean</td><td>Якщо значення...</td><td>…і якщо…</td><td>..., то...</td></tr>
<tr><td rowspan="2">TRUE</td><td>об’єкт ще не обрано</td><td>об’єкт додається до списку обраних об’єктів.</td></tr>
<tr><td>об’єкт вже обрано</td><td>об’єкт видаляється зі списку обраних об’єктів.</td></tr>
<tr><td rowspan="2">FALSE</td><td>об’єкт ще не обрано</td><td>список обраних об’єктів очищується, і об’єкт обирається.</td></tr>
<tr><td>об’єкт вже обрано</td><td>список обраних об’єктів не змінюється.</td></tr>
<tr><td>M</td><td>integer</td><td colspan="3">Значення для інших функції.</td></tr>
<tr><td>C</td><td>callout</td><td colspan="3">Покажчик, пов’язаний з об’єктом callout.</td></tr>
<tr><td>SO</td><td>integer</td><td colspan="3">Опції вибору визначаються системною змінною swSelectOption_e [10].</td></tr>
</table>

Для скасування усіх виділень використовується метод **`ClearSelection2.`**

Синтаксис:

```
Part.ClearSelection2 All
```

де:

`All` – аргумент логічного типу. `True` – очищується увесь список обраних об'єктів, `False` – очищуться список активних об'єктів.

Після зміни розмірів модель, креслення або складання потрібно перебудувати, щоб проведені зміни було відображено. Для цього використовується метод **`EditRebuild3`**.

Синтаксис:

```
value = instance.EditRebuild3()
```

Метод не має аргументів.

Приклади зміни розмірів в основних документах

Розглянемо зміну розміру **`D4`** в ескізі **`Эскиз1`** наявної деталі (рис.6.6).

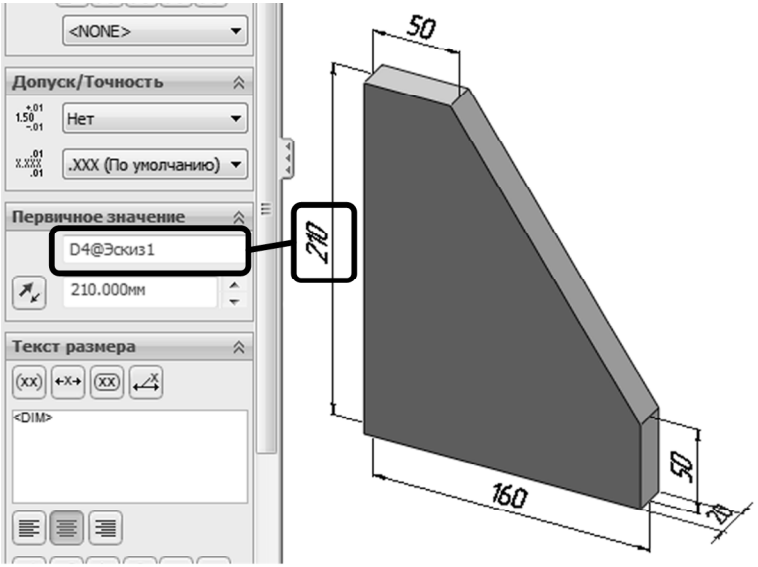


Рис. 6.6. Зміна значення розміру в ескізі

```
boolstatus = Part.Extension.SelectByID2(«Эскиз1”,
“SKETCH”, 0, 0, 0, False, 0, Nothing, 0)
                          ‘вибір ескізу, в якому
                          ‘потрібно змінити розмір
boolstatus = Part.Extension.SelectByID2(“D4”, “DIMEN-
SION”, 0, 0, 0, True, 0, Nothing, 0)
                          ‘вибір розміру D4 для
                          ‘редагування
Dim myDimension As Object
Set myDimension = Part.Parameter(“D4@Эскиз1”)
                          ‘ініціалізація розміру D4 з
                          ‘ескізу 1
myDimension.SystemValue = 0.268
                          ‘установлення нового значення
                          ‘розміру
Part.EditRebuild3()       ‘перебудова моделі
Part.ClearSelection2 True
                          ‘зняття виділення
```

Розглянемо зміну розміру **D1** в операції побудови **Бобышка-вытянуть1** наявної деталі (рис.6.7).

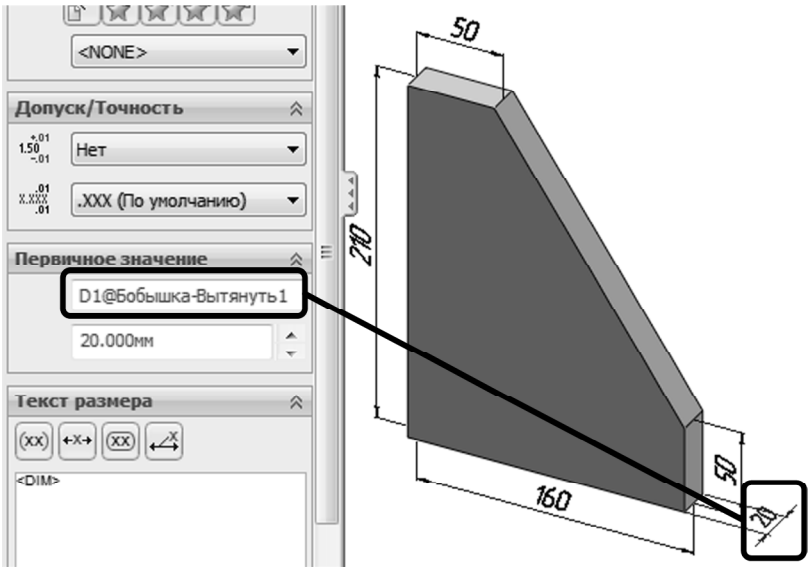


Рис. 6.7. Зміна значення розміру в операції побудови

```
boolstatus     =     Part.Extension.SelectByID2(«Бобышка–
вытянуть1”, “BODYFEATURE”, 0, 0, 0, False, 0, Nothing, 0)
                          ‘вибір операції, в якій
                          ‘потрібно змінити розмір
boolstatus      =      Part.Extension.SelectByID2(“D1”,
“DIMENSION”, 0, 0, 0, True, 0, Nothing, 0)
                          ‘вибір розміру D1 для
                          ‘редагування
Dim myDimension As Object
Set myDimension = Part.Parameter(“D1@«Бобышка–вытянуть1”)
                          ‘ініціалізація розміру D1 з
                          ‘операції побудови Бобышка–
                          ‘вытянуть1
myDimension.SystemValue = 0.014
                          ‘установлення нового значення
                          ‘розміру
Part.EditRebuild3()       ‘перебудова моделі
Part.ClearSelection2 True
                          ‘зняття виділення
```

Розглянемо зміну розміру **D4** на аркуші креслення (рис.6.8). Слід зазначити, що оскільки креслення побудовано на основі твердотільної моделі, то розмір слід змінювати саме у контексті цієї моделі.

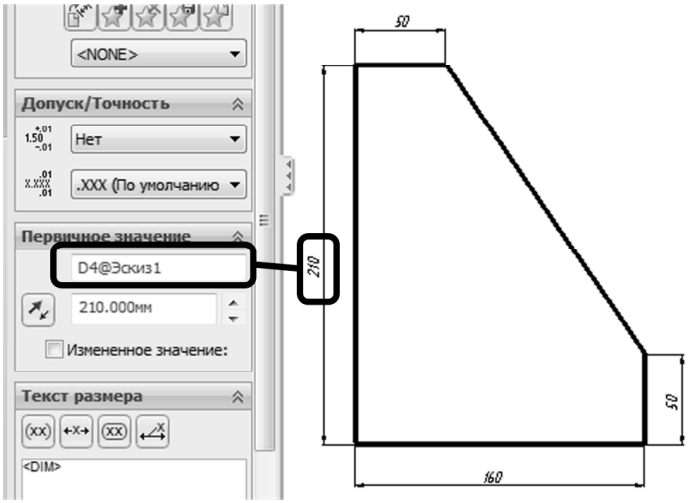


Рис. 6.8. Зміна значення розміру в кресленні

```
boolstatus = Part.Extension.SelectByID2("D1@Эскиз1@Де-
таль1-1@Чертежный вид1", "DIMENSION", 0, 0, 0, False,
0, Nothing, 0)            'вибір розміру для редагування
                          'з креслярського виду
Dim myDimension As Object
Set myDimension = Part.Parameter("D1@Эскиз1@Деталь1.Part")
                          'ініціалізація розміру D1 з
                          'ескізу 1 з файлу деталі
                          'Деталь1.Part
myDimension.SystemValue = 0.014
                          'установлення нового значення
                          'розміру
Part.EditRebuild3()       'перебудова креслення
Part.ClearSelection2 True
                          'зняття виділення
```

Розглянемо зміну розміру **D1** у спряженні **Растояние1** складання з двох деталей (рис.6.9).

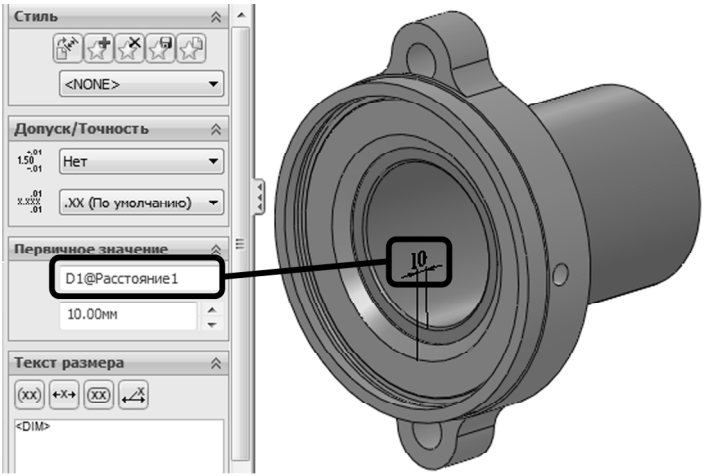


Рис. 6.9. Зміна значення розміру в складанні

```
boolstatus = Part.Extension.SelectByID2("D1@
Расстояние1@Сборка1.SLDASM", "DIMENSION", 0, 0, 0,
False, 0, Nothing, 0)    'вибір розміру спраження
Dim myDimension As Object
Set myDimension = Part.Parameter("D1@Расстояние1")
                          'ініціалізація розміру D1 зі
                          'спраження Расстояние1
```

```
myDimension.SystemValue = 0.015
                          'установлення нового значення
                          'розміру
Part.EditRebuild3()       'перебудова складання
Part.ClearSelection2 True
                          'зняття виділення
```

6.6.2. Зміна рівнянь у наявній моделі

Послідовність зміни розміру в документі SolidWorks наведено на рис. 6.10.

При зміні рівнянь модель повинна бути побудована раніше і містити управляючі рівняння. Особливість роботи з рівняннями полягає у тому, що вони не редагуються. Тобто для зміни рівняння у моделі необхідно видалити усі рівняння і переписати їх наново.

Рис. 6.10. Послідовність зміни управляючих рівнянь

Для видалення наявних рівнянь використовується метод **DeleteAllRelations**.

Для запуску менеджеру рівнянь використовується метод **GetEquationMgr**. Обидва методи не мають аргументів.

Для додавання рівнянь використовується метод **Add()**.

Синтаксис:

```
swEquationMgr.Add type, parametr
```

де:

type – тип рівняння [10];

parametr – вираз рівняння.

Приклад

Розглянемо зміну рівняння у твердотільній моделі (рис. 6.11).

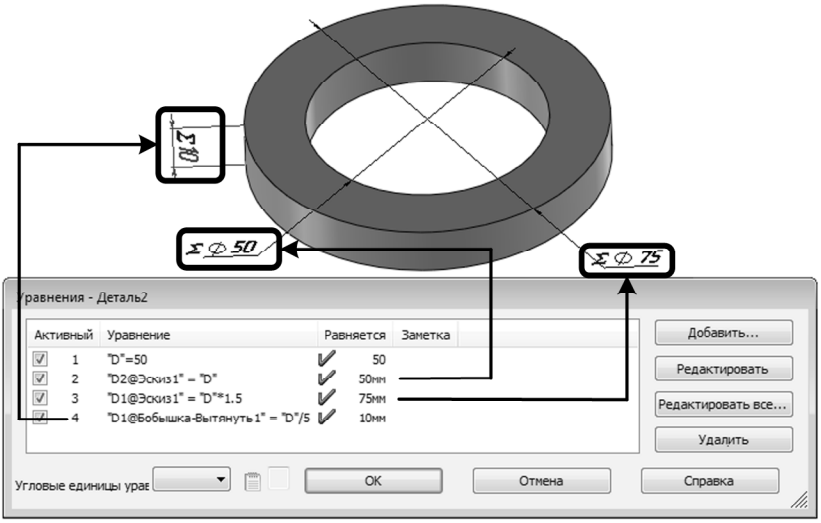


Рис. 6.11. Управляючі рівняння у моделі

```
boolstatus = Part.Extension.SelectByID2(«Эскиз1”, “SKETCH”,
0, 0, 0, False, 0, Nothing, 0)
                          'вибір ескізу, в якому потрібно
                          'змінити рівняння
Part.DeleteAllRelations 'видалення усіх рівнянь
       '–– Переписування усіх інших рівнянь ––
Dim swEquationMgr As Object
Set swEquationMgr = Part.GetEquationMgr()
                          'активація менеджера рівнянь
swEquationMgr.Add –1, “””D””=50”
                          'запис рівняння
SetswEquationMgr = Part.GetEquationMgr()
swEquationMgr.Add –1, “””D1@Эскиз1”” = ””D””
                          'запис рівняння
Set swEquationMgr = Part.GetEquationMgr()
swEquationMgr.Add –1, “””D2@Эскиз1”” = ””L””*1.25”
                          'запис зміненого рівняння
Set swEquationMgr = Part.GetEquationMgr()
swEquationMgr.Add –1, “””D1@Бобышка–
Вытянуть1””=””D””/5”    'запис рівняння
```

6.6.3. Зміна конфігурації в наявній моделі

Послідовність зміни розміру в документі SolidWorks наведено на рис. 6.12.

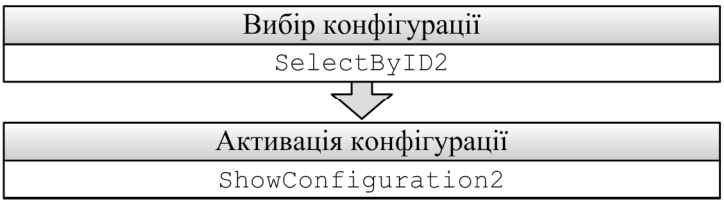


Рис. 6.12. Послідовність зміни активної конфігурації

Для активації обраної конфігурації використовується метод **ShowConfiguration2.**

Синтаксис

Part.ShowConfiguration2("Ім'я конфігурації")

де:

Ім'я конфігурації – конфігурація, яку потрібно активувати.

Приклад

Розглянемо зміну активної конфігурації в моделі.

```
boolstatus = Part.Extension.SelectByID2("Исполнение 1",
"CONFIGURATIONS", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір конфігурації
boolstatus = Part.ShowConfiguration2("Исполнение 1")
                          'активація обраної конфігурації
```

Послідовність увімкнення або вимкнення операцій побудови у поточній конфігурації наведено на рис 6.13.

Рис. 6.13. Послідовність увімкнення / вимкнення операцій побудови у конфігурації

Для увімкнення операції побудови використовується метод **EditSuppress2,** для вимкнення – метод **EditUnsuppress2**. Обидва методи не мають аргументів.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Вырез-
Вытянуть1", "BODYFEATURE", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір операції побудови
Part.EditSuppress2        'увімкнення операції побудови
Part.ClearSelection2 True
                          'скидання виділення
boolstatus = Part.Extension.SelectByID2("Вырез-
Вытянуть2", "BODYFEATURE", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір операції побудови
Part.EditUnsuppress2      'вимкнення операції побудови
```

6.7. Приклад розроблення підсистеми побудови косинок

Розглянемо проектну об'єктну підсистему для автоматизованого проектування косинок.

Проектна підсистема розробляється на основі зміни розмірів та активної конфігурації наявної моделі. Модель косинки побудована раніше і містить 2 конфігурації – без фаски та з фаскою. Після зміни розмірів модель зберігається під новим ім'ям.

Вихідні дані, що задаються користувачем – розміри косинки, товщина металу, наявність фаски на куті та її розміри.

Проектування форми користувача. На формі користувача розташовуються елементи управляння згідно з рис. 6.14.

При розробленні форми користувача у менеджері властивостей встановлюються такі параметри:

– для форми **UserForm1**:

```
UserForm1.Caption = "Косинки"
```

обробник події **UserForm_Initialize()**

– для надписів **Label1–12** у властивість **Caption** записуються значення згідно рис 6.14. Додатково для надписів **Label6**, **Label12** та поля для введення **TextBox5** у властивість **Visible** записується `False`.

– для рамки **Frame1**:

```
Frame1.Caption = "Вхідні дані"
```

– для прапорця **CheckBox1**:

```
CheckBox1.Caption = "Фаска"
```

обробник події зміни прапорця **CheckBox1_Click()**

– для кнопки **CommandButton1**:

```
CommandButton1.Caption = "Побудувати"
```

обробник події **CommandButton2_Click()**

– для кнопки **CommandButton2**:

```
CommandButton2.Caption = "Вихід"
```

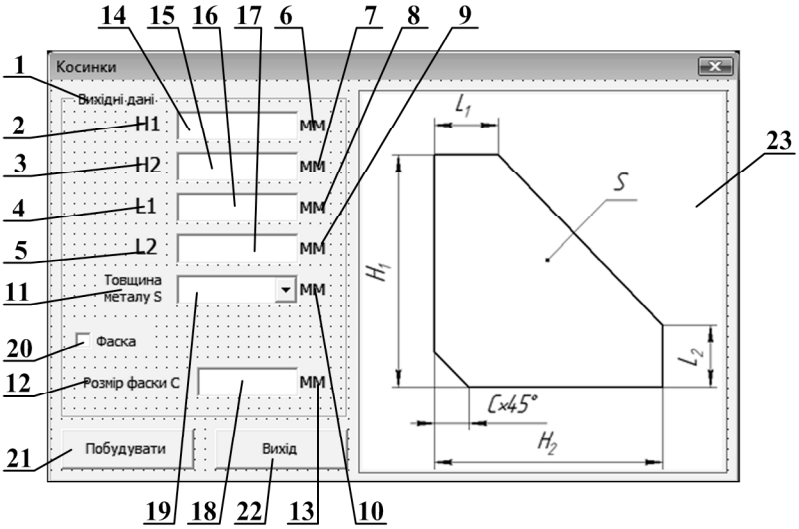


Рис. 6.14. Форма користувача та елементи управління: *1* – рамка **Frame1**; *2–13* – надписи **Label1** – **Label12**; *14–18* – поля для введення **TextBox1** – **TextBox5**; *19* – випадаючий список **ComboBox1**; *20* – позначка **CheckBox1**; *21* – кнопка **CommandButton1** з надписом «**Побудувати**»; *22* – кнопка **CommandButton2** з надписом «**Вихід**»; *23* – зображення **Image1**

Проектна підсистема містить головну програму та 4 обробники подій:

– обробник події створення форми користувача **Initialize**. При цьому в елемент управління **ComboBox1** завантажуються припустимі значення товщини металу косинки;

– обробник події зміни стану **CheckBox1** відображає або ховає елементи управління параметрів фаски косинки відповідно до вибору користувача;

– обробник події натиснення кнопки **CommandButton1** для перебудови косинки;
– обробник події натиснення кнопки **CommandButton2** для завершення роботи.

Головна програма

```
Sub main()
  UserForm1.Show          'запуск форми користувача
End Sub
```

Обробник події створення форми користувача

```
Private Sub UserForm_Initialize()
For A = 1 To 10          'організація циклу заповнення
                         'елементу управління
  ComboBox1.AddItem (Str(A))
                         'завантаження в елемент
                         'управління ComboBox1 значень
                         'товщини металу
Next A                   'завершення циклу
ComboBox1.Text = "1"     'визначення товщини металу за
                         'замовчуванням
End Sub                  'завершення процедури
```

Обробник події зміни стану **CheckBox1**

```
Private Sub CheckBox1_Click()
If CheckBox1.Value Then 'якщо прапорець фаски
                         'встановлено, то..
  Label12.Visible = True
                         'відобразити надпис Label12
  Label13.Visible = True
                         'відобразити надпис Label13
  TextBox5.Visible = True
                         'відобразити поле для введення
                         'TextBox5
Else                     '...інакше...
  Label12.Visible = False
                         'сховати надпис Label12
  Label13.Visible = False
                         'сховати надпис Label13
  TextBox5.Visible = False
                         'сховати поле для введення
                         'TextBox5
End If                   'завершення умови
End Sub                  'завершення процедури
```

Обробник події натиснення кнопки **CommandButton1** для побудови косинки

```
Private Sub CommandButton1_Click()
Dim H1 As Double, H2 As Double, L1 As Double
Dim L2 As Double, S As Double, C As Double
Dim swApp As Object, Dim Part As Object
Dim boolstatus As Boolean, myDimension As Object
Dim longstatus As Long, longwarnings As Long
                          'оголошення змінних
Set swApp = Application.SldWorks
                          'активація SolidWorks
Set Part = swApp.OpenDoc6("C:\косынка.SLDPRT", 1, 0, "",
longstatus, longwarnings)
                          'відкриття наявного документа
                          'косынка.SLDPRT
Set Part = swApp.ActiveDoc
                          'активація документа
   '-- Завантаження розмірів з елементів управління,
        'переведення їх з міліметрів у метри --
H1 = Val(TextBox1.Text) / 1000
H2 = Val(TextBox2.Text) / 1000
L1 = Val(TextBox3.Text) / 1000
L2 = Val(TextBox4.Text) / 1000
S = Val(ComboBox1.Text) / 1000
If CheckBox1.Value Then C = Val(TextBox5.Text) / 1000
If (H1<>0) And (H2<>0) And (L1<>0) And (L2<>0) Then
                          'перевірка коректності введення
                          'даних. Якщо дані введено
                          'вірно, то...
boolstatus = Part.Extension.SelectByID2("Эскиз1", "SKETCH",
0, 0, 0, False, 0, Nothing, 0)
                          'вибір Ескізу 1
boolstatus=Part.Extension.SelectByID2("D1","DIMENSION",
0, 0, 0, True, 0, Nothing, 0)
                          'вибір розміру D1
Set myDimension = Part.Parameter("D1@Эскиз1")
                          'визначення розміру D1
myDimension.SystemValue = H2
                          'встановлення значення розміру
Part.ClearSelection2 True
                          'скидання виділення
```

```
boolstatus = Part.Extension.SelectByID2("Эскиз1", "SKETCH",
0, 0, 0, False, 0, Nothing, 0)
                        'вибір ескізу 1
boolstatus = Part.Extension.SelectByID2("D2", "DIMENSION",
0, 0, 0, True, 0, Nothing, 0)
                        'вибір розміру D2
Set myDimension = Part.Parameter("D2@Эскиз1")
                        'визначення розміру D2
myDimension.SystemValue = H1
                        'встановлення значення розміру
Part.ClearSelection2 True
                        'скидання виділення
boolstatus = Part.Extension.SelectByID2("Эскиз1", "SKETCH",
0, 0, 0, False, 0, Nothing, 0)
                        'вибір ескізу 1
boolstatus = Part.Extension.SelectByID2("D3", "DIMENSION",
0, 0, 0, True, 0, Nothing, 0)
                        'вибір розміру D3
Set myDimension = Part.Parameter("D3@Эскиз1")
                        'визначення розміру D3
myDimension.SystemValue = L1
                        'встановлення значення розміру
Part.ClearSelection2 True
                        'скидання виділення
boolstatus = Part.Extension.SelectByID2("Эскиз1", "SKETCH",
0, 0, 0, False, 0, Nothing, 0)
                        'вибір ескізу 1
boolstatus = Part.Extension.SelectByID2("D4", "DIMENSION",
0, 0, 0, True, 0, Nothing, 0)
                        'вибір розміру D3
Set myDimension = Part.Parameter("D4@Эскиз1")
                        'визначення розміру D4
myDimension.SystemValue = L2
                        'встановлення значення розміру
Part.ClearSelection2 True
                        'скидання виділення
boolstatus    =    Part.Extension.SelectByID2("Бобышка-
вытянуть1", "BODYFEATURE", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір операції побудови
                        'Бобышка-вытянуть1
boolstatus     =     Part.Extension.SelectByID2("D1",
"DIMENSION", 0, 0, 0, True, 0, Nothing, 0)
                        'вибір розміру D1
```

```
Set myDimension = Part.Parameter("D1@Бобышка-вытянуть")
                        'визначення розміру D1
myDimension.SystemValue = S
                        'встановлення значення розміру
If CheckBox1.Value Then 'якщо косинка з фаскою, то...
boolstatus  =  Part.Extension.SelectByID2("С  фаской",
"CONFIGURATIONS", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір конфігурації "С фаской"
boolstatus = Part.ShowConfiguration2("С фаской")
                        'активація конфігурації
                        '"С фаской"
boolstatus   =   Part.Extension.SelectByID2("Фаска1",
"BODYFEATURE", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір операції побудови Фаска1
boolstatus=Part.Extension.SelectByID2("D1","DIMENSION",
0, 0, 0, True, 0, Nothing, 0)
                        'вибір розміру D1
Set myDimension = Part.Parameter("D1@Фаска1")
                        'визначення розміру D1
myDimension.SystemValue = C
                        'встановлення значення розміру
Else                    'інакше...
boolstatus  =  Part.Extension.SelectByID2("Без  фаски",
"CONFIGURATIONS", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір конфігурації "Без фаски"
boolstatus = Part.ShowConfiguration2("Без фаски")
                        'активація конфігурації
                        '"Без фаски"
End If                  'кінець блоку прийняття рішень
boolstatus = Part.EditRebuild3()
                        'перебудувати модель
longstatus = Part.SaveAs("C:\косынка2.SLDPRT")
                        'збереження документу під новим
                        'ім'ям
Else MsgBox "Не вірно введені початкові дані", 48,
"Помилка"'якщо вихідні дані введено
                        'невірно, відображається вікно
                        'повідомлення про помилку
                        '(рис 5.2).
End Sub                 'завершення процедури
```

Обробник події натиснення кнопки **CommandButton2** завершення роботи макросу.

```
Private Sub CommandButton2_Click()
  Unload UserForm1          'закриття форми користувача
End Sub                     'завершення процедури
```

Приклад роботи проектної підсистеми наведено на рис 6.15.

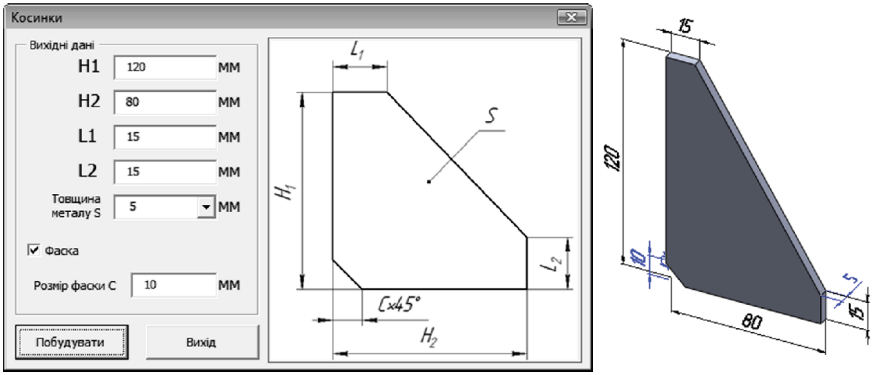


Рис. 6.15. Приклад роботи проектної процедури

Контрольні запитання і завдання

1. Як у SolidWorks API розглядається кожен основний документ SolidWorks API?
2. Яка послідовність дій та які методи використовуються для відкриття існуючих основних документів SolidWorks?
3. Назвіть послідовність дій та методи, що використовуються для створення нового основного документу.
4. Яка послідовність дій та які методи використовуються для збереження основного документа у файл?
5. Які існують способи роботи з деталями, складаннями і кресленнями за допомогою SolidWorks API?
6. Назвіть переваги та недоліки зміни моделі методами SolidWorks API.
7. Назвіть переваги та недоліки розроблення нової моделі методами SolidWorks API.

8. Які існують способи зміни моделі за допомогою SolidWorks API?
9. Назвіть послідовність дій та методи, що використовуються для зміни розмірів.
10. У чому полягає особливість роботи при редагуванні рівнянь у документі моделі?
11. Яка послідовність дій та які методи використовуються для зміни управляючих рівнянь?
12. Назвіть послідовність дій та методи, що використовуються для зміни активної конфігурації.
13. Яка послідовність дій та які методи використовуються для увімкнення / вимкнення операцій побудови у конфігурації?
14. Побудуйте блок-схеми процедур обробників подій проектної підсистеми побудови косинок.
15. Створіть підсистему побудови шпонок сегментних за ГОСТ 24071-80. Передбачити можливість побудови двох видів виконань.
16. Створіть підсистему побудови рейок кранових ДСТУ 2484-94 (ГОСТ 4121-96) за типом рейки та її довжиною.

7. Основні методи побудови ескізів

7.1. Створення ескізів

Послідовність побудови ескізу наведено на рис. 7.1.

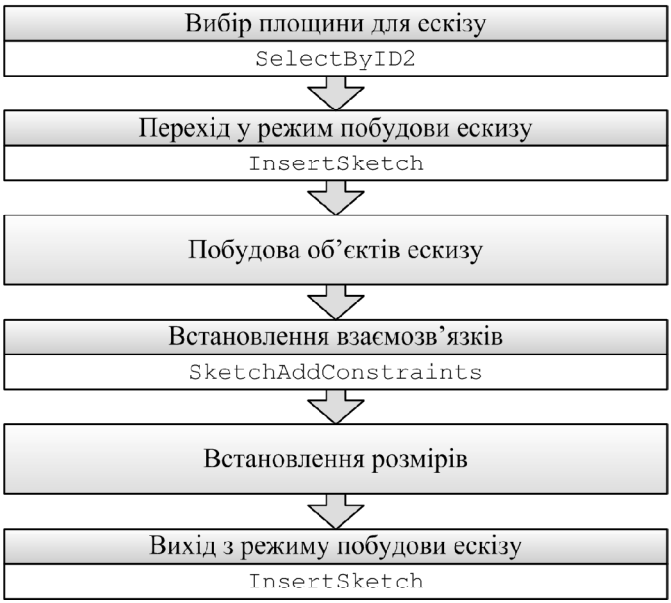


Рис. 7.1. Послідовність створення ескізу

Для відкриття/закриття режиму побудови ескізу використовується метод **`InsertSketch`**.

Синтаксис

`Part.SketchManager.InsertSketch UER`

де:

`UER` – аргумент логічного типу. `True` – вийти з режиму побудови ескізу зі збереженням створених змін, `False` – вийти з режиму побудови ескізу без збереження змін.

Приклад

```
Part.SketchManager.InsertSketch True
                              'відкрити режим побудови ескізу
```

7.2. Методи побудови об'єктів ескізу

Розглянемо основні методи побудови об'єктів ескізу.

Побудова прямого відрізку

Синтаксис (рис. 7.2)

```
Retval = SketchManager.CreateLine (X1, Y1, Z1,
X2, Y2, Z2)
```

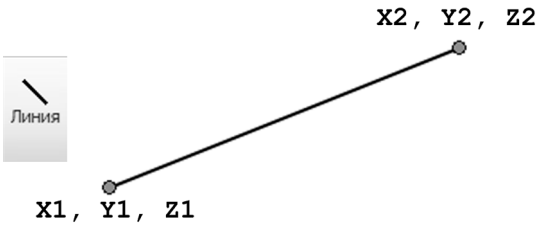


Рис. 7.2. Побудова прямого відрізку

Приклад

```
Set SkLine = Part.SketchManager.CreateLine(-0.09,
0.06, 0, -0.03, 0.08, 0)
```

Побудова осьового прямого відрізку

Синтаксис (рис. 7.3)

```
Retval = SketchManager.CreateCenterLine ( X1, Y1,
Z1, X2, Y2, Z2)
```

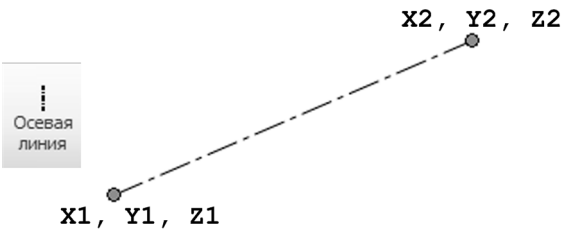


Рис. 7.3. Побудова осьового прямого відрізка

Приклад

```
Set SkLine = Part.SketchManager.CreateCenterLine(
-0.08, 0.04, 0, -0.01, 0.06, 0)
```

Побудова кутового прямокутника

Синтаксис (рис. 7.4)

```
Retval = SketchManager.CreateCornerRectangle (
X1, Y1, Z1, X2, Y2, Z2)
```

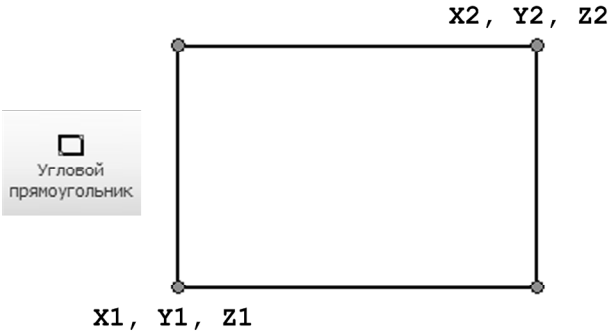


Рис. 7.4. Побудова кутового прямокутника

Приклад

```
vSkLines = Part.SketchManager.CreateCornerRectan-
gle(-0.06, 0.01, 0, -0.01, -0.01, 0)
```

Побудова прямокутника з центру

Синтаксис (рис. 7.5)

```
Retval = SketchManager.CreateCenterRectangle (
X1, Y1, Z1, X2, Y2, Z2)
```

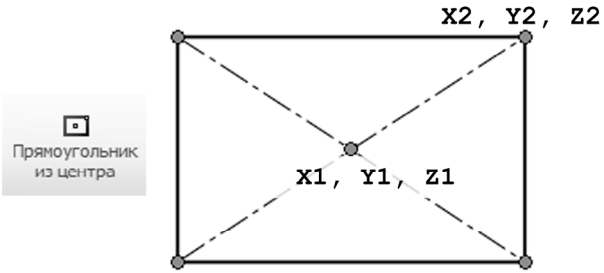


Рис. 7.5. Побудова прямокутника з центру

Приклад

```
vSkLines = Part.SketchManager.CreateCornerRectangle(
-0.06, 0.01, 0, -0.01, -0.01, 0)
```

Побудова кутового прямокутника за 3–ма точками

Синтаксис (рис. 7.6)

```
Retval = SketchManager.Create3PointCornerRectangle(
X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)
```

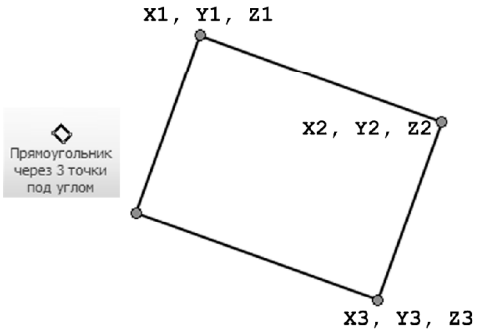


Рис. 7.6. Побудова кутового прямокутника за 3–ма точками

Приклад

```
vSkLines = Part.SketchManager.Create3PointCornerRectangle(
0.05, 0.05, 0, 0.08, 0.02, 0, 0.06, –0.001, 0)
```

Побудова прямокутника з центру за 3–ма точками

Синтаксис (рис. 7.7)

```
Retval = SketchManager.Create3PointCenterRectangle(
X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)
```

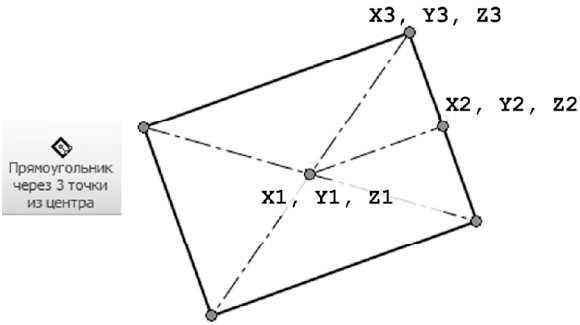


Рис. 7.7. Побудова прямокутника з центру за 3–ма точками

Приклад

```
vSkLines = Part.SketchManager.Create3PointCenterRect-
angle(0.1, –0.04, 0, 0.1, –0.02, 0, 0.1, –0.009, 0)
```

Побудова паралелограма

Синтаксис (рис. 7.8)

```
Retval = SketchManager.CreateParallelogram (X1,
Y1, Z1, X2, Y2, Z2, X3, Y3, Z3)
```

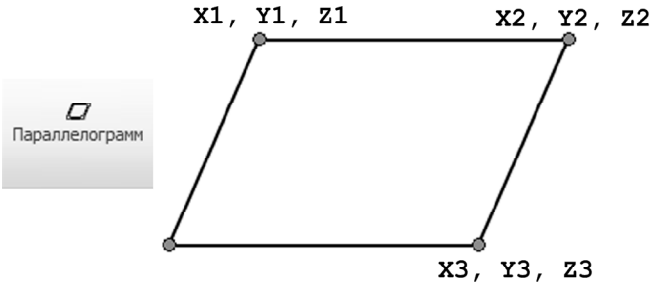


Рис. 7.8. Побудова паралелограма

Приклад

```
vSkLines = Part.SketchManager.CreateParallelogram(0.1,
0.07, 0, 0.1, 0.06, 0, 0.1, 0.05, 0)
```

Побудова багатокутника

Синтаксис (рис. 7.9)

```
Retval = SketchManager.CreatePolygon ( XC, YC,
ZC, XP, YP, ZP, Sides, Inscribed)
```

де:

Sides – кількість сторін багатокутника;

Inscribed – логічний аргумент. `True` – будується вписане коло, `False` – будується описане коло.

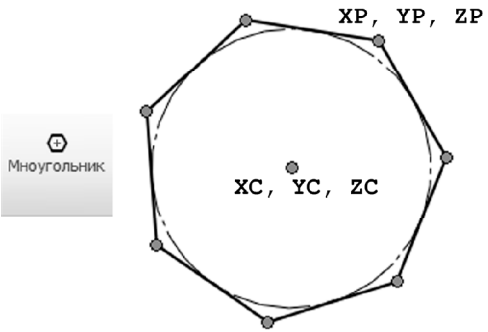


Рис. 7.9. Побудова багатокутника

Приклад

```
vSkLines = Part.SketchManager.CreatePolygon(-0.06,
-0.06, 0, -0.04, -0.07, 0, 7, True)
```

Побудова кола

Синтаксис (рис. 7.10)

```
Retval = SketchManager.CreateCircle ( XC, YC, ZC,
XP, YP, ZP)
```

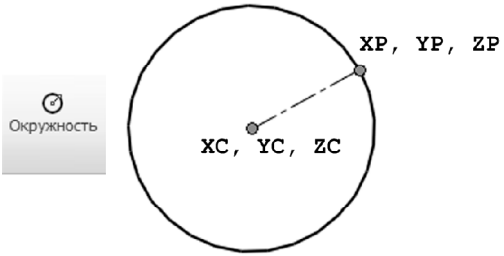


Рис. 7.10. Побудова кола

Приклад

```
Set SkCircle = Part.SketchManager.CreateCircle ( 0.01,
0.07, 0, 0.02, 0.06, 0)
```

Побудова кола за 3–ма точками

Синтаксис (рис. 7.11)

```
result = SketchManager.PerimeterCircle ( x1, y1,
x2, y2, x3, y3)
```

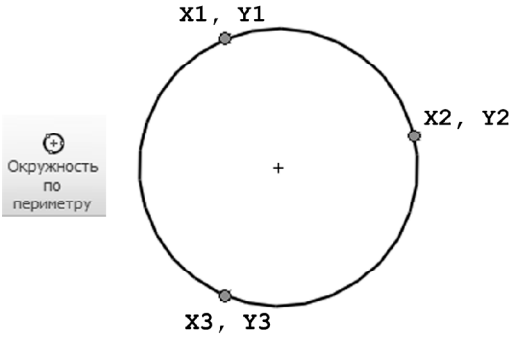


Рис. 7.11. Побудова кола за 3–ма точками

Приклад

```
Set SkCircle = Part.SketchManager.PerimeterCircle(
-0.1, -3.6E-04, -0.08, -0.02, -0.08, 0.01)
```

Побудова дуги

Синтаксис (рис. 7.12)

```
Retval = SketchManager.CreateArc ( XC, YC, ZC,
X1, Y1, Z1, X2, Y2, Z2, Напрям)
```

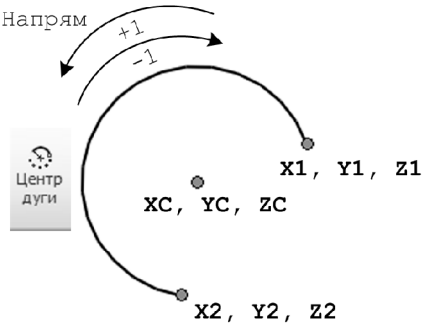


Рис. 7.12. Побудова дуги

Приклад

```
Set SkArc = Part.SketchManager.CreateArc(-0.01, 0.04,
0, -0.005, 0.05, 0, 1.4E-04, 0.03, 0, -1)
```

Побудова дотичної дуги

Синтаксис (рис. 7.13)

```
Retval = SketchManager.CreateTangentArc ( X1, Y1,
Z1, X2, Y2, Z2, ArcType)
```

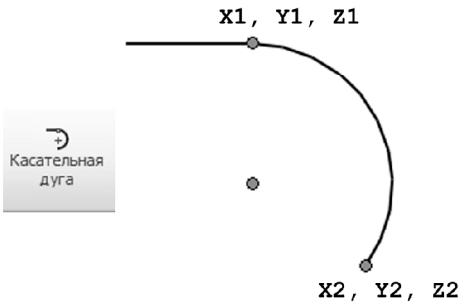


Рис. 7.13. Побудова дотичної дуги

де:

ArcType – завдання типу дуги, визначається системною змінною **swTangentArcTypes_e [10].**

Перед побудовою потрібно обрати точку на іншому об'єкті.

Приклад

```
Set SkArc = Part.SketchManager.CreateTangentArc(–0.03,
0.08, 0, –0.01, 0.07, 0, 1)
```

Побудова дуги за 3–ма точками

Синтаксис (рис. 7.14)

```
Retval = SketchManager.Create3PointArc ( X1, Y1, Z1,
X2, Y2, Z2, X3, Y3, Z3)
```

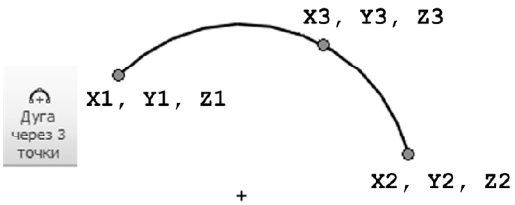


Рис. 7.14. Побудова дуги за 3–ма точками

Приклад

```
Set SkArc = Part.SketchManager.Create3PointArc(0.002,
–0.08, 0, 0.02, –0.06, 0, 0.002, –0.07, 0)
```

Побудова сплайна

Синтаксис (рис. 7.15)

```
Retval = SketchManager.CreateSpline ( PointData )
```

де:

PointData – одновимірний масив координат опорних точок **X**, **Y**, **Z**, що використовуються для побудови сплайна.

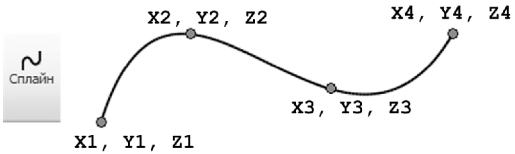


Рис. 7.15. Побудова сплайна

Приклад

```
Dim vPoints As Variant,
Dim pA(0 To 12) As Double
                              'оголошення масиву для 4-х точок
pA(0) = 0.021: pA(1) = 0.002: pA(2) = 0
pA(3) = 0.051: pA(4) = -0.042: pA(5) = 0
pA(6) = 0.085: pA(7) = -0.006: pA(8) = 0
pA(9) = 0.096: pA(10) = 0.030: pA(11) = 0
                              'координати точок
vPoints = pA
Dim SkSpline As Object
Set SkSpline = Part.SketchManager.CreateSpline(vPoints)
```

Побудова еліпсу

Синтаксис (рис. 7.16)

```
Retval = SketchManager.CreateEllipse ( XC, YC, ZC,
XMajor, YMajor, ZMajor, XMinor, YMinor, ZMinor)
```

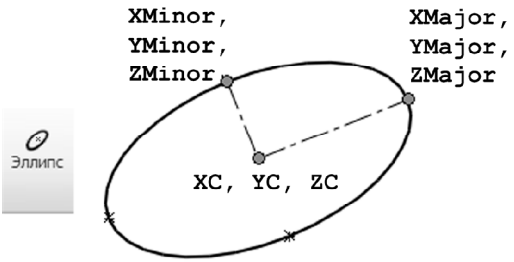


Рис. 7.16. Побудова еліпсу

Приклад

```
Set SkEllipse = Part.SketchManager.CreateEllipse(0.1,
0.01, 0, 0.1, 0.02, 0, 0.1, 0.001, 0)
```

Побудова точки

Синтаксис (рис. 7.17)

```
Retval = SketchManager.CreatePoint ( X, Y, Z)
```

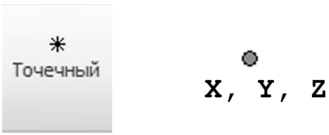


Рис. 7.17. Побудова точки

Приклад

```
Set SkPoint = Part.SketchManager.CreatePoint(-0.04,
0.03, 0)
```

Слід зазначити, що усі розглянуті методи використовуються для побудови як 2D, так і 3D ескізів та креслень.

Для перетворення побудованих об'єктів ескізу у довідкову геометрію (перетворення основних ліній в осьові) використовується метод **CreateConstructionGeometry**.

Синтаксис

```
Part.SketchManager.CreateConstructionGeometry
```

Метод не має аргументів.

Приклад

```
boolstatus    =    Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір лінії №4
Part.SketchManager.CreateConstructionGeometry
                          'переведення обраної лінії у
                          'довідкову геометрію
```

7.3. Взаємозв'язки

Послідовність встановлення взаємозв'язків між елементами ескізу наведено на рис. 7.18.

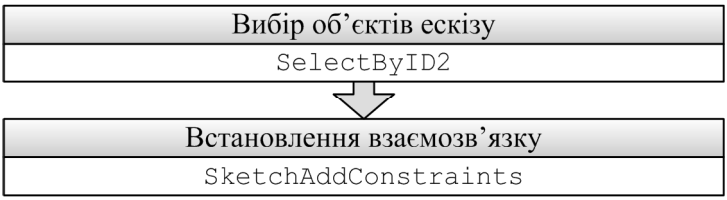


Рис. 7.18. Послідовність встановлення взаємозв'язку

Для встановлення взаємозв'язків використовується метод **SketchAddConstraints**.

Синтаксис

Part.SketchAddConstraints "назва взаємозв'язку"

де:

назва взаємозв'язку – взаємозв'язок, що встановлюється.

Наведемо назви деяких взаємозв'язків: **sgHORIZONTAL2D** – горизонтально 2D; **sgVERTICAL2D** – вертикально 2D; **sgCOLINEAR** – колінеарно; **sgCORADIAL** – корадиально; **sgPERPENDICULAR** – перпендикулярно; **sgPARALLEL** – паралельно; **sgTANGENT** – дотично; **sgCONCENTRIC** – концентрично; **sgSYMMETRIC** – симетрично; **sgFIXED** – зафіксовано.

Приклади

```
boolstatus = Part.Extension.SelectByID2("Line4",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір лінії №4
Part.SketchAddConstraints "sgHORIZONTAL2D"
                          'взаємозв'язок Горизонтально
boolstatus = Part.Extension.SelectByID2("Line3",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір лінії №3
Part.SketchAddConstraints "sgVERTICAL2D"
                          'взаємозв'язок Вертикально
```

7.4. Встановлення розмірів

Послідовність встановлення розмірів між елементами ескізу наведено на рис. 7.19.

Для увімкнення/вимкнення режиму автоматичного відкриття вікна зміни значення розміру (рис. 7.20) застосовується універсальний метод **SetUserPreferenceToggl**e, який уключає/вимикає різноманітні налаштування у системі.

Синтаксис

retval=ModelDoc2.SetUserPreferenceToggle(uPV,oF)

де:

uPV – параметр, що визначається системною змінною **swUserPreferenceToggle_e** [10]. Для увімкнення/вимкнення вікна зміни значення розміру значення змінної повинно дорівнювати 10.

oF – логічна змінна; `True` – увімкнення параметру, `False` – вимкнення параметру.

Рис. 7.19. Послідовність встановлення розміру

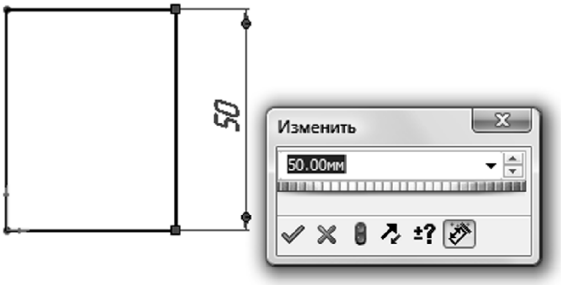


Рис. 7.20. Вікно зміни розміру

Приклад

```
Boolstatus = swApp.SetUserPreferenceToggle ( 10,
False)
                                'вимкнення режиму автоматичного
                                'відкриття вікна зміни значення
                                'розміру
```

Примітка. Не забувайте після усіх геометричних побудов вмикати режим автоматичного відкриття вікна зміни значення розміру.

Для розміщення тексту розміру використовується метод **AddDimension2**.

Синтаксис:

```
retval = ModelDoc2.AddDimension2 ( x, y, z )
```

де:

x, **y**, **z** – координати розміщення тексту розміру.

Для створення розміру використовується метод **Parameter**.

Синтаксис

```
retval = ModelDoc2.Parameter (sI)
```

де:

sI – повна назва розміру.

Для встановлення значення розміру використовується метод **SystemValue**.

Синтаксис

```
myDimension.SystemValue = значення
```

де:

значення – значення розміру у метрах.

Приклад

```
Boolstatus = swApp.SetUserPreferenceToggle ( 10,
False)
                          'вимкнення режиму автоматичного
                          'відкриття вікна зміни значення
                          'розміру
Boolstatus = Part.Extension.SelectByID2( "Line4", "SKE
TCHSEGMENT",0,0,0,False,0,n,0)
                          'обрати першу лінію №4
Boolstatus = Part.Extension.SelectByID2( "Line6", "SKE
TCHSEGMENT",0,0,0,True,0,n,0)
                          'обрати другу лінію №6
SetSkPoint = Part.AddDimension2(0.01, 0.07, 0)
                          'розмістити розмір
myDimension = Part.Parameter ("D1@Эскиз1")
                          'створити розмір D1@Эскиз1
myDimension.SystemValue = 0.07
                          'значення розміру 70 мм
Boolstatus = swApp.SetUserPreferenceToggle ( 10,
False)
                          'вимкнення режиму автоматичного
                          'відкриття вікна зміни значення
                          'розміру
```

7.5. Інструмент ескізу Скругление

Послідовність створення скруглення наведено на рис. 7.21.

Рис. 7.21. Послідовність створення скруглення

Синтаксис

```
Retval = SketchManager.CreateFillet ( R, CC)
```

Аргументи методу **CreateFillet** наведено у табл. 7.1.

Таблиця 7.1

Аргументи методу CreateFillet

Аргумент	Тип	Призначення
R	double	Радіус скруглення у метрах.
CC	integer	Режим побудови скруглення при обмеженнях взаємозв'язками або розмірами кута для скруглення: 0 – запитати користувача, чи слід видаляти геометрію або зупинити обробку; 1 – залишити взаємозв'язки або розмір і побудувати скруглення; 2 – видалити будь-які обмеження і побудувати скруглення; 3 – не видаляти обмежень і не будувати скруглень. За відсутності обмежень, цей аргумент ігнорується.

Скруглення можна побудувати між відрізками, які не перетинаються, але найменша відстань між ними менша за радіус.

Приклад

Створення скруглення на основі точки перетину двох ліній.

```
boolstatus = Part.Extension.SelectByID2("Point11",
"SKETCHPOINT", 0, 0, 0, False, 1, Nothing, 0)
                              'обрати точку перетину ліній
Set SkArc = Part.SketchManager.CreateFillet(0.005, 1)
                              'побудувати скруглення радіусом
                              '5 мм
```

Створення скруглення на основі двох відрізків (що можуть і не перетинатися).

```
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                          'обрати першу лінію №1
boolstatus = Part.Extension.SelectByID2("Line2",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
                          'обрати другу лінію №2
Set SkArc = Part.SketchManager.CreateFillet(0.01, 1)
                          'побудувати скруглення радіусом
                          '10 мм
```

7.6. Інструмент ескізу Фаска

Послідовність побудови фаски наведено на рис. 7.22.

Рис. 7.22. Послідовність побудови фаски

Синтаксис

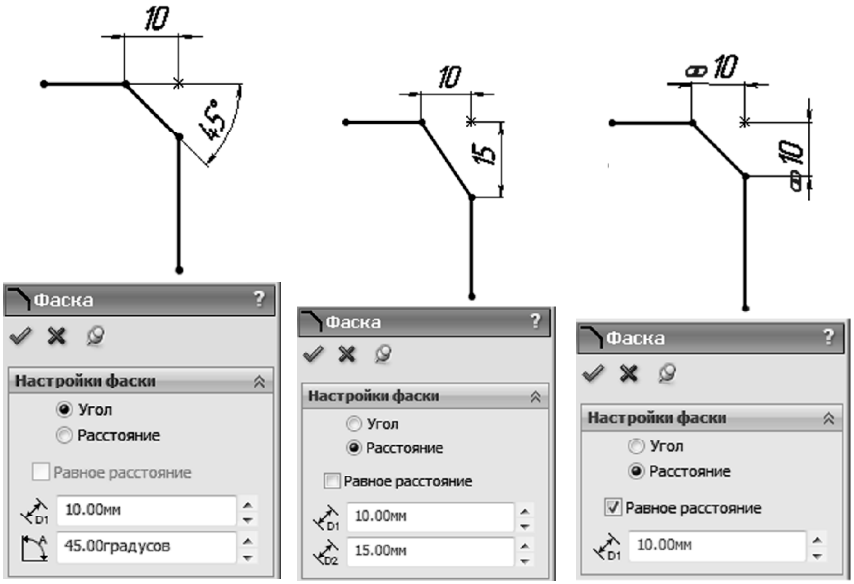
```
Retval = SketchManager.CreateChamfer(T,AOD,D)
```

Аргументи методу **`CreateChamfer`** наведено у табл. 7.2.

Таблиця 7.2

Аргументи методу `CreateChamfer`

Аргумент	Тип	Призначення
`T`	`long`	Режим побудови фаски: 0 – за відстанню та кутом (рис 7.23, *а*); 1 – за двома різними відстанями (рис 7.23, *б*); 2 – за двома однаковими відстанями(рис 7.23, *в*).
`AOD`	`double`	Якщо аргумент **`T`** = 0, то цей аргумент є кутом. Якщо аргумент **`T`** = 1, то цей аргумент є відстанню. Якщо аргумент **`T`** = 2, то цей аргумент ігнорується.
`D`	`double`	Розмір фаски у метрах.



а *б* *в*

Рис. 7.23. Побудова фаски: *а* – за розміром і кутом; *б* – за двома різними розмірами; *в* – за двома однаковими розмірами

Приклад

```
boolstatus = Part.Extension.SelectByID2("Point3",
"SKETCHPOINT", 0.002, 0.02, 0, False, 1, Nothing, 0)
                          'обрати точку перетину ліній
Dim SkLine As Object
Set SkLine = Part.SketchManager.CreateChamfer(2, 0.01, 0.01)
                          'створити фаску за двома
                          'однаковими розмірами 10 мм
```

Фаску, так само як і скруглення, можна побудувати між відрізками, які не перетинаються, але найменша відстань між ними повинна бути менша за розмір фаски.

7.7. Масиви

У контексті ескізу доступні три інструменти побудови масивів: **Зеркальное отражение**, **Линейный масив** та **Круговой масив**.

7.7.1. Дзеркальне відображення

Послідовність побудови дзеркального відображення наведено на рис 7.24.

Рис. 7.24. Послідовність побудови дзеркального відображення

Синтаксис

```
void ModelDoc2.SketchMirror ()
```

Метод не має аргументів.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Arc3",
"SKETCHSEGMENT", 0.007, 0.02, 0, True, 0, Nothing, 0)
                          'вибір об'єкту для
                          'дзеркального відображення
boolstatus = Part.Extension.SelectByID2("Line18",
"SKETCHSEGMENT", 0.03, 0.01, 0, True, 0, Nothing, 0)
                          'вибір лінії дзеркального
                          'відображення
Part.SketchMirror         'побудова відображення
```

7.7.2. Лінійний масив

Послідовність побудови лінійного масиву елементів ескізу наведено на рис 7.25.

Синтаксис

```
retval =
ModelDoc2.CreateLinearSketchStepAndRepeat (n1,
n2, s1, s2, a1, a2, dI )
```

Рис. 7.25. Послідовність побудови лінійного масиву

Аргументи методу **CreateLinearSketchStepAndRepeat** наведено у табл. 7.3.

Таблиця 7.3

Аргументи методу побудови лінійного масиву

Аргумент	Тип	Призначення
n1	long	Кількість елементів масиву у напрямку 1, включно з базовим.
n2	long	Кількість елементів масиву у напрямку 2, включно з базовим.
s1	double	Відстань між елементами вздовж напрямку 1.
s2	double	Відстань між елементами вздовж напрямку 2.
a1	double	Кут для напрямку 1 по відношенню до осі **X**.
a2	double	Кут для напрямку 2 по відношенню до осі **X**.
dI	string	Вказує на елементи масиву, які потрібно пропустити у вигляді рядка у форматі: "(a) (б) (c)"

Приклад

```
boolstatus = Part.Extension.SelectByID2("Line9",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
                          'вибір об'єкту для лінійного
                          'масиву
Part.CreateLinearSketchStepAndRepeat 6, 4, 0.01, 0.01,
0, 1.5, "(3,2) (3,3)"
                          'побудова лінійного масиву
                          'розміром 6x4 з пропуском двох
                          'екземплярів
```

7.7.3. Круговий масив

Послідовність побудови лінійного масиву елементів ескізу наведено на рис 7.26.

Рис. 7.26. Послідовність побудови кругового масиву

Синтаксис

```
retval = ModelDoc2.CreateCircularSketchStepAndRepeat (
aR, aA, pN, pS, pR, dI )
```

Аргументи методу **CreateLinearSketchStepAndRepeat** наведено у табл. 7.4.

Таблиця 7.4

Аргументи методу побудови кругового масиву

Аргумент	Тип	Призначення
aR	double	Радіус обертання масиву.
aA	double	Кут обертання масиву.
pN	double	Кількість елементів масиву, включно з базовим.
pS	double	Відстань між елементами масиву у радіанах.
pR	boolean	Напрям обертання: True – за годинниковою стрілкою, False – проти годинникової стрілки.
dI	string	Вказує на елементи масиву, які потрібно пропустити у вигляді рядка у форматі: "(а) (б) (с)".

Приклад

```
boolstatus = Part.Extension.SelectByID2("Arc1",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір об'єктів для кругового
                          'масиву
boolstatus = Part.SketchManager.
CreateCircularSketchStepAndRepeat(0.0693, 5.54, 4,
1.57, True, "", False, False, True)
                          'побудова кругового масиву
```

7.8. Приклад розроблення проектної процедури

Розглянемо проектну об'єктну підсистему для автоматизованої побудови ескізів прокладок.

Вихідні дані, що задаються користувачем, – розміри прокладки та кількість отворів під кріпильні вироби.

Проектування форми користувача. На формі користувача розташовуються елементи управляння згідно з рис. 7.27.

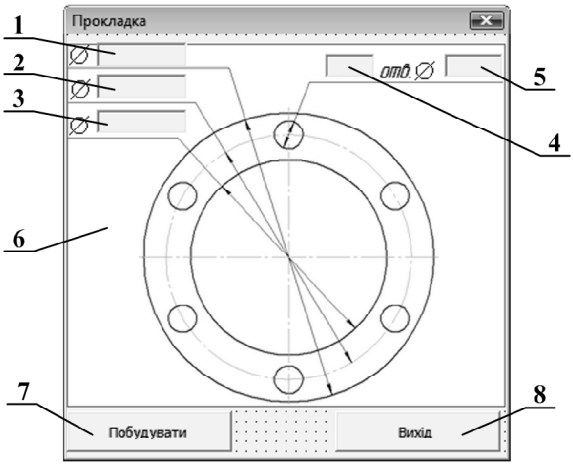


Рис. 7.27. Форма користувача та елементи управління: *1–5* – поля для введення **TextBox1**–**TextBox5**; *6* – рисунок **Image1**; *7* – кнопка **CommandButton1**; *8* – кнопка **CommandButton2**

При розробленні форми користувача у менеджері властивостей установлюються такі параметри:

– для форми **UserForm1**:

```
UserForm1.Caption = "Прокладка"
```

– для кнопки **CommandButton1**:

```
CommandButton1.Caption = "Побудувати";
```

обробник події **CommandButton1_Click()**;

– для кнопки CommandButton2:

```
CommandButton2.Caption = "Вихід";
```

обробник події **CommandButton2_Click()**.

Проектна підсистема містить головну програму та 2 обробники подій елементів управління:

– обробник події натиснення кнопки **CommandButton1** для побудови ескізу прокладки;

– обробник події натиснення кнопки **CommandButton2** для завершення роботи.

Головна програма

```
Sub main()
  UserForm1.Show          'запуск форми користувача
End Sub
```

Обробник події натиснення кнопки **CommandButton1** для побудови ескізу прокладки.

```
Private Sub CommandButton1_Click()
Dim swApp As Object, Part As Object
Dim longstatus As Long, longwarnings As Long
Dim skSegment As Object, boolstatus As Boolean
Dim myDisplayDim As Object, n As Integer
                          'оголошення змінних
D1 = Val(TextBox1.Text) / 1000
D2 = Val(TextBox2.Text) / 1000
D3 = Val(TextBox3.Text) / 1000
n = Val(TextBox4.Text)
d = Val(TextBox4.Text) / 1000
                          'завантаження значень розмірів з
                          'елементів управління у змінні
If (D1>0) And (D2>0) And (D3>0) And (n>2) And (d>0)
And (D1>D2) And (D2>D3) And (D1-D3>d) Then
                          'перевірка коректності введення
                          'даних. Якщо значення коректні,
                          'то...
Set swApp = Application.SldWorks
                          'активація SolidWorks
Set Part = swApp.NewDocument("C:\ProgramData\SolidWorks\
SolidWorks 2011\templates\Деталь.prtdot", 0, 0, 0)
                          'створення документа деталі з
                          'шаблону Деталь.prtdot
Set Part = swApp.ActiveDoc
                          'активація документа
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір площини Спереди для
                          'створення ескізу
Part.SketchManager.InsertSketch True
                          'запуск режиму побудови ескізу
Set skSegment = Part.SketchManager.CreateCircle(0#,
0#, 0#, -0.084039, 0.05233, 0#)
Set skSegment = Part.SketchManager.CreateCircle(0#,
0#, 0#, -0.072367, 0.040658, 0#)
Set skSegment = Part.SketchManager.CreateCircle(0#,
0#, 0#, -0.047855, 0.04377, 0#)
```

```
Set skSegment = Part.SketchManager.CreateCircle(0#,
0.083006, 0#, 0.004504, 0.073473, 0#)
                        'побудова кіл ескізу прокладки
boolstatus= swApp.SetUserPreferenceToggle ( 10, False)
                        'вимкнення підтвердження
                        'значення розміру
boolstatus = Part.Extension.SelectByID2("Arc2",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір другого кола ескізу
Part.SketchManager.CreateConstructionGeometry
                        'переведення до допоміжної
                        'геометрії (осьова лінія)
boolstatus = Part.Extension.SelectByID2("Arc1",
"SKETCHSEGMENT", 0, 0, 0, False,0, Nothing,0)
                        'вибір першого кола
Set myDisplayDim = Part.AddDimension2(-0.117, 0.119, 0)
                        'розміщення розміру
Set myDimension = Part.Parameter("D1@Эскиз1")
                        'створити розмір D1@Эскиз1
myDimension.SystemValue = D1
                        'встановити значення розміру
boolstatus = Part.Extension.SelectByID2("Arc2",
"SKETCHSEGMENT", 0, 0, 0, False, 0,Nothing, 0)
                        'вибір другого кола
Set myDisplayDim = Part.AddDimension2(-0.125, 0.095, 0)
                        'розмістити розмір
Set myDimension = Part.Parameter("D2@Эскиз1")
                        'створити розмір D2@Эскиз1
myDimension.SystemValue = D2
                        'встановити значення розміру
boolstatus = Part.Extension.SelectByID2("Arc3",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing,0)
                        'вибір третього кола
Set myDisplayDim = Part.AddDimension2(-0.121, 0.067, 0)
                        'розмістити розмір
Set myDimension = Part.Parameter("D3@Эскиз1")
                        'створити розмір D3@Эскиз1
myDimension.SystemValue = D3
                        'встановити значення розміру
boolstatus = Part.Extension.SelectByID2("Arc4",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір четвертого кола
Set myDisplayDim = Part.AddDimension2(0.037, 0.117, 0)
                        'розмістити розмір
Set myDimension = Part.Parameter("D4@Эскиз1")
                        'створити розмір D4@Эскиз1
myDimension.SystemValue = d
                        'встановити значення розміру
```

```
Part.ClearSelection2 True
                        'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Arc4",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір четвертого кола для
                        'побудови кругового масиву
boolstatus = Part.SketchManager.CreateCircularSk
etchStepAndRepeat(D2 / 2, 4.712388980385, n, 2 *
3.14159265358979 / n, True, "", False, False, True)
                        'побудова кругового масиву з
                        'кількістю елементів n
boolstatus = Part.Extension.SelectByID2("Point9",
"SKETCHPOINT", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір центральної точки другої
                        'копії кола у масиві
boolstatus = Part.Extension.SelectByID2("Arc2",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
                        'вибір другого кола
Part.SketchAddConstraints "sgCOINCIDENT"
                        'встановлення взаємозв'язку
                        'Совпадение
Part.ClearSelection2 True
                        'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Point11",
"SKETCHPOINT", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір центральної точки
                        'третьої копії кола у масиві
boolstatus = Part.Extension.SelectByID2("Arc2",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
                        'вибір другого кола
Part.SketchAddConstraints "sgCOINCIDENT"
                        'встановлення взаємозв'язку
                        'Совпадение
boolstatus= swApp.SetUserPreferenceToggle (10, True)
                        'включення підтвердження
                        'значення розміру
Part.SketchManager.InsertSketch True
                        'вихід з режиму побудови ескізу
                        'зі збереженням змін
Else MsgBox "Не вірно введені початкові дані", 48, "Помилка"
                        'якщо початкові дані введені не
                        'вірно, відображається вікно
                        'повідомлення про помилку
                        '(рис. 7.2).
End Sub                 'завершення процедури
```

Обробник події натиснення кнопки **CommandButton2** завершення роботи макросу

```
Private Sub CommandButton2_Click()
    Unload UserForm1      'закриття форми користувача
End Sub                   'завершення процедури
```

Приклад роботи проектної процедури наведено на рис. 7.28.

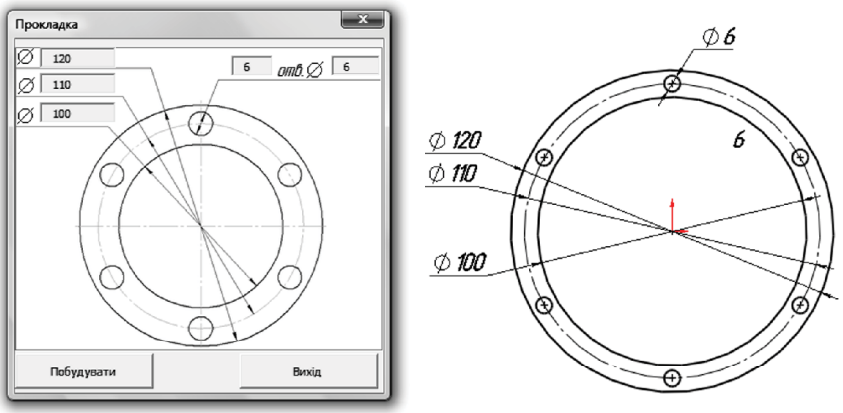


Рис. 7.28. Приклад роботи проектної процедури

Контрольні запитання і завдання

1. Назвіть послідовність дій та методи, що використовуються для створення ескізу.
2. Назвіть методи для побудови об'єктів ескізу.
3. Який метод використовуються для побудови взаємозв'язків?
4. Назвіть усі можливі взаємозв'якзи у ескізі.
5. Яка послідовність дій та які методи використовуються для створення розмірів?
6. Назвіть особливості використання методу для увімкнення/вимкнення режиму автоматичного відкриття вікна зміни значення розміру.
7. Яка послідовність дій та які методи використовуються для побудови скруглення?

8. Назвіть послідовність дій та методи, що використовуються для побудови фаски.
9. Яка послідовність дій та які методи використовуються для побудови дзеркального відображення?
10. Назвіть послідовність дій та методи, що використовуються для побудови лінійного масиву.
11. Яка послідовність дій та які методи використовуються для побудови кругового масиву?
12. Побудуйте блок-схему проектної процедури побудови ескізів прокладок.
13. Створіть проектну процедуру побудови ескізу прямокутника зі скругленими кутами.
14. Створіть проектну процедуру побудови ескізу прямокутника з фасками по кутах.
15. Створіть проектну процедуру побудови ескізу прямокутної решітки прямокутними отворами.
16. Створіть проектну процедуру побудови ескізу косинки з п. 6.7

8. Основні методи побудови твердотільних моделей

Основні інструменти побудови твердотільних моделей можна поділити на дві групи:

а) інструменти додавання матеріалу:

- **Вытянутая бобышка**;
- **Повёрнутая бобышка**;
- **По траектории**;
- **По сечениям**.

б) інструменти для видалення матеріалу:

- **Вытянутый выре**з;
- **Повёрнутый вырез**;
- **Вырез по траектории**;
- **Вырез по сечениям**.

8.1. Методи додавання матеріалу

8.1.1. Метод інструменту Вытянутая бобышка

Послідовність використання інструменту **Вытянутая бобышка** наведено на рис. 8.1.

Рис. 8.1. Послідовність використання інструменту **Вытянутая бобышка**

Синтаксис:

```
pFeat = FeatureManager.FeatureExtrusion2 ( sd,
flip, dir, t1, t2, d1, d2, dchk1, dchk2, ddir1,
ddir2, dang1, dang2, oR1, oR2, tS1, tS2, m, uFS,
uAS, t0, sO, fSO )
```

Аргументи методу **`FeatureExtrusion2`** наведено у табл. 8.1.

Таблиця 8.1

Аргументи методу `FeatureExtrusion2`

Аргумент	Тип	Призначення
1	2	3
sd	boolean	True – витягування в одну сторону; False – витягування у дві сторони.
flip	boolean	Аргумент використовується тільки для витягнутого вирізу. Видаляє весь матеріал із зовнішньої сторони ескізу. За замовчуванням матеріал видаляється всередині замкненого ескізу. True – змінити сторону видалення матеріалу; False – не змінювати напрям.
dir	boolean	True – змінити напрям витягування; False – не змінювати напрям.
t1	integer	Кінцева умова витягування ескізу за напрямом 1 визначається системною змінною **swEndConditions_e** [10]: 0 – на задану відстань; 1 – крізь усе; 2 – до наступного; 3 – до вершини; 4 – до поверхні; 5 – зі зміщенням від поверхні; 6 – середня площина; 7 – до тіла.
t2	integer	Кінцева умова витягування ескізу за напрямом 2 визначається таким же чином, як аргумент **t1**.
d1	double	Довжина витягування за напрямом 1 у метрах.
d2	double	Довжина витягування за напрямом 2 у метрах.
dchk1	boolean	True – витягування з ухилом за напрямом 1; False – без ухилу.
dchk2	boolean	True – витягування з ухилом за напрямом 2; False – без ухилу.
ddir1	boolean	True – ухил усередину за напрямом 1; False – ухил назовні.
ddir2	boolean	True – ухил усередину за напрямом 2; False – ухил назовні.
dang1	double	Величина ухилу за напрямом 1.
dang2	double	Величина ухилу за напрямом 2.
oR1	boolean	Якщо обрано зміщення за напрямом 1 від поверхні або площини, то True – зміщення від площі ескізу; False – зміщення від поверхні або площини.
oR2	boolean	Якщо обрано зміщення за напрямом 2 від поверхні або площини, то True – зміщення від площі ескізу; False – зміщення від поверхні або площини.

Продовження табл. 8.1

1	2	3
tS1	boolean	Якщо обрано кінцеву умову витягування зі зміщенням від поверхні за напрямом 1, то True – кінець витягування зміщений відносно опорної поверхні; False – задавання реального зміщення.
tS2	boolean	Якщо обрано кінцеву умову витягування зі зміщенням від поверхні за напрямом 2, то True – кінець витягування зміщений відносно опорної поверхні; False – задавання реального зміщення.
m	boolean	True – об'єднати результат побудови у єдине тіло; False – окремі тіла.
uFS	boolean	True – елемент впливає тільки на визначені тіла; False – елемент впливає на всі тіла.
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент.
t0	integer	Те, від чого витягувати ескіз, визначається системною змінною **swStartConditions_e** [10]: 0 – від площини ескізу; 1 – від поверхні; 2 – від вершини; 3 – зі зміщенням.
s0	double	Якщо витягування ескізу починається зі зміщенням, тоді тут записується значення зміщення.
fS0	boolean	Якщо витягування ескізу починається зі зміщенням, тоді True – змінити напрям зміщення; False – не змінювати напрям зміщення.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                              'вибір ескізу для витягування
Part.FeatureManager.FeatureExtrusion2 ( False, False,
False, 0, 0, 0.01, 0.01, False, False, False, False,
0.01, 0.01, False, False, False, False, 1, 1, 1, 0, 0,
False )                       'витягування ескізу у дві
                              'сторони на 10 мм
```

8.1.2. Метод інструменту Повернутая бобышка

Послідовність використання інструменту **Повёрнутая бобышка** наведено на рис. 8.2.

Рис. 8.2. Послідовність використання інструменту **Повёрнутая бобышка**

Синтаксис

```
retVal = FeatureManager.FeatureRevolve ( a, rD,
a2, rT, o, m, uFS, uASl )
```

Аргументи методу **`FeatureRevolve`** наведено у табл. 8.2.

Таблиця 8.2

Аргументи методу `FeatureRevolve`

Аргумент	Тип	Призначення
1	2	3
a	double	Кут обертання ескізу в радіанах у напряму 1.
rD	boolean	True – змінити напрям обертання ескізу; False – не змінювати напрям.
a2	double	Кут обертання ескізу в радіанах у напряму 2.
rT	long	Тип обертання визначається системною змінною **swRevolveType_e** [10]: 0 – обертання в одному напряму; 1 – обертання у двох напрямках симетрично до площини ескізу; 2 – обертання у двох напрямах; 3 – обертання у одному нарямку на кут 360^0 з можливістю редагування; 4 – обертання у двох напрямках симетрично до площини ескізу на кут 360^0 з можливістю редагування; 5 – обертання у двох напрямках з можливістю редагування.
o	long	Додаткові опції контролю, що визначаються системною змінною **swRevolveOptions_e** [10]: 0 – без змін; 1 – автоматично закрити ескіз.

Продовження табл. 8.2

1	2	3
m	boolean	True – об'єднати результат побудови у єдине тіло; False – окремі тіла.
uFS	boolean	True – елемент впливає тільки на визначені тіла; False – елемент впливає на усі тіла.
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
                          'вибір ескізу для обертання
boolstatus = Part.Extension.SelectByID2("Line3@Эскиз1",
"EXTSKETCHSEGMENT", 0, 0, 0, False, 16, Nothing, 0)
                          'вибір прямого відрізка в
                          'ескізі для осі обертання,
                          'причому аргумент m = 16.
Dim myFeature As Object
Set myFeature = Part.FeatureManager.FeatureRevolve(6.2,
0, 0, 0, 1, 1, 1)         'побудова тіла обертання на
                          'основі ескізу
```

8.1.3. Метод інструменту Вытянутая бобышка по траектории

Послідовність використання інструменту **Вытянутая бобышка по траектории** наведено на рис. 8.3.

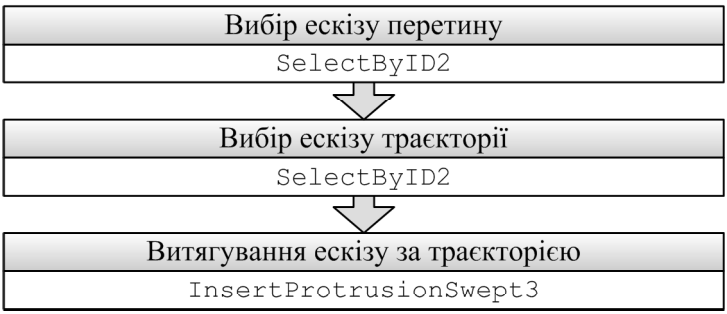


Рис. 8.3. Послідовність витягування ескізу за траєкторією

Синтаксис

```
retval = FeatureManager.InsertProtrusionSwept3 (
p, a, tCO, kT, bAS, sMT, eMT, iTB, t1, t2, tT, pA,
m, uFS, uAS, tA, bMSF)
```

Аргументи методу **InsertProtrusionSwept3** наведено у табл. 8.3.

Таблиця 8.3

Аргументи методу `InsertProtrusionSwept3`

Аргумент	Тип	Призначення
1	2	3
p	boolean	True – поширити витягування ескізу до наступної дотичної кромки; False – не поширювати.
a	boolean	True – витягування ескізу крізь кінець грані, якщо траєкторія, що використовується для витягування, проходить від однієї грані до іншої або від однієї кромки до іншої; False – витягування ескізу починається і завершується перпендикулярно до траєкторії витягування і не може проходити крізь дві кінцеві грані тіла.
tCO	short	Опції кручення визначаються системною змінною **swTwistControlType_e** [10]: 0 – скручування за напрямом; 1 – скручування за напрямом та першою направляючою кривою; 2 – скручування за напрямом та першою і другою направляючими кривими; 8 – скручування вздовж маршруту.
kT	boolean	Якщо витягуваний переріз має дотичні сегменти, то True – відповідні поверхні у результуючій моделі будуть дотичними; False – не будуть дотичними.
bAS	boolean	Якщо перетин, що витягується, є коловим або еліптичним, то True – апроксимувати перетин та згладити поверхні; False – не згладжувати поверхні.
sMT	short	Тип початку дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.

Продовження табл. 8.3

1	2	3
eMT	short	Тип кінця дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
iTB	boolean	True – будується тонке тіло; False – суцільне тіло.
t1	double	Значення товщини у першому напряму.
t2	double	Значення товщини у другому напряму.
tT	short	Тип товщини стінки, що визначається системною змінною **swThinWallType_e** [10]: 0 – в одному напряму; 1 – у протилежному напряму; 2 – від середньої площини; 3 – у двох напрямах.
pA	short	Тип шляху витягування: 0 – за замовчуванням; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
m	boolean	True – об'єднати результат побудови у єдине тіло; False – окремі тіла.
uFS	boolean	True – елемент впливає тільки на обрані тіла, False – елемент впливає на усі тіла.
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент.
tA	double	Якщо аргумент **tCO =** 8 (скручування вздовж маршруту), то тут вказується кінцевий кут повороту.
bMSF	boolean	True – об'єднати гладкі поверхні; False – не об'єднувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                              'вибір ескізу перетину
boolstatus = Part.Extension.SelectByID2("Эскиз3",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
                              'вибір ескізу траєкторії
```

```
Dim SweepFeature As Object
Set SweepFeature = Part.FeatureManager.
InsertProtrusionSwept3(False, False, 0, False, False,
0, 0, False, 0, 0, 0, 0, 1, 1, 1, 0, 1)
                          'побудова витягнутого ескізу за
                          'траєкторією
```

8.1.4. Метод інструменту Вытянутая бобышка по сечениям

Послідовність використання інструменту **Вытянутая бобышка по траектории** наведено на рис. 8.4.

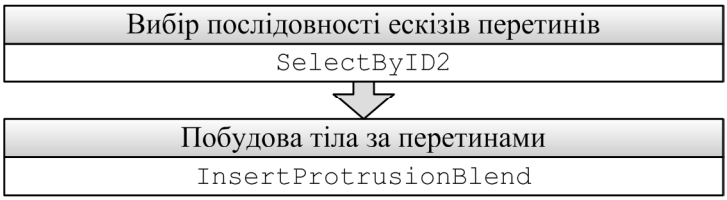


Рис. 8.4. Послідовність створення тіла за перетинами

Синтаксис

```
retval = FeatureManager.InsertProtrusionBlend (
c, kT, fNR, tTF, sMT, eMT, sTL, eTL, sTD, eTD,
iTB, t1, t2, tT, m, uFS, uAS )
```

Аргументи методу **InsertProtrusionBlend** наведено у табл. 8.4.

Таблиця 8.4

Аргументи методу InsertProtrusionBlend

Аргумент	Тип	Призначення
1	2	3
c	boolean	True – замкнути поверхню; False – залишити відкритою. Якщо встановлено True та обрано менше ніж 3 перетини, то будь-які обрані направляючі криві повинні бути замкненими.
kT	boolean	True – підтримати дотичність, як на кривих розділу; False – не підтримувати. Якщо криві перетинів дотичні, то є можливість указати, чи є також дотичними результуючі грані; при створенні дотичних поверхонь SolidWorks підтримує пласкі та циліндричні форми поверхонь, якщо криві перетинів проявляють ці характеристики.

Продовження табл. 8.4

1	2	3
fNR	boolean	True – створити більш гладкі поверхні; False – не створювати. Відповідає опції **Повышение гладкости** інтерфейсу користувача. Ця опція доступна тільки тоді, коли витягуваний переріз має циліндричні або еліптичні дуги. Перетин апроксимується та ескізи дуг можуть бути перетворені у сплайни.
tTF	double	Аргумент, який визначає кількість проміжних перетинів; значення за замовчуванням 1.0; чим більше значення, тим більше проміжних секцій створюється.
sMT	short	Тип початку дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
eMT	short	Тип кінця дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
sTL	double	Початкова довжина дотичної.
eTL	double	Кінцева довжина дотичної.
sTD	boolean	True – початок дотичності в одному напряму; False – у протилежному напряму.
eTD	boolean	True – завершення дотичності в одному напряму; False – у протилежному напряму.
iTB	boolean	True – будується тонке тіло; False – суцільне тіло.
t1	double	Значення товщини у першому напряму.
t2	double	Значення товщини у другому напряму.
tT	short	Тип товщини стінки, що визначається системною змінною **swThinWallType_e** [10]: 0 – в одному напряму; 1 – у протилежному напряму; 2 – від середньої площини; 3 – у двох напрямах.
m	boolean	True – результат злиття; False – окремі тіла.
uFS	boolean	True – елемент впливає тільки на обрані тіла; False – елемент впливає на усі тіла.
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент.

Приклад

```
    ' -- Вибір послідовності ескізів перетинів --
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз2",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз3",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз4",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
Part.FeatureManager.InsertProtrusionBlend (False,
True, False, 1, 6, 6, 1, 1, True, True, False, 0, 0,
0, True, True, True)     'побудова тіла за послідовністю
                         'перетинів
```

8.2. Методи видалення матеріалу

8.2.1. Метод інструменту Вытянутый вырез

Послідовність використання інструменту **Вытянутый вырез** наведено на рис. 8.5.

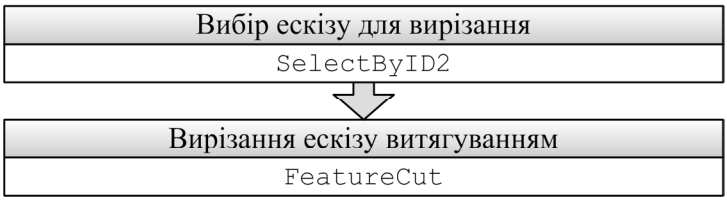


Рис. 8.5. Послідовність створення витягнутого вирізу

Синтаксис

```
pFeat = FeatureManager.FeatureCut ( sd, flip,
dir, t1, t2, d1, d2, dchk1, dchk2, ddir1, ddir2,
dang1, dang2, oR1, oR2, tS1, tS2, nC, uFS, uAS )
```

Аргументи методу **FeatureCut** наведено у табл. 8.5.

Слід зазначити, що більшість аргументів методу співпадають за назвою і призначенням із аргументами методу **FeatureExtrusion2** для побудови витягнутої бобишки (див. табл. 8.1).

Таблиця 8.5

Аргументи методу `FeatureCut`

Аргумент	Тип	Призначення
1	2	3
sd	boolean	True – витягування в одну сторону; False – витягування у дві сторони.
flip	boolean	Видаляє весь матеріал із зовнішньої сторони ескізу. За замовчуванням матеріал видаляється всередині замкненого ескізу. True – змінити сторону видалення матеріалу; False – не змінювати напрям.
dir	boolean	True – змінити напрям вирізання; False – не змінювати напрям.
t1	integer	Кінцева умова вирізання ескізу за напрямом 1 визначається системною змінною **swEndConditions_e** [10]: 0 – на задану відстань; 1 – крізь усе; 2 – до наступного; 3 – до вершини; 4 – до поверхні; 5 – зі зміщенням від поверхні; 6 – середня площина; 7 – до тіла.
t2	integer	Кінцева умова вирізання ескізу за напрямом 2 визначається таким же чином, як аргумент **t1**.
d1	double	Глибина вирізання за напрямом 1 у метрах.
d2	double	Глибина вирізання за напрямом 2 у метрах.
dchk1	boolean	True – вирізання з ухилом за напрямом 1; False – без ухилу.
dchk2	boolean	True – вирізання з ухилом за напрямом 2; False – без ухилу.
ddir1	boolean	True – ухил усередину за напрямом 1; False – ухил назовні.
ddir2	boolean	True – ухил усередину за напрямом 2; False – ухил назовні.
dang1	double	Величина ухилу за напрямом 1.
dang2	double	Величина ухилу за напрямом 2.
oR1	boolean	Якщо обрано зміщення за напрямом 1 від поверхні або площини, то True – зміщення від площі ескізу; False – зміщення від поверхні або площини.
oR2	boolean	Якщо обрано зміщення за напрямом 2 від поверхні або площини, то True – зміщення від площі ескізу; False – зміщення від поверхні або площини.

Продовження табл. 8.5

1	2	3
tS1	boolean	Якщо обрано кінцеву умову вирізання зі зміщенням від поверхні за напрямом 1, то `True` – кінець вирізання, зміщений відносно опорної поверхні; `False` – задавання реального зміщення.
tS2	boolean	Якщо обрано кінцеву умову вирізання зі зміщенням віл поверхні за напрямом 2, то `True` – кінець вирізання, зміщений відносно опорної поверхні; `False` – задавання реального зміщення.
nC	boolean	Аргумент стосується листового металу. `True` – розріз виконується нормально до товщі листового металу; `False` – ненормально або виріз на деталі з нелистового металу.
uFS	boolean	`True` – елемент впливає тільки на обрані тіла; `False` – елемент впливає на усі тіла.
uAS	boolean	`True` – усі тіла обираються автоматично та елемент впливає на них; `False` – користувач повинен сам обрати тіла, на які впливатиме елемент.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз2",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір ескізу для побудови
                          'вирізу
Part.FeatureManager.FeatureCut (True, False, False, 0,
0, 0.05, 0.05, False, False, False, False, 0.01, 0.01,
False, False, False, False, 0, 1, 1)
                          'побудова вирізу
```

8.2.2. Метод інструменту Повёрнутый вырез

Послідовність використання інструменту **Повёрнутый вырез** наведено на рис. 8.6.

Синтаксис

```
retVal = FeatureManager.FeatureRevolveCut ( a,
rD, a2, rT, o, uFS, uAS )
```

Аргументи методу **`FeatureRevolveCut`** наведено у табл. 8.6.

Слід зазначити, що більшість аргументів методу співпадають за назвою і призначенням із аргументами методу **`FeatureRevolve`** для побудови бобишки обертанням (див. табл. 8.2).

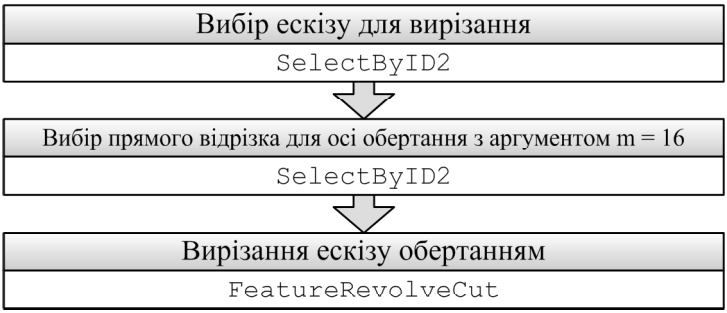


Рис. 8.6. Послідовність створення вирізу обертанням

Таблиця 8.6

Аргументи методу `FeatureRevolveCut`

Аргумент	Тип	Призначення
a	`double`	Кут обертання ескізу в радіанах у напряму 1.
rD	`boolean`	`True` – змінити напрям обертання ескізу; `False` – не змінювати напрям.
a2	`double`	Кут обертання ескізу в радіанах у напряму 2.
rT	`long`	Тип обертання визначається системною змінною **`swRevolveType_e`** [10]: 0 – обертання у одному напрямку; 1 – обертання у двох напрямках симетрично до площини ескізу; 2 – обертання у двох напрямах; 3 – обертання у одному напряму на кут 360^0 з можливістю редагування; 4 – обертання у двох напрямах симетрично до площини ескізу на кут 360^0 з можливістю редагування; 5 – обертання у двох напрямах з можливістю редагування.
o	`long`	Додаткові опції контролю, що визначаються системною змінною **`swRevolveOptions_e`** [10]: 0 – без змін; 1 – автоматично закрити ескіз.
uFS	`boolean`	`True` – елемент впливає тільки на обрані тіла; `False` – елемент впливає на усі тіла.
uAS	`boolean`	`True` – усі тіла обираються автоматично та елемент впливає на них; `False` – користувач повинен сам обрати тіла, на які впливатиме елемент.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз8",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір ескізу для обертання
boolstatus = Part.Extension.SelectByID2("Line1@
Эскиз8", "EXTSKETCHSEGMENT", 0, 0, 0, True, 16, Noth-
ing, 0)
                          'вибір прямого відрізка для
                          'осі обертання, причому m = 16
Set myFeature = Part.FeatureManager.
FeatureRevolve2(True, True, False, True, False, False,
0, 0, 6.2, 0, False, False, 0.01, 0.01, 0, 0, 0, True,
True, True)               'побудова вирізу обертанням
```

8.2.3. Метод інструменту Вырез по траектории

Послідовність використання інструменту **Вырез по траектории** наведено на рис. 8.7.

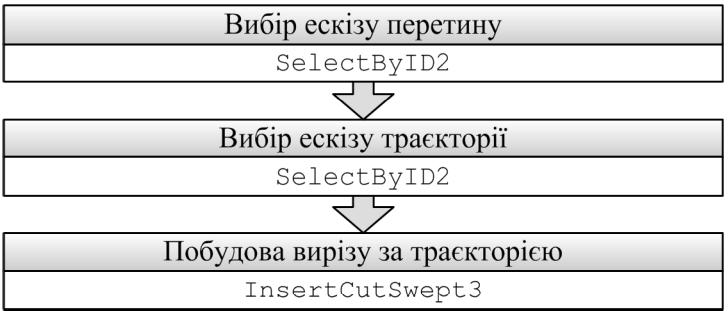


Рис. 8.7. Послідовність вирізання ескізу за траєкторією

Синтаксис

```
retval = FeatureManager.InsertCutSwept3 ( p, a,
tCO, kT, bAS, sMT, eMT, iTB, t1, t2, tT, pA, uFS,
uAS, tA, bMSF)
```

Аргументи методу **InsertCutSwept3** наведено у табл. 8.7.

Слід зазначити, що більшість аргументів методу співпадають за назвою і призначенням із аргументами методу **InsertProtrusionSwept3** для побудови витягнутої бобишки (див. табл. 8.3).

Таблиця 8.7

Аргументи методу `InsertCutSwept3`

Аргумент	Тип	Призначення
1	2	3
p	boolean	True – поширити вирізання ескізу до наступної дотичної кромки; False – не поширювати.
a	boolean	True – вирізання ескізу крізь кінець грані, якщо траєкторія, що використовується для вирізання, проходить від однієї грані до іншої або від однієї кромки до іншої; False – вирізання ескізу починається і завершується перпендикулярно до траєкторії вирізання і не може проходити крізь дві кінцеві грані тіла.
tCO	short	Опції кручення визначаються системною змінною **swTwistControlType_e** [10]: 0 – скручування за напрямом; 1 – скручування за напрямом та першою направляючою кривою; 2 – скручування за напрямом та першою і другою направляючими кривими; 8 – скручування вздовж маршруту.
kT	boolean	Якщо вирізуваний переріз має дотичні сегменти, то True – відповідні поверхні у результуючій моделі будуть дотичними; False – не будуть дотичними.
bAS	boolean	Якщо перетин, що вирізається, є коловим або еліптичним, то True – апроксимувати перетин та згладити поверхні; False – не згладжувати поверхні.
sMT	short	Тип початку дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
eMT	short	Тип кінця дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
iTB	boolean	True – будується тонке тіло; False – суцільне тіло.
t1	double	Значення товщини у першому напряму.
t2	double	Значення товщини у другому напряму.

Продовження табл. 8.7

1	2	3
tT	short	Тип товщини стінки, що визначається системною змінною **swThinWallType_e** [10]: 0 – в одному напряму; 1 – у протилежному напряму; 2 – від середньої площини; 3 – у двох напрямках.
pA	short	Тип шляху вирізання: 0 – за замовчуванням; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем/
uFS	boolean	True – елемент впливає тільки на обрані тіла, False – елемент впливає на усі тіла
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент
tA	double	Якщо аргумент **tCO =** 8 (закручування вздовж маршруту), то тут вказується кінцевий кут повороту.
bMSF	boolean	True – об'єднати гладкі поверхні; False – не об'єднувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір ескізу перетину
boolstatus = Part.Extension.SelectByID2("Эскиз3",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
                          'вибір ескізу траєкторії
Dim SweepFeature As Object
Set SweepFeature = Part.FeatureManager.
InsertCutSwept3(False, True, 0, False, False, 0, 0,
False, 0, 0, 0, 0, 1, 1, 0, 1)
                          'побудова вирізу за траєкторією
```

8.2.4. Метод інструменту Вырез по сечениям

Послідовність використання інструменту **Вырез по сечениям** наведено на рис. 8.8.

Рис. 8.8. Послідовність побудови вирізу за перетиними

Синтаксис

```
retval = FeatureManager.InsertCutBlend ( c, kT,
fNR, tTF, sMT, eMT, iTB, t, t2, tT, uFS, uAS )
```

Аргументи методу **InsertCutBlend** наведено у табл. 8.8.

Слід зазначити, що більшість аргументів методу співпадають за назвою і призначенням із аргументами методу **InsertProtrusionBlend** для побудови витягнутої бобишки за перетинами (див. табл. 8.4).

Таблиця 8.8

Аргументи методу InsertCutBlend

Аргумент	Тип	Призначення
1	2	3
c	boolean	True – замкнути поверхню; False – залишити відкритою. Якщо встановлено True та обрано менше ніж 3 перетини, то будь-які обрані направляючі криві повинні бути замкненими.
kT	boolean	True – підтримати дотичність, як на кривих розділу; False – не підтримувати. Якщо криві перетинів дотичні, то є можливість указати, чи є також дотичними результуючі грані; при створенні дотичних поверхонь SolidWorks підтримує пласкі та циліндричні форми поверхонь, якщо криві перетинів проявляють ці характеристики.
fNR	boolean	True – примусове створення нераціональної результуючої поверхні; False – створити більш раціональну поверхню.
tTF	double	Аргумент, який визначає кількість проміжних перетинів; значення за замовчуванням 1.0; чим більше значення, тим більше проміжних секцій створюється.

Продовження табл. 8.8

1	2	3
sMT	short	Тип початку дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
eMT	short	Тип кінця дотичності визначається системною змінною **swTangencyType_e** [10]: 0 – без дотичності; 1 – дотично до нормалі перетину; 2 – дотично до обраного вектора; 3 – дотично до усіх суміжних граней включно з кромками з початковим профілем.
iTB	boolean	True – будується тонке тіло; False – суцільне тіло.
t1	double	Значення товщини у першому напряму.
t2	double	Значення товщини у другому напряму.
tT	short	Тип товщини стінки, що визначається системною змінною **swThinWallType_e** [10]: 0 – у одному напряму; 1 – у протилежному напряму; 2 – від середньої площини; 3 – у двох напрямах.
uFS	boolean	True – елемент впливає тільки на обрані тіла; False – елемент впливає на усі тіла.
uAS	boolean	True – усі тіла обираються автоматично та елемент впливає на них; False – користувач повинен сам обрати тіла, на які впливатиме елемент.

Приклад

```
    ' -- Вибір послідовності ескізів перетинів --
boolstatus = Part.Extension.SelectByID2("Эскиз1",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз2",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз3",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз4",
"SKETCH", 0, 0, 0, True, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Эскиз5",
"SKETCH", 0, 0, 0, True, 1, Nothing, 0)
Part.FeatureManager.InsertCutBlend (0, 1, 0, 1, 0, 0,
0, 0, 0, 0, 1, 1)        'побудова вирізу за
                         'послідовністю перетинів
```

8.3. Методи побудови скруглень та фасок

8.3.1. Метод інструменту Скругление

Послідовність побудови скруглення наведено на рис. 8.9.

Рис. 8.9. Послідовність побудови скруглення

Синтаксис

```
retval = FeatureManager.FeatureFillet ( o, r1,
ft, oFT, radii, sBD, pRA)
```

Аргументи методу **FeatureFilletd** наведено у табл. 8.9.

Примітка. Особливістю вибору кромки або грані твердого тіла методом **SelectByID2** є те, що вони не мають власних імен, тому слід застосовувати вибір указанням точки на цій грані або кромці.

Таблиця 8.9

Аргументи методу **FeatureFillet**

Аргумент	Тип	Призначення
1	2	3
r1	double	Значення радіуса при постійному скругленні. Використовується тільки тоді, коли аргумент **o =** 2, а аргумент **ft** не дорівнює 1.
ft	long	Тип скруглення визначається системною змінною **swFeatureFilletType_e** [10]: 0 – просте скруглення; 1 – скруглення зі змінним радіусом; 2 – скруглити грані; 3 – повне скруглення.
radii	Variant	Масив, що містить змінні радіуси скруглень.
sBD	Variant	Масив, що містить відстані між опорними точками скруглення за кромкою.
pRA	Variant	Масив, що містить значення радіусів опорних точок протягом довжини кромки і використовується при змінному радіусі скруглення.

Продовження табл. 8.9

1	2	3
o	long	Опції скруглення визначаються системною змінною **swFeatureFilletOptions_**e [10]: 1 – розповсюдити скруглення; 2 – постійний радіус скруглення; 4 – прямий перехід, який не використовується для плавного переходу; 8 – скруглення з використанням допоміжних точок; 16 – скруглення за допомогою дотичної опорної лінії; 32 – скруглення кутів; 64 – скруглення пов'язаних кромок 128 – скруглення зі збереженням елементів; 256 – скруглення з постійним згином; 512 – скруглення з постійною шириною; 1024 – прив'язане скруглення; 2048 – зміна напряму скруглення першої грані; 4096 – зміна напряму скруглення другої грані; 8192 – розповсюдження по деталях.
oFT	long	Контроль набігання скруглення на прилеглі поверхні, що визначається системною змінною **swFilletOverFlowType_e** [10]: 0 – скруглення за замовчуванням. Система обирає відповідний метод для створення скруглення. Коли поверхня скруглення поєднує суміжні поверхні, вона або плавно зливається із сусідніми поверхнями, або обмежує поверхню скруглення сусідніми кромками не змінюючи край, або обрізає поверхню фаски прилеглою поверхнею, на яку скруглення набігає. Ця опція намагається створити скруглення, коли це можливо. 1 – краї скруглення, що набігають, не змінюються. Поверхня скруглення створюється усіма сусідніми краями. У результаті, для завершення скруглення може бути потрібна додаткова перехідна поверхня скруглення; 2 – поверхня скруглення поєднується з суміжними поверхнями та згладжується або обшивається суміжними поверхнями. У результаті мало вірогідно, що поверхню скруглення можна побудувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.091, -0.00031, 0.0069, False, 1, Nothing, 0)
                              'вибір кромки для скруглення
Part.FeatureManager.FeatureFillet (195, 0.01, 0, 0, 0,
0, 0)                         'скруглення радіусом 10 мм
```

Причому перший аргумент 195 = 128 + 64 +2 + 1, тобто будується розповсюджене скруглення пов'язаних кромок зі збереженням елементів з постійним радіусом.

8.3.2. Метод інструменту Фаска

Послідовність побудови фаски наведено на рис. 8.10.

Рис. 8.10. Послідовність побудови фаски

Синтаксис

```
pFeat = FeatureManager.InsertFeatureChamfer ( o, cT, w, a, oD, vCD1, vCD2, vCD3)
```

Аргументи методу **InsertFeatureChamfer** наведено у табл. 8.10.

Таблиця 8.10.

Аргументи методу `InsertFeatureChamfer`

Аргумент	Тип	Призначення
o	long	Опції фаски, визначені системною змінною **swFeatureChamferOption_e** [10]: 1 – змінити напрям фаски; 2 – фаска зі збереженням елементів; 4 – дотичне розповсюдження; 8 – розповсюдження по деталях.
cT	long	Тип фаски, визначуваний системною змінною **swChamferType_e** [10]: 1 – фаска за кутом та відстанню; 2 – фаска за двома різними відстанями; 3 – фаска на вершині; 16 – фаска за двома однаковими відстанями.
w	double	Якщо аргумент **cT =** 1, то вказується відстань.
a	double	Якщо аргумент **cT** дорівнює 1, то вказується кут.
oD	double	Якщо аргумент **cT =** 2 або 3, то вказується єдина однакова відстань.
vCD1	double	Якщо аргумент **cT =** 1 або 3, то вказується відстань для першої сторони.
vCD2	double	Якщо аргумент **cT =** 1 або 3, то вказується відстань для другої сторони.
vCD3	double	Якщо аргумент **cT =** 3, то вказується відстань для третьої сторони.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
-8.8E-05, 9.02E-05, 0.0027, True, 0, Nothing, 0)
                          'вибір кромки для фаски
Part.FeatureManager.InsertFeatureChamfer 4, 1, 0.01,
0.78, 0, 0, 0, 0          'побудова фаски 10 мм під кутом
                          '0.78 рад.
```

8.4. Методи побудови довідкової геометрії

8.4.1. Метод інструменту Справочная система координат

Послідовність побудови довідкової системи координат наведено на рис. 8.11.

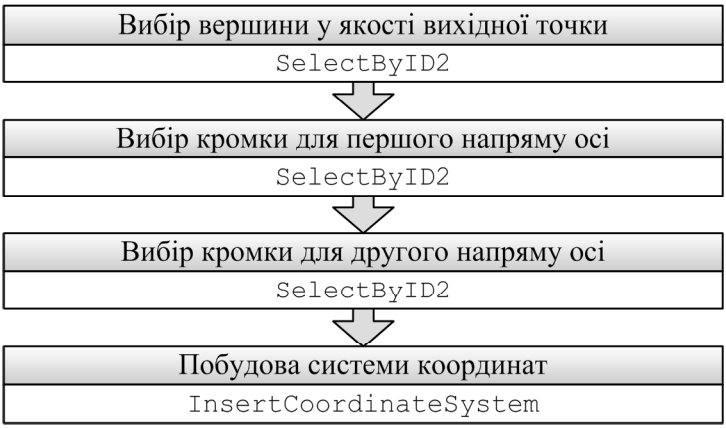


Рис. 8.11. Послідовність побудови довідкової системи координат

Синтаксис

retval = ModelDoc2.InsertCoordinateSystem (xF,yF,zF)

де:

xF, **yF**, **zF** – логічні аргументи; `True` – змінити напрям осей **X**, **Y** або **Z**; `False` – залишити напрям осей **X**, **Y** або **Z** за замовчуванням.

Примітка. При виборі кромки твердого тіла методом **SelectByID2** застосовується вибір указанням точки на цій грані або кромці. При виборі осі **X** аргумент **m =** `2`, для осі **Y** – **m =** `4`, для осі **Z** – **m =** `8`.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "VERTEX",
0, 0, 0.01, False, 1, Nothing, 0)
                          'вибір вершини для початку
                          'координат
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.015, -2.49E-04, 0.0101, True, 2, Nothing, 0)
                          'вибір кромки для осі X,
                          'аргумент m = 2
boolstatus = Part.Extension.SelectByID2("", "EDGE",
-3.98E-04, -0.011, 0.0101, True, 4, Nothing, 0)
                          'вибір кромки для осі Y,
                          'аргумент m = 4
Part.InsertCoordinateSystem False, False, False
                          'побудова системи координат
```

8.4.2. Метод інструменту Точка

Послідовність побудови довідкової точки наведено на рис. 8.12.

Рис. 8.12. Послідовність побудови довідкової точки

Синтаксис

pRefPointFeatures=FeatureManager.InsertReferencePoint (nRPT, nRP, dDoP, nORP)

Аргументи методу **InsertReferencePoin** наведено у табл. 8.11.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.022, 0.034, 0.009, True, 0, Nothing, 0)
                          'вибір першої кромки
boolstatus = Part.Extension.SelectByID2("", "EDGE",
-0.03, -0.009, 0.01, True, 0, Nothing, 0)
                          'вибір другої кромки
vRefPointFeatures = Part.FeatureManager.
InsertReferencePoint(6, 0, 0.001, 40)
                          'побудова довідкової точки
```

Таблиця 8.11

Аргументи методу `InsertReferencePoint`

Аргумент	Тип	Призначення
nRPT	long	Тип довідкової точки визначається системною змінною **swRefPointType_e** [10]: 0 – невірно; 1 – не визначено; 2 – уздовж кривої; 3 – центр кромки; 4 – центр грані; 5 – проекція вершини грані; 6 – перетин; 7 – точка ескізу.
nRP	long	Відстань у процентах або рівномірне розподілення визначаються системною змінною **swRefPointAlongCurveType_e** [10]: 0 – відстань уздовж кривої; 1 – рівномірне розподілення уздовж кривої; 2 – проценти відстані уздовж кривої.
dDoP	double	Відстань, на якій створюються довідкові точки або проценти довжини на обраній лінії, якщо аргумент **nRP =** 0 або 2.
nORP	long	Кількість довідкових точок, що створюються та рівномірно розподіляються уздовж кривої, якщо аргумент **nRP =** 1.

8.4.3. Метод інструменту Ось

Послідовність побудови фаски наведено на рис. 8.13.

Рис. 8.13. Послідовність побудови довідкової осі

Синтаксис

```
retval = ModelDoc2.InsertAxis2 ( aS )
```

де:
aS – логічний аргумент. `True` – розмір осі буде встановлено автоматично, `False` – не автоматично.

Приклади

Побудова осі на кромці

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.05, 0.054, 0.009, True, 0, Nothing, 0)
                          'вибір кромки
boolstatus = Part.InsertAxis2(True)
                          'побудова осі на кромці
```

Побудова осі за двома вершинами

```
boolstatus = Part.Extension.SelectByID2("", "VERTEX",
0.07, -0.05, 0.01, True, 0, Nothing, 0)
                          'вибір першої вершини
boolstatus = Part.Extension.SelectByID2("", "VERTEX",
0, 0, 0.01, True, 0, Nothing, 0)
                          'вибір другої вершини
boolstatus = Part.InsertAxis2(True)
                          'побудова осі за двома
                          'вершинами
```

8.4.4. Метод інструменту Плоскость

Послідовність побудови довідкової площини наведено на рис. 8.14.

Рис. 8.14. Послідовність побудови довідкової площини

Синтаксис

```
retval = IFeatureManager.InsertRefPlane ( FC,
FCAOD, SC, SCAOD, TC, TCAOD)
```

Аргументи методу **InsertRefPlane** наведено у табл. 8.12.

Таблиця 8.12

Аргументи методу `InsertRefPlane`

Аргумент	Тип	Призначення
FC	integer	Перша прив'язка визначається системною змінною **swRefPlaneReferenceConstraints_e** [10]: 1 – паралельно; 2 – перпендикулярно; 4 – збіг; 8 – на відстані; 16 – під кутом; 32 – дотично; 64 – проекція; 128 – середня площина.
FCAOD	double	Кут або відстань відносно першої прив'язки.
SC	integer	Друга прив'язка визначається системною змінною **swRefPlaneReferenceConstraints_e** [10].
SCAOD	double	Кут або відстань відносно другої прив'язки.
TC	integer	Третя прив'язка визначається системною змінною **swRefPlaneReferenceConstraints_e** [10].
TCAOD	double	Кут або відстань відносно третьої прив'язки.

Примітка. При виборі об'єктів твердого тіла як прив'язок методом **`SelectByID2`** для першої прив'язки аргумент **`m`** повинен дорівнювати `0`, для другої прив'язки – **`m =`** `1`, для третьої – **`m =`** `2`.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.20, -0.007, 0.009, True, 0, Nothing, 0)
                          'вибір першої кромки
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.07, -0.005, 0.01, True, 1, Nothing, 0)
                          'вибір другої кромки
Dim myRefPlane As Object
Set myRefPlane = Part.FeatureManager.InsertRefPlane(4,
0, 4, 0, 0, 0)
                          'побудова площини на основі
                          'двох кромок
```

8.5. Масиви

8.5.1. Метод інструменту Зеркальное отражение

Послідовність побудови дзеркального відображення наведено на рис. 8.15.

Рис. 8.15. Послідовність побудови дзеркального відображення

Синтаксис

```
retval = FeatureManager.InsertMirrorFeature (
bMB, bGP, bM, bK)
```

Аргументи методу **`InsertMirrorFeature`** наведено у табл. 8.13.

Таблиця 8.13

Аргументи методу `InsertMirrorFeature`

Аргумент	Тип	Призначення
`bMB`	`boolean`	`True` – відобразити тверді тіла; `False` – відобразити елементи побудови.
`bGP`	`boolean`	`True` – відобразити тільки геометрію елементу; `False` – вирішити увесь елемент. Використовується тільки для відображення елементів.
`bM`	`boolean`	`True` – поєднати будь-які відображені тверді тіла; `False` – не поєднувати. Використовується тільки при відображенні твердих тіл.
`bK`	`boolean`	`True` – пов'язати поверхні; `False` – не пов'язувати. Використовується тільки для відображення поверхонь.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Плоскость1",
"PLANE", 0, 0, 0, False, 2, Nothing, 0)
                          'вибір площини віддзеркалення
boolstatus = Part.Extension.SelectByID2("Бобышка-
Вытянуть22", "BODYFEATURE", 0, 0, 0, True, 1, Nothing, 0)
                          'вибір елементу для
                          'відображення
Part.FeatureManager.InsertMirrorFeature (False, False,
False, False)             'побудова дзеркального
                          'відображення елементу побудови
```

8.5.2. Метод інструменту Линейный массив

Послідовність побудови лінійного масиву наведено на рис. 8.16.

Рис. 8.16. Послідовність побудови лінійного масиву

Синтаксис

```
Retval=FeatureManager.FeatureLinearPattern2 ( n1,
s1, n2, s2, fD1, fD2, dN1, dN2, gP)
```

Аргументи методу **FeatureLinearPattern2** наведено у табл. 8.14.

Таблиця 8.14

Аргументи методу FeatureLinearPattern2

Аргумент	Тип	Призначення
1	2	3
n1	long	Кількість екземплярів лінійного масиву у напряму 1, враховуючи оригінал.
s1	double	Крок між екземплярами масиву в напряму 1 у метрах.

Продовження табл. 8.14

1	2	3
n2	long	Кількість екземплярів лінійного масиву в напряму 2, враховуючи оригінал.
s2	double	Крок між екземплярами масиву у напряму 2 у метрах.
fD1	boolean	True – змінити напрям 1 лінійного масиву; False – не змінювати.
fD2	boolean	True – змінити напрям 2 лінійного масиву; False – не змінювати.
dN1	string	Ім'я напряму 1.
dN2	string	Ім'я напряму 2.
gP	boolean	True – використати геометричний шаблон; False – не використовувати.

Примітка. При виборі об'єктів для копіювання, аргумент **m** методу **SelectByID2** повинен дорівнювати 4, при виборі об'єктів для напрямів **X** і **Y** він повинен дорівнювати 1 і 2 відповідно.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Вытянутая-
Бобышка2", "BODYFEATURE", 0, 0, 0, False, 4, Nothing, 0)
                          'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.01, 0.03, 0.009, True, 1, Nothing, 0)
                          'вибір кромки для напряму X
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.07, 0.01, 0.009, True, 2, Nothing, 0)
                          'вибір кромки для напряму Y
Set myFeature = Part.FeatureManager.
FeatureLinearPattern2(3, 0.03, 3, 0.03, False, True,
"NULL", "NULL", False)   'побудова масиву
```

8.5.3. Метод інструменту Круговой массив

Послідовність побудови кругового масиву наведено на рис. 8.17.

Синтаксис

```
Retval=FeatureManager.FeatureCircularPattern2( n,
s, fD, dN, gP)
```

Аргументи методу **FeatureCircularPattern2** наведено у табл. 8.15.

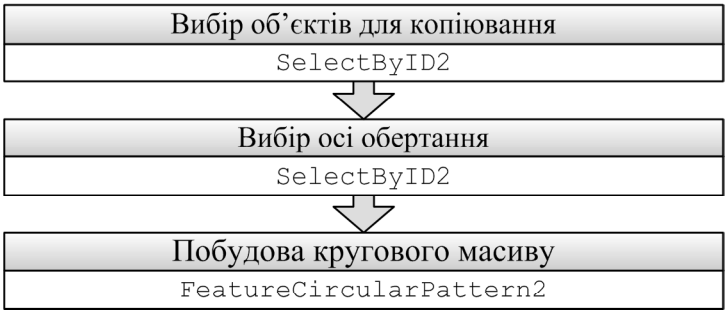


Рис. 8.17. Послідовність побудови кругового масиву

Таблиця 8.15

Аргументи методу `FeatureCircularPattern2`

Аргумент	Тип	Призначення
n	long	Кількість екземплярів кругового масиву включно з оригіналом.
s	double	Крок між екземплярами у радіанах.
fD	boolean	True – змінити напрям кругового масиву; False – не змінювати.
dN	string	Ім'я кругового розміру.
gP	boolean	True – використати геометричний шаблон, False – не використовувати.

Примітка. При виборі об'єктів для копіювання аргумент **m** методу **`SelectByID2`** повинен дорівнювати 4, при виборі осі обертання він повинен дорівнювати 1.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Бобышка-
Вытянуть2", "BODYFEATURE", 0, 0, 0, False, 4, Nothing, 0)
                          'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("Бобышка-
Вытянуть3", "BODYFEATURE", 0, 0, 0, True, 4, Nothing, 0)
                          'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("Ось1",
"AXIS", 0, 0, 0, True, 1, Nothing, 0)
                          'вибір осі обертання
Set myFeature = Part.FeatureManager.FeatureCircular-
Pattern2(7, 0.2617993877992, False, "NULL", False)
                          'побудова кругового масиву
```

8.5.4. Метод інструменту Массив, управляемый эскизом

Послідовність побудови масиву, керованого ескізом, наведено на рис. 8.18.

Рис. 8.18. Послідовність побудови масиву, керованого ескізом

Синтаксис

```
pFeat = FeatureManager.FeatureSketchDrivenPattern (
uC, bGP)
```

де:

uC – логічний аргумент; `True` – використати центр ваги як точку відліку, `False` – не використовувати.

bGP – логічний аргумент; `True` – використати геометричний шаблон, `False` – не використовувати.

Примітка. При виборі об'єктів для копіювання, аргумент **m** методу **SelectByID2** повинен дорівнювати `4`, при виборі управляючого ескізу він повинен дорівнювати `64`.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Бобышка-
Вытянуть2", "BODYFEATURE", 0, 0, 0, False, 4, Nothing, 0)
                          'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("Эскиз9",
"SKETCH", 0, 0, 0, True, 64, Nothing, 0)
                          'вибір управляючого ескізу
Part.FeatureManager.FeatureSketchDrivenPattern (True,
False)
                          'побудова масиву, керованого
                          'ескізом
```

8.5.5. Метод інструменту Массив, управляемый кривой

Послідовність побудови масиву, керованого кривою, наведено на рис. 8.19.

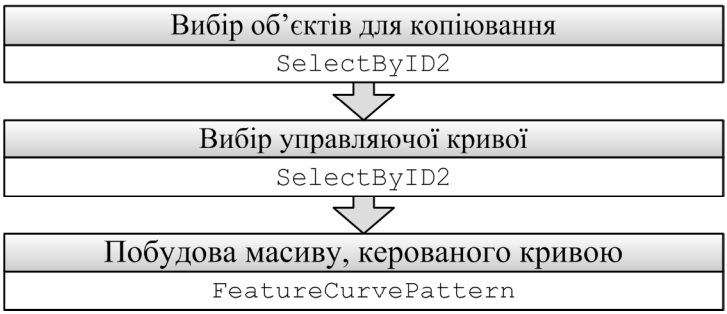


Рис. 8.19. Послідовність побудови масиву, керованого кривою

Синтаксис

```
ModelDoc2.FeatureCurvePattern ( n1, s1, n2, s2,
fD1, fD2, eS1, eS2, uC, aTS, oC, pSO )
```

Аргументи методу **FeatureCurvePatter** наведено у табл. 8.16.

Таблиця 8.16

Аргументи методу FeatureCurvePatter

Аргумент	Тип	Призначення
1	2	3
n1	long	Кількість екземплярів у першому напрямі включно з оригіналом.
s1	double	Крок у радіанах.
n2	long	Кількість екземплярів у другому напрямі включно з оригіналом.
s2	double	Крок у радіанах.
fD1	boolean	True – змінити перший напрям.
fD2	boolean	True – змінити другий напрям.
eS1	boolean	True – використовувати рівномірний розподіл екземплярів у першому напряму.
eS2	boolean	True – використовувати рівномірний розподіл екземплярів у другому напряму.
uC	boolean	True – використати центр ваги як точку відліку.

Продовження табл. 8.16

1	2	3
aTS	boolean	True – вирівнювати нові об'єкти з базовим екземпляром.
oC	boolean	True – компенсувати криву; False – трансформувати криву.
pSO	boolean	True – використовувати тільки базовий екземпляр у другому напряму.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Вытянуть2",
"BODYFEATURE", 0, 0, 0, False, 4, Nothing, 0)
                              'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("Кривая1",
"REFERENCECURVES", 0, 0, 0, True, 1, Nothing, 0)
                              'вибір управляючої кривої
Part.FeatureCurvePattern (6, 0.015, 1, 0.01, False,
False, False, False, True, True, False, False)
                              'побудова масиву, керованого
                              'кривою
```

8.5.6. Метод інструменту Образец заполнения

Послідовність побудови зразку заповнення наведено на рис. 8.20.

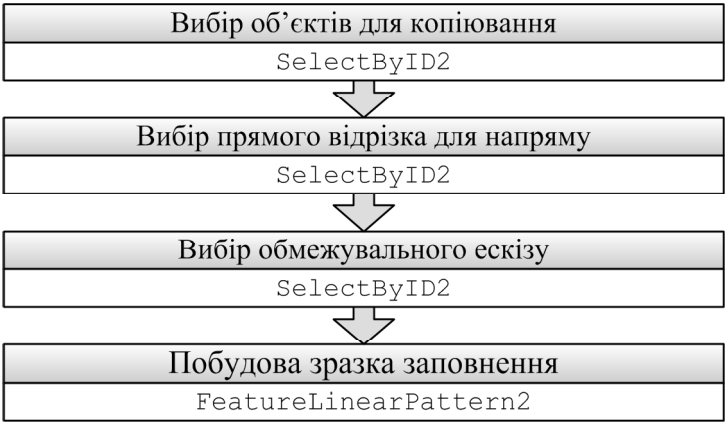


Рис. 8.20. Послідовність побудови зразка заповнення

Синтаксис

Retval=FeatureManager.FeatureLinearPattern2 (n1, s1, n2, s2, fD1, fD2, dN1, dN2, gP)

Аргументи методу **FeatureLinearPattern2** наведено у табл. 8.17.

Таблиця 8.17

Аргументи методу FeatureLinearPattern2

Аргумент	Тип	Призначення
n1	long	Кількість екземплярів у напряму 1 включно з оригіналом.
s1	double	Крок між екземплярами масиву у напряму 1 у метрах.
n2	long	Кількість екземплярів у напряму 2 включно з оригіналом.
s2	double	Крок між екземплярами масиву у напряму 2 у метрах.
fD1	boolean	True – змінити напрям 1; False – не змінювати.
fD2	boolean	True – змінити напрям 2; False – не змінювати.
dN1	string	Ім'я напряму 1.
dN2	string	Ім'я напряму 2.
gP	boolean	True – використати геометричний шаблон; False – не використовувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Бобышка-
Вытянуть3", "BODYFEATURE", 0, 0, 0, False, 4, Nothing, 0)
                          'вибір об'єкта для копіювання
boolstatus = Part.Extension.SelectByID2("Line1@
Эскиз10", "EXTSKETCHSEGMENT", 0, 0, 0, True, 1,
Nothing, 0)
                          'вибір прямого відрізка для
                          'направлення
boolstatus = Part.Extension.SelectByID2("Эскиз10",
"SKETCH", 0, 0, 0, True, 16384, Nothing, 0)
                          'вибір ескізу для обмеження
Set myFeature = Part.FeatureManager.
FeatureLinearPattern2(3, 0.03, 1, 0.03, False, False,
"NULL", "NULL", False)
                          'побудова зразка заповнення
```

8.6. Приклад розроблення підсистеми побудови твердотільної моделі глухої кришки підшипникового вузла

Розглянемо проектну об'єктну підсистему для побудови моделі кришки глухої підшипникового вузла для підшипника із зовнішнім діаметром 80 мм згідно з ГОСТ 18511–73 (рис. 8.21).

Процедура не містить форми користувача. Кришка будується у три етапи. На першому етапі будується основна частина кришки за допомогою інструмента **Повернутая бобышка** (рис. 8.22). Ескіз містить 8 прямих відрізків та 6 розмірів. На другому етапі вирізається отвір під кріплення за допомогою інструменту **Повернутый вырез** (рис. 8.23). Ескіз містить 6 прямих відрізків та 5 розмірів. На третьому етапі будується круговий масив, що містить 6 отворів під кріплення.

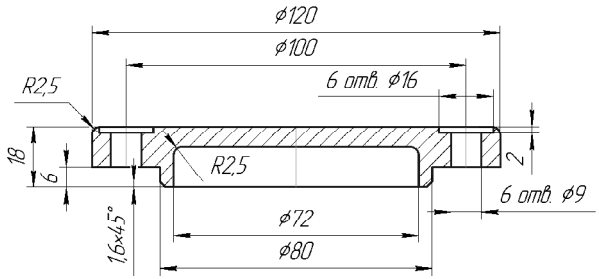
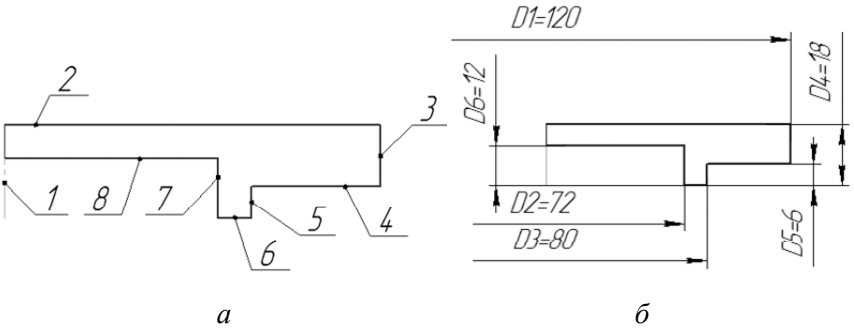
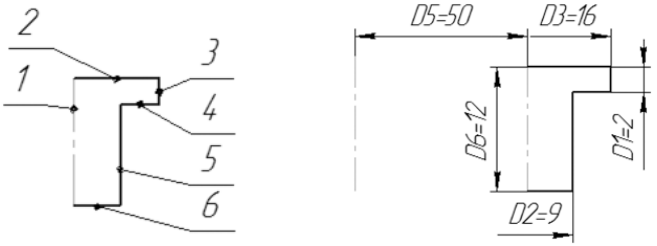


Рис. 8.21. Креслення кришки глухої підшипникового вузла



а *б*

Рис. 8.22. Послідовність побудови ескізу для першого етапу: *а* – нумерація відрізків; *б* – нумерація розмірів



а *б*

Рис. 8.23. Послідовність побудови ескізу для другого етапу: *а* – нумерація відрізків; *б* – нумерація розмірів

Головна програма.

```
Dim swApp As Object, Part As Object
Dim myFeature As Object, skSegment As Object
Dim myDisplayDim As Object, myDimension As Object
Dim boolstatus As Boolean, longwarnings As Long
Dim longstatus As Long  'оголошення змінних
Sub main()              'оголошення головної програми
Set swApp = Application.SldWorks
                        'активація SolidWorks
Set Part = swApp.NewDocument("C:\ProgramData\ SolidWorks\
SolidWorks 2011\templates\Деталь.prtdot", 0,0,0)
                        'створення документа за
                        'шаблоном Деталь.prtdot
Set Part = swApp.ActiveDoc
                        'активація документа
   '- Перший етап. Побудова основної частини кришки -
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір площини Спереди для
                        'створення ескізу
Part.SketchManager.InsertSketch True
                        'запуск режиму побудови ескізу
Set skSegment = Part.SketchManager.
CreateCenterLine(0#, 0#, 0#, 0#, -0.018, 0#)
                        'побудова осьової лінії №1
                        'згідно з рис. 8.22, а
Set skSegment = Part.SketchManager.CreateLine(0#, 0#,
0#, 0.061, 0#, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.061,
0#, 0#, 0.061, -0.012, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.061,
-0.011, 0#, 0.043, -0.011, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.043,
-0.011, 0#, 0.043, -0.018, 0#)
```

```
Set skSegment = Part.SketchManager.CreateLine(0.043,
-0.018, 0#, 0.034, -0.018, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.034,
-0.018, 0#, 0.034, -0.0059, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.034,
-0.0059, 0#, 0#, -0.0059, 0#)
                        'побудова ліній 2-8 згідно з
                        'рис. 8.22, а
Part.ClearSelection2 True
                        'скасування усіх виділень
boolstatus= swApp.SetUserPreferenceToggle ( 10, False)
                        'вимкнення підтвердження
                        'значення розміру
boolstatus = Part.Extension.SelectByID2("Line3",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(-0.019, 0.012, 0)
Set myDimension = Part.Parameter("D1@Эскиз1")
myDimension.SystemValue = 0.12
                        'створення розміру D1 згідно з
                        'рис. 8.22, б
boolstatus = Part.Extension.SelectByID2("Line7",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(-0.02, -0.011, 0)
Set myDimension = Part.Parameter("D2@Эскиз1")
myDimension.SystemValue = 0.072
                        'створення розміру D2 згідно з
                        'рис. 8.22, б
boolstatus = Part.Extension.SelectByID2("Line5",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(-0.017, -0.02, 0)
Set myDimension = Part.Parameter("D3@Эскиз1")
myDimension.SystemValue = 0.08
                        'створення розміру D3 згідно з
                        'рис. 8.22, б
boolstatus = Part.Extension.SelectByID2("Line2",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line6",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.081, -0.004, 0)
Set myDimension = Part.Parameter("D4@Эскиз1")
myDimension.SystemValue = 0.018
                        'створення розміру D4 згідно з
                        'рис. 8.22, б
```

```
boolstatus = Part.Extension.SelectByID2("Line4",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line6",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.069, -0.013, 0)
Set myDimension = Part.Parameter("D5@Эскиз1")
myDimension.SystemValue = 0.006
                          'створення розміру D5 згідно з
                          'рис. 8.22, б
boolstatus = Part.Extension.SelectByID2("Line8",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line6",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.014, -0.015, 0)
Set myDimension = Part.Parameter("D6@Эскиз1")
myDimension.SystemValue = 0.012
                          'створення розміру D6 згідно з
                          'рис. 8.22, б
Part.ClearSelection2 True 'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Line8",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line7",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 1)
Set skSegment=Part.SketchManager.
CreateFillet(0.0025,1)    'побудова скруглення радіусом
                          '2,5 мм між лініями 7 і 8
boolstatus = Part.Extension.SelectByID2("Line2",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line3",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 1)
Set skSegment=Part.SketchManager.
CreateFillet(0.0025,1)    'побудова скруглення радіусом
                          '2,5 мм між лініями 2 і 3
boolstatus = Part.Extension.SelectByID2("Line5",
"SKETCHSEGMENT", 0, 0, 0, False, 1, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line6",
"SKETCHSEGMENT", 0, 0, 0, True, 1, Nothing, 1)
Set skSegment = Part.SketchManager.CreateChamfer(2,
0.0001, 0.0001)           'побудова фаски розміром 1 мм
                          'між лініями 5 і 6
Part.ClearSelection2 True 'скасування усіх виділень
Part.SketchManager.InsertSketch True
                          'вихід з режиму побудови ескізу
                          'зі збереженням змін
boolstatus = Part.Extension.SelectByID2("Эскиз1","SKETCH",
0,0,0,False,0,Nothing,0) 'вибір ескізу для обертання
boolstatus = Part.Extension.SelectByID2("Line1@
Эскиз1", "EXTSKETCHSEGMENT", -0.15, 0.015, 0,
True,16,Nothing,0)        'вибір прямого відрізку у
                          'якості осьової лінії
```

```
Set myFeature = Part.FeatureManager.
FeatureRevolve2(True, True, False, False, False,
False, 0, 0, 6.283185, 0, False, False, 0.01, 0.01, 0,
0, 0, True, True, True) 'побудова основної частини
                        'кришки за допомогою інструменту
                        'Повернутая бобышка
    '- Другий етап. Побудова отвору під кріплення -
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, False, 0, Nothing, 0)
                        'вибір площини Спереди для
                        'створення ескізу
Part.SketchManager.InsertSketch True
                        'запуск режиму побудови ескізу
Set skSegment = Part.SketchManager.
CreateCenterLine(0.046472, 0#, 0#, 0.046, -0.012, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.046,
0#, 0#, 0.056, 0#, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.056,
0#, 0#, 0.056, -0.0038, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.056,
-0.0038, 0#, 0.051, -0.0038, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.051,
-0.0038, 0#, 0.051, -0.012, 0#)
Set skSegment = Part.SketchManager.CreateLine(0.051,
-0.012, 0#, 0.046, -0.012, 0#)
                        'побудова ліній згідно з
                        'рис 8.23, а
Part.ClearSelection2 True
                        'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Line2",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line4",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim =Part.AddDimension2(0.066, -0.0015, 0)
Set myDimension = Part.Parameter("D1@Эскиз2")
myDimension.SystemValue = 0.002
                        'створення розміру D1 згідно з
                        'рис. 8.23, б
boolstatus = Part.Extension.SelectByID2("Line5",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim =Part.AddDimension2(0.041, -0.0068, 0)
Set myDimension = Part.Parameter("D2@Эскиз2")
myDimension.SystemValue = 0.009
                        'створення розміру D2 згідно з
                        'рис. 8.23, б
```

```
boolstatus = Part.Extension.SelectByID2("Line3",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.039, 0.003, 0)
Set myDimension = Part.Parameter("D3@Эскиз2")
myDimension.SystemValue = 0.016
                          'створення розміру D3 згідно з
                          'рис. 8.23, б
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Point1@Исходная
точка", "EXTSKETCHPOINT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.013, 0.0049, 0)
Set myDimension = Part.Parameter("D4@Эскиз2")
myDimension.SystemValue = 0.05
                          'створення розміру D4 згідно з
                          'рис. 8.23, б
boolstatus = Part.Extension.SelectByID2("Line2",
"SKETCHSEGMENT", 0, 0, 0, False, 0, Nothing, 0)
boolstatus = Part.Extension.SelectByID2("Line6",
"SKETCHSEGMENT", 0, 0, 0, True, 0, Nothing, 0)
Set myDisplayDim = Part.AddDimension2(0.064, -0.012, 0)
Set myDimension = Part.Parameter("D5@Эскиз2")
myDimension.SystemValue = 0.012
                          'створення розміру D5 згідно з
                          'рис. 8.23, б
Part.SketchManager.InsertSketch True
                          'вихід з режиму побудови ескізу
                          'із збереженням змін
boolstatus = Part.Extension.SelectByID2("Эскиз2",
"SKETCH", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір ескізу для вирізання
boolstatus = Part.Extension.SelectByID2("Line1",
"SKETCHSEGMENT", 0.05, 0, 0, True, 16, Nothing, 0)
                          'вибір прямого відрізка у
                          'якості осьової лінії
Set myFeature = Part.FeatureManager.
FeatureRevolve2(True, True, False, True, False, False,
0, 0, 6.28318530718, 0, False, False, 0.01, 0.01, 0,
0, 0, True, True, True) 'побудова кріпильного отвору
                          'за допомогою інструменту
                          'Повернутый вырез
  '- Третій етап. Побудова кругового масиву отворів -
boolstatus = Part.Extension.SelectByID2("", "FACE",
0.06, -0.01, 0, True, 0, Nothing, 0)
                          'вибір зовнішньої циліндричної
                          'грані кришки для побудови осі
boolstatus = Part.InsertAxis2(True)
                          'побудова довідкової осі
```

```
Part.ClearSelection2 True 'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Вырез-Повер-
нуть1", "BODYFEATURE", 0, 0, 0, False,4,Nothing,0)
                          'вибір операції побудови для
                          'копіювання
boolstatus = Part.Extension.SelectByID2("Ось1",
"AXIS", 0, 0, 0, True, 1, Nothing, 0)
                          'вибір осі обертання масиву
Set myFeature = Part.FeatureManager.
FeatureCircularPattern2(6, 1.0471975511966, False,
"NULL", False)            'побудова кругового масиву,що
                          'містить 6 отворів під кріплення
boolstatus = swApp.SetUserPreferenceToggle ( 10, True)
                          'включення підтвердження
                          'значення розміру
End Sub                   'завершення головної програми
```

Результат роботи проектної процедури наведено на рис. 8.24.

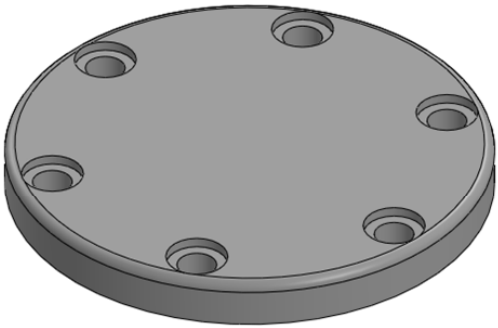


Рис. 8.24. Результат роботи проектної процедури

Контрольні запитання і завдання

1. Яка послідовність дій та які методи використовуються для побудови витягнутої бобишки?
2. Яка послідовність дій та які методи використовуються для побудови повернутої бобишки?
3. Назвіть послідовність дій та методи, що використовуються для побудови витягнутої бобишки за траєкторією.
4. Яка послідовність дій та які методи використовуються для побудови витягнутої бобишки за перетинами?

5. Назвіть послідовність дій та методи, що використовуються для побудови витягнутого вирізу.
6. Яка послідовність дій та які методи використовуються для побудови повернутого вирізу?
7. Назвіть послідовність дій та методи, що використовуються для побудови вирізу за траєкторією.
8. Яка послідовність дій та які методи використовуються для побудови вирізу за перетинами?
9. Яка послідовність дій та які методи використовуються для побудови скруглення?
10. Назвіть послідовність дій та методи, що використовуються для побудови фаски.
11. Яка послідовність дій та які методи використовуються для побудови довідкової системи координат?
12. Назвіть послідовність дій та методи, що використовуються для побудови довідкової точки.
13. Яка послідовність дій та які методи використовуються для побудови довідкової осі?
14. Яка послідовність дій та які методи використовуються для побудови довідкової площини?
15. Назвіть послідовність дій та методи, що використовуються для побудови дзеркального відображення.
16. Назвіть послідовність дій та методи, що використовуються для побудови лінійного масиву.
17. Яка послідовність дій та які методи використовуються для побудови кругового масиву?
18. Яка послідовність дій та які методи використовуються для побудови масиву, керованого ескізом?
19. Назвіть послідовність дій та методи, що використовуються для побудови масиву, керованого кривою?
20. Яка послідовність дій та які методи використовуються для побудови зразка заповнення?
21. Модифікуйте проектну процедуру побудови кришки глухої підшипникового вузла, додавши до неї форму користувача з можливістю задавати діаметр підшипника та кількість кріпильних отворів.

9. Основні методи побудови креслень

9.1. Види на кресленнях

9.1.1. Створення виду за моделлю

Послідовність створення виду за моделлю не передбачає ніяких попередніх дій.

Синтаксис

```
Retval = DrawingDoc.CreateDrawViewFromModelView3(
MN, VN, LocX, LocY, LocZ)
```

Аргументи методу **CreateDrawViewFromModelView3** наведено у табл. 9.1.

Таблиця 9.1

Аргументи методу `CreateDrawViewFromModelView3`

Аргумент	Тип	Призначення
MN	string	Найменування документа моделі, з якої потрібно створити вид (включаючи повний шлях до файлу на диску та розширення .sldprt або .sldasm).
VN	string	Назва виду моделі,з якої створюється вид креслення.
LocX	double	**X** – координата розташування виду на аркуші від центру креслення.
LocY	double	**Y** – координата розташування виду на аркуші від центру креслення.
LocZ	double	**Z** – координата розташування виду на аркуші від центру креслення.

Приклад

```
Set DrawView = Part.CreateDrawViewFromModelView2("D:\
Documents\Деталь1.SLDPRT", "*СВЕРХУ", 0, 0, 0)
                        'створення виду на основі
                        'деталі з файлу Деталь1.SLDPRT
                        'за її видом зверху
```

9.1.2. Проекція

Послідовність створення проекції на основі базового виду за моделлю наведено на рис 9.1.

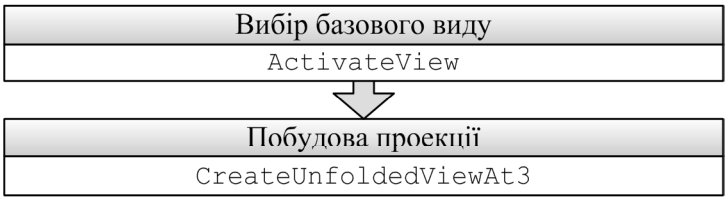


Рис. 9.1. Послідовність створення проекції

Синтаксис

```
retval = DrawingDoc.CreateUnfoldedViewAt3 ( x, y, z, nA)
```

де:

X, **Y**, **Z** – координати розташування виду на аркуші від центру креслення. Зображення на проекційнім виді залежить ‘від розташування цього виду відносно базового;

nA – аргумент логічного типу; `True` – розірвання зв’язку з базовим видом; `False` – зв’язок з базовим видом залишається.

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид2")
                              ‘вибір виду для проекції
SetDrawView = Part.CreateUnfoldedViewAt3(0.068, 0.22, 0, 0)
                              ‘створення проекції
```

9.1.3. Розріз

Послідовність створення розрізу на основі базового виду за моделлю наведено на рис 9.2.

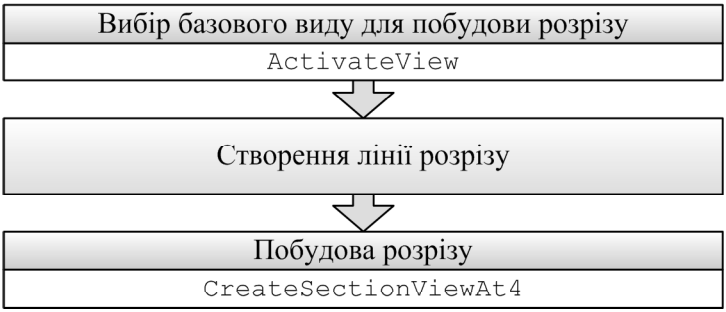


Рис. 9.2. Послідовність створення розрізу

Синтаксис

```
retval = DrawingDoc.CreateSectionViewAt4 ( X, Y, Z, SL, O, EC)
```

Аргументи методу **CreateSectionViewAt4** наведено у табл. 9.2.

Таблиця 9.2

Аргументи методу CreateSectionViewAt4

Аргумент	Тип	Призначення
X,Y,Z	double	Координати розташування виду на аркуші від центру креслення.
SL	string	Буквена назва розрізу.
O	long	Параметри відображення розрізу визначаються системною змінною **swCreateSectionViewAtOptions_e** [10]: 1 – розріз не відривається від базового виду; 2 – створюється вирівняний розріз; 4 – напрям розрізу буде змінено; 8 – вид розрізу буде мати масштаб базового виду; 16 – буде створено частковий розріз; 32 – будуть розрізані тільки грані, по яких проходить лінія розрізу.
EC	Variant	Масив деталей, що не розсікаються.

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид2")
                              'вибір виду для розрізу
Set skSegment = Part.SketchManager.CreateLine(0, 0.03,
0#, 0#, 0#, 0#)               'побудова лінії для розрізу
Set myView = Part.CreateSectionViewAt5(0.09, 0.21, 0,
"A", 0, (excludedComponents), 0)
                              'створення розрізу А
```

9.1.4. Вирівняний розріз

Для побудови вирівняного розрізу використовується той самий метод, що і для побудови звичайного розрізу. Відмінність одного розрізу від іншого визначається аргументом **O**. Для вирівняного розрізу він повинен бути **O = 2**.

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид2")
                          'вибір виду для розрізу
Set skSegment = Part.SketchManager.CreateLine(0, 0.03,
0#, 0#, 0#, 0#)
Set skSegment = Part.SketchManager.CreateLine(0#, 0#,
0#, 0.02, -0.03, 0#)     'побудова ламаної лінії для
                          'вирівняного розрізу
Set DrawView = Part.CreateSectionViewAt4(0.19, 0.11,
0, "C", 2, ExcludedComponents)
                          'створення вирівняного розрізу
```

9.1.5. Місцевий вид

Послідовність створення місцевого виду на основі базового виду за моделлю наведено на рис 9.3.

Рис. 9.3. Послідовність створення місцевого виду

Синтаксис

```
retval = DrawingDoc.CreateDetailViewAt3 ( x, y,
z, s, s1, s2, lI, st, fo )
```

Аргументи методу **CreateDetailViewAt3** наведено у табл. 9.3.

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид5")
                          'вибір базового виду
SetSkCircle = Part.SketchManager.CreateCircle(0.033, 0,
0, 0.022, -0.0018, 0)     'контур у вигляді кола для
                          'місцевого виду
Part.CreateDetailViewAt3 (0.12, 0.22, 0, 0, 2#, 1#,
"D", 1, 0)                'створення місцевого виду
```

Таблиця 9.3

Аргументи методу `CreateDetailViewAt3`

Аргумент	Тип	Призначення
`X,Y,Z`	`double`	Координати розташування виду на аркуші від центру креслення.
`s`	`long`	Стиль для місцевого виду визначається системною змінною **`swDetViewStyle_e`** [10]: 1 – використовується стандартний стиль; 2 – розімкнене коло; 3 – з виноскою; 4 – без виноски; 5 – з'єднаний.
`s1`	`double`	Чисельник масштабу.
`s2`	`double`	Знаменник масштабу.
`lI`	`string`	Буквена назва виду.
`st`	`long`	Тип відображення місцевого виду визначається системною змінною **`swDetCircleShowType_e`** [10]: 1 – використати як контур ескіз користувача для створення місцевого виду; 2 – використати як контур коло для створення місцевого виду; 3 – не відображати контур на місцевому виді.
`fo`	`boolean`	`True` – відобразити повний контур виду; `False` – відображення неповного контуру.

9.1.6. Три стандартних види

Послідовність створення трьох стандартних видів за моделлю не передбачає ніяких попередніх дій.

Синтаксис

```
retval = DrawingDoc.Create1stAngleViews2 (mN)
```

де:

`mN` – назва документа моделі, з якої створюються три види.

Приклад

```
Part.Create1stAngleViews("D:\Documents\Деталь1.SLDPRT")
                         'створення трьох стандартних
                         'видів за моделлю
```

9.1.7. Вирив деталі

Послідовність створення вириву на основі базового виду за моделлю наведено на рис 9.4.

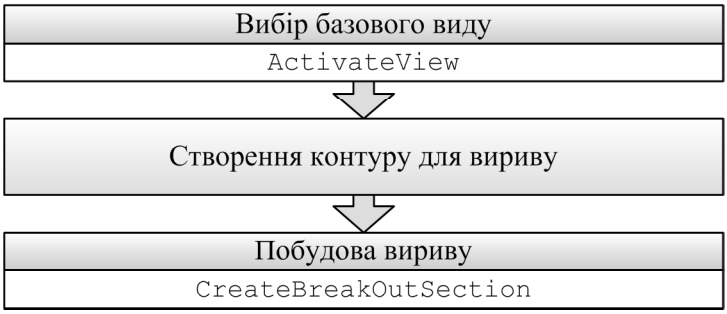


Рис. 9.4. Послідовність створення вириву

Синтаксис

```
retval = DrawingDoc.CreateBreakOutSection (d)
```

де:

d – глибина вириву у метрах.

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид9")
                          'вибір базового виду
SetSkCircle = Part.SketchManager.CreateCircle(0.033,
0, 0, 0.022, -0.0018, 0)
                          'контур у вигляді кола для
                          'вириву
Part.CreateBreakOutSection 0.028
                          'побудова вириву глибиною 28 мм
```

9.1.8. Лінія розриву

Послідовність створення лінії розриву на основі базового виду за моделлю наведено на рис 9.5.

Синтаксис

```
retval = View.InsertBreak ( O, P1, P2, S)
```

Аргументи методу **InsertBreak** наведено у табл. 9.4.

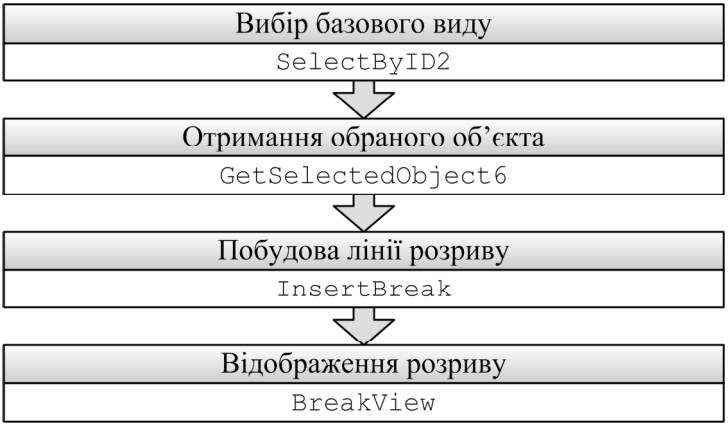


Рис. 9.5. Послідовність створення лінії розриву

Таблиця 9.4

Аргументи методу `InsertBreak`

Аргумент	Тип	Призначення
`O`	`long`	Горизонтальна або вертикальна лінія розриву, визначувана системною змінною **`swBreakLineOrientation_e`** [10]: 0 – горизонтально; 1 – вертикально.
`P1`	`double`	Розташування першої лінії розриву. Якщо лінія розташована горизонтально (**`O`** = 0), то тут записується **`Y`**–координата. Якщо лінія вертикальна (**`O`** = 1), то тут записується **`X`**–координата.
`P2`	`double`	Розташування другої лінії розриву. Якщо лінія розташована горизонтально (**`O`** = 0), то тут записується **`Y`**–координата. Якщо лінія вертикальна (**`O`** = 1), то тут записується **`X`**–координата.
`S`	`long`	Стиль лінії розриву, визначуваний системною змінною **`swBrealkLineStyle_e`** [10]: 0 – прямий виріз; 1 – кривий виріз; 2 – лінія розриву; 3 – маленька лінія розриву.

Для відображення розірваного виду використовується метод **BreakView**, що не має аргументів.

Приклад

```
boolstatus = Part.Extension.SelectByID2"Чертежный вид3",
"DRAWINGVIEW", 0, 0, 0, False, 0, Nothing, 0)
                          'вибір базового виду
Set myView = Part.SelectionManager.GetSelectedObject5(1)
                         отримання обраного об'єкта
SetmyBreakLine = myView.InsertBreak(0, -0.01, 0.013, 1)
                          'побудова ліній розриву
Part.BreakView            'відображення розірваного виду
```

9.1.9. Обрізаний вид

Послідовність створення лінії розриву на основі базового виду за моделлю наведено на рис 9.6.

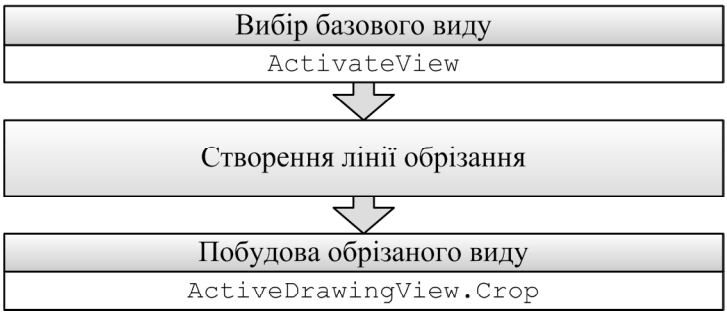


Рис. 9.6. Послідовність створення обрізаного виду

Синтаксис

```
longstatus = Part.ActiveDrawingView.Crop
```

Приклад

```
boolstatus = Part.ActivateView("Чертежный вид4")
                          'вибір базового виду
Set skSegment = Part.SketchManager.CreateCircle(0#,
0.016, 0#, 0.0021, 0.02, 0#)
                          'побудова границі обрізаного
                          'виду
longstatus = Part.ActiveDrawingView.Crop
                          'побудова обрізаного виду
```

9.1.10. Елементи моделі

Послідовність перенесення елементів твердотільної моделі (розміри, спеціальні позначення, осьові лінії, тощо) на креслення не передбачає ніяких попередніх дій.

Синтаксис

```
retval = DrawingDoc.InsertModelAnnotations3 ( o,
t, aV, dD, hFD, uPIS)
```

Аргументи методу **InsertModelAnnotations3** наведено у табл. 9.5.

Таблиця 9.5

Аргументи методу InsertModelAnnotations3

Аргумент	Тип	Призначення
o	long	Параметри відображення розмірів: 0 – усі розміри моделі; 1 – усі розміри обраної деталі (для складань); 2 – усі розміри обраного елементу деталі; 3 – усі розміри у складанні.
t	long	Тип приміток, що переносяться з моделі на креслення, визначається системною змінною **swInsertAnnotation_e** [10].
aV	boolean	True – відобразити усі примітки в усіх видах креслення; False – відобразити примітки тільки в обраних видах.
dD	boolean	True – усувати дублюючі розміри; False – не усувати їх.
hFD	boolean	True – додавати розміри погашених елементів моделі; False – не додавати їх.
uPIS	boolean	True – додавати розміри так, як вони розташовані на ескізі; False – не розташовувати їх таким чином.

Приклад

```
Annotations = Part.InsertModelAnnotations3(0, 1048575,
True, True, False, False)
                          'перенесення усіх елементів на
                          'креслення з моделі
```

9.2. Примітки

9.2.1. Побудова примітки

Послідовність створення примітки на аркуші креслення наведено на рис 9.7.

Рис. 9.7. Послідовність створення примітки

Для задавання тексту примітки використовується метод **InsertNote**.

Синтаксис

retval = Part.InsertNote ("текст примітки")

де:

текст примітки – текст примітки або текстова змінна, що містить текст.

Текст примітки може містити багато рядків, але весь текст записується як одна рядкова змінна. Для візуального поділу тексту примітки на рядки на аркуші в кінці додаються службові символи завершення рядка **Chr(13)** і **Chr(10)**.

Приклад

```
Set myNote = Part.InsertNote("Це перший рядок тексту
примітки"_
         + Chr(13) + Chr(10) + _
       "це другий рядок тексту примітки"_
          + Chr(13) + Chr(10) + _
"це останній рядок тексту примітки")
```

На аркуші такий текст виглядатиме так:

```
Це перший рядок тексту примітки
це другий рядок тексту примітки
це останній рядок тексту примітки
```

Шрифт тексту примітки оформлюється згідно з параметрами, встановленими у властивостях документа (**Параметры→ Свойства документа**). Для зміни оформлення шрифту у тексті розміщуються службові примітки – *теги*. Теги беруться у спеціальні дужки **< >**.

Основні теги для оформлення тексту примітки наведено у табл. 9.6.

Таблиця 9.6

Теги форматування тексту примітки

Призначення	Тег	Скасування дії тегу
Жирний шрифт	**<FONT style=B>**	**<FONT style=RB>**
Курсив	**<FONT style=I>**	**<FONT style=RI>**
Виділення кольором	**<FONT color=0x** номер кольору**>**	**<FONT color=D>**
Підкреслення	**<FONT effect=U>**	**<FONT effect=RU>**
Перекреслення	**<FONT effect=S>**	**<FONT effect=RS>**

Синтаксис тегу для задавання шрифту, що відрізняється від шрифту за замовчуванням

<FONT name=""назва шрифту**"">**

Синтаксис тегу для задавання розміру шрифту, що відрізняється від розміру за замовчуванням

<FONT size = значення розміру **PTS>**

Текст примітки може містити спеціальні символи і дроби. Приклади тегів для деяких спеціальних символів:

– **<MOD-DEG>** градус °;
– **<MOD-DIAM>** діаметр ∅;
– **<MOD-PM>** знак плюс-мінус ±.

Синтаксис тегу для запису дробу

<STACK size=1>текст чисельника**<OVER>**текст знаменника **</STACK>**

Вираз **size =** 1 означає, що розмір шрифту дробу буде таким, як у основного тексту. Якщо цифра менша за **1**, то розмір шрифту дробу буде відповідно меншим, якщо цифра більша за **1** – то більшим.

Текст примітки може містити нумеровані та не нумеровані списки.

Синтаксис тегу для задавання нумерованого списку:

```
<PARA indent=10 findent=-10 number=on ntype=1
nformat=$$. nstartNum=0>
      "текст пункта списку" + Chr(13) + Chr(10) + _
      "текст пункта списку" + Chr(13) + Chr(10) + _
      . . . . . .
<PARA indent=0 findent=0 number=off> завершення
списку
```

Синтаксис тегу для задавання ненумерованого списку:

```
<PARA indent=10 findent=-10 bullet=on>
    "текст пункта списку" + Chr(13) + Chr(10) + _
    "текст пункта списку" + Chr(13) + Chr(10) + _
    . . . . . .
<PARA indent=0 findent=0 bullet=off>завершення списку
```

Для створення рамки навколо тексту примітки використовується метод **SetBalloon**.

Синтаксис

```
retval = Note.SetBalloon (style, size)
```

де:

style – стиль рамки: 0 – без рамки; 1 – коло; 2 – трикутник; 3 – шестикутник; 4 – прямокутник; 5 – ромб; 6 – п'ятикутник; 8 – стрілка; 9 – горизонтальний трикутник; 11 – квадрат; 12 – квадрат із вписаним колом.

size – розмір рамки: 0 – по контуру тексту; 1 – один символ; 2 – два символи; 3 – три символи; 4 – чотири символи; 5 – п'ять символів.

Для створення стрілки і полички для примітки використовується метод **SetLeader3**.

Синтаксис

```
Retval = Annotation.SetLeader3 (LS, LeS, SAHS, P, AA, D)
```

Аргументи методу **SetLeader3** наведено у табл. 9.7.

Таблиця 9.7

Аргументи методу SetLeader3

Аргумент	Тип	Призначення
LS	long	Стиль полички 0 – тільки стрілка; 1 – стрілка і поличка; 3 – стрілка і підкреслення.
LeS	long	Розташування полички: 0 – автоматичне розташування; 1 – ліворуч; 2 – праворуч.
SAHS	boolean	True – автоматична установка стрілки, False – без стрілки.
P	boolean	True – перпендикулярна лінія до полички, False – без лінії.
AA	boolean	True – відображення усіх символів, False – без відображення.
D	boolean	True – пунктирна лінія полички; False – суцільна лінія полички.

Для визначення позиції примітки використовується метод **SetPosition**.

Синтаксис

```
retval = Annotation.SetPosition ( x, y, z)
```

де:

x, **y**, **z** – координати позиції примітки.

Для визначення формату тексту примітки використовується метод **SetTextFormat**.

Синтаксис

```
retval = Annotation.SetTextFormat ( i, uD, tF)
```

де:

i – номер, що вказує, який текст у даній примітці потрібно форматувати.

uD – аргумент логічного типу; `True` – використання установок формату тексту за замовчуванням, `False` – параметри формату тексту визначаються окремо.

tF – вказівник на змінну, що містить інформацію про формат тексту.

Приклад

```
DimNoteAsObject,Annotation As Object
Dim TextFormat As Object  'оголошення змінних
Set Note = Part.InsertNote("Примітка")
                          'текст примітки
Note.Angle = 0'кут нахилу примітки
boolstatus = Note.SetBalloon(0, 0)
                          'рамка навколо тексту примітки
Set Annotation = Note.GetAnnotation()
                          'створення об'єкта Annotation
longstatus = Annotation.SetLeader2(False, 0, True,
False, False, False)      'покажчик з поличкою
boolstatus = Annotation.SetPosition(0.1, 0.02, 0)
                          'позиція примітки
boolstatus = Annotation.SetTextFormat(0, True, TextFormat)
                          'шрифт примітки
```

9.2.2. Шорсткість

Послідовність створення позначки шорсткості аркуші креслення наведено на рис 9.8.

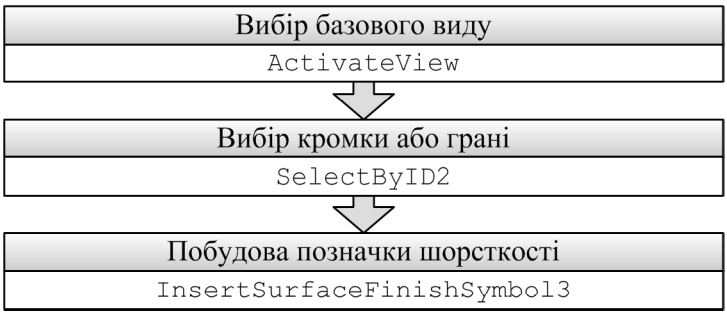


Рис. 9.8. Послідовність створення позначки шорсткості

Синтаксис

```
lpSfSymbolDisp = ModelDocExtension.
InsertSurfaceFinishSymbol3 ( sT, lT, X, Y, Z, lS,
aT, mA, oV, pM, sL, mR, mR, rS)
```

Аргументи методу **InsertSurfaceFinishSymbol3** наведено у табл. 9.8.

Таблиця 9.8

Аргументи методу InsertSurfaceFinishSymbol3

Аргумент	Тип	Призначення
sT	long	Тип символу, визначений системною змінною **swSFSymType_e** [10].
lT	long	Тип пилочки, визначений системною змінною **swLeaderStyle_e** [10]: 0 – без виноски; 1 – пряма виноска; 2 – виноска із зігнутим покажчиком; 3 – підкреслене.
X,Y,Z	double	Координати розташування символу.
lS	long	Напрям символу, визначений системною змінною **swSFLaySym_e [10]**: 0 – немає; 1 – круговий; 2 – перетинний; 3 – багато напрямів; 4 – паралельність; 5 – перпендикулярність; 6 – радіальний; 7 – макрочастка.
aT	long	Тип стрілки, визначений системною змінною **swArrowStyle_e** [10],
mA	string	Видалення матеріалу.
oV	string	Інші позначення шорсткості.
pM	string	Метод оброблення.
sL	string	Довжина.
mR	string	Максимальна шорсткість.
nR	string	Мінімальна шорсткість.
rS	string	Інтервал шорсткості.

Примітка. Шорсткість може бути встановлено без прив'язки до об'єкта креслення. У цьому разі не передбачено ніяких попередніх дій.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
0.12, 0.23, 0.003, False, 0, Nothing, 0)
                          'вибір кромки моделі для якої
                          'задається шорсткість
Set mySFSymbol = Part.Extension.InsertSurfaceFinish-
Symbol3(0, 0, 0.12, 0.23, 0.0035, 0, 7, "", "", "",
"", "", "", "")           'задавання шорсткості
```

9.2.3. Штрихування

Послідовність побудови штрихування замкненої області на кресленні не передбачає ніяких попередніх дій.

Синтаксис

```
Retval = SketchManager.CreateRegionHatch ( X, Y,
Z, A, S, C, H, L)
```

Аргументи методу **CreateRegionHatch** наведено у табл. 9.9.

Таблиця 9.9

Аргументи методу CreateRegionHatch

Аргумент	Тип	Призначення
X,Y,Z	double	Координати точки, що знаходиться у замкненій області, яку потрібно заштрихувати.
A	double	Кут нахилу ліній штрихування.
S	double	Масштаб штрихування.
C	long	Колір штрихування.
H	string	Ім'я штрихування з файлу штрихувань **Sldwks.ptn**.
L	string	Ім'я шару штрихування.

Приклад

```
Set SkHatch = Part.SketchManager.CreateRegionHatch(
0.11, 0.18, 0.34, 0, 1, 0, "ANSI31 (ЖелезоКирпич)", "")
                          'штрихування замкненої області
```

9.2.4. Таблиця

Послідовність створення таблиці на аркуші креслення наведено на рис 9.9.

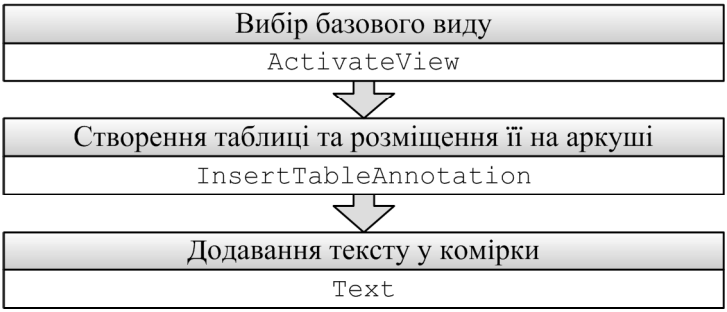


Рис. 9.9. Послідовність створення таблиці

Синтаксис

```
Retval = InsertTableAnnotation( X,Y, AT,R, C)
```

Аргументи методу **InsertTableAnnotation** наведено у табл. 9.10.

Таблиця 9.10

Аргументи методу InsertTableAnnotation

Аргумент	Тип	Призначення
X,Y	double	Координати розташування точки прив'язки таблиці на аркуші.
AT	integer	Точка прив'язки таблиці: 0 – верхній лівий кут; 1 – верхній правий кут; 2 – нижній лівий кут; 3 – нижній правий кут.
R	integer	Кількість рядків.
C	integer	Кількість стовпців.

Для запису тексту у комірки таблиці використовується метод **Text**.

Синтаксис

```
retval = Table.Text(R, C) = "текст"
```

де:

R – номер рядка, де розташовано комірку;
C – номер стовпця, де розташовано комірку;
текст – текст, що заноситься до комірки

Приклад

```
Set MyTable = Part.InsertTableAnnotation(0.06, 0.10,
1, 2, 2)                    'додавання на лист таблиці з
                            'двома рядками і двома
                            'стовпцями
MyTable.Text(0, 0) = "Перший стовпець"
                            'додавання надпису у комірку,
                            'яка розташована у першому
                            'рядку першого стовпця
MyTable.Text(0, 1) = "Другий стовпець"
                            'додавання надпису у комірку,
                            'яка розташована у першому
                            'рядку другого стовпця
```

9.3. Приклад розробки підсистеми побудови креслення кришки підшипникового вузла

Розглянемо проектну об'єктну підсистему для побудови креслення на основі моделі кришки глухої підшипникового вузла для підшипника із зовнішнім діаметром 80 мм згідно з ГОСТ 18511–73, побудованої раніше (див. п. 8.6).

Процедура не має форми користувача. Креслення виконується на аркуші форматом А3 і містить 2 види (див. рис 9.10): вид збоку та вид спереду. Причому вид збоку є видом–розрізом. Крім видів деталі на аркуші розташовуються невказані граничні відхилення та невказана шорсткість.

Нажаль, редагувати положення розмірів на аркуші креслення за допомогою макросу неможливо. Тому при створенні розмірів у контексті ескізу для твердотільної моделі їх положення слід визначати з урахуванням їх майбутнього розташування на аркуші креслення.

Головна програма.

```
Dim swApp As Object, myView As Object, Part As Object
Dim vSkLines As Variant, boolstatus As Boolean
Dim longstatus As Long, longwarnings As Long
Dim vAnnotations As Variant, Note As Object
Dim Annotation As Object, TextFormat As Object
Dim mySFSymbol As Object 'оголошення змінних
Sub main()               'оголошення головної програми
Set swApp = Application.SldWorks
                         'активація SolidWorks
```

```
Set Part = swApp.NewDocument( "C:\ProgramData\Solid-
Works\SolidWorks 2011\templates\Чертеж.drwdot", 12,
0.2159, 0.2794)          'створення документа креслення
                         'за шаблоном Чертеж.prtdot
Set Part = swApp.ActiveDoc
                         'активація документа
boolstatus = Part.SetupSheet5("Лист1", 12, 12, 1, 1,
True, "a3 - gost.slddrt", 0.42, 0.2794, "По умолчанию",
True)                    'встановлення шаблону основного
                         'надпису для аркуша А3 ГОСТ з
                         'файлу a3-gost.slddrt,
                         'визначеного користувачем.
Set myView = Part.CreateDrawViewFromModelView3("C:\
Крышка80.SLDPRT", "*Спереди", 0.127, 0.202, 0)
                         'створення виду Спереди з моделі
Set myView = Part.CreateDrawViewFromModelView3("C:\
Крышка80.SLDPRT", "*Снизу", 0.126, 0.143, 0)
                         'створення виду Снизу з моделі
Part.ClearSelection2 True
                         'скасування усіх виділень
boolstatus = Part.ActivateView("Чертежный вид1")
                         'активувати вид на кресленні 1
vSkLines = Part.SketchManager.CreateCornerRectan-
gle(-0.07, 0.02, 0, 0.07, -0.024, 0)
                         'створення прямокутника
Part.CreateBreakOutSection 0.06
                         'створення вириву деталі
Part.ClearSelection2 True
                         'скасування усіх виділень
boolstatus = Part.Extension.SelectByID2("Лист1",
"SHEET", 0, 0, 0, False, 0, Nothing, 0)
                         'виділення аркуша 1
vAnnotations = Part.InsertModelAnnotations3 (0,
15728639, True, True, False, True)
                         'перенесення приміток з
                         'твердотільної моделі у
                         'креслення
'- Побудова надпису про не вказані граничні відхилення -
Set Note = Part.InsertNote("<FONT name=""GOST type
A"">1. Невказані граничні відхилення H14, h14, <MOD-
PM><STACK size=1>IT14<OVER>2</STACK>")
                         'надпис примітки про не вказані
                         'граничні відхилення. Причому
                         'надпис виконується шрифтом
                         'GOSTtypeA. У надписі
                         'використовується спеціальні
                         'теги: спецзнак плюс-мінус
                         '<MOD-PM> та дріб, що
```

```
                        'позначається тегами розмітки
                        '<STACK size=1> знаменник
                        '<OVER> чисельник </STACK>
Note.Angle = 0          'кут нахилу примітки
boolstatus = Note.SetBalloon(0, 0)
                        'рамка навколо тексту примітки
Set Annotation = Note.GetAnnotation()
                        'створення об'єкта Annotation
longstatus = Annotation.SetLeader2(False, 0, True,
False, False, False)    'покажчик з поличкою
boolstatus = Annotation.SetPosition(0.24, 0.08, 0)
                        'позиція примітки
boolstatus = Annotation.SetTextFormat(0, True, TextFormat)
                        'шрифт примітки
    '- Побудова позначення не вказаної шорсткості -
Set mySFSymbol=Part.Extension.InsertSurfaceFinishSymbol3(
0, 0, 0.38, 0.26, 0, 0, 1, "", "", "", "", "", "", "")
                        'символ шорсткості
Set myAnnotation = mySFSymbol.GetAnnotation()
                        'створення об'єкта
boolstatus = myAnnotation.SetPosition(0.38, 0.26, 0)
                        'розташування знака шорсткості
                        'на аркуші
Set mySFSymbol = Part.Extension.InsertSurfaceFinishSymbol3(
2, 0, 0.36, 0.26, 0, 0, 1, "", "", "", "", "", "", "")
                        'символ шорсткості без зняття
                        'матеріалу
Set myAnnotation = mySFSymbol.GetAnnotation()
                        'створення об'єкта
boolstatus = myAnnotation.SetPosition(0.36, 0.26, 0)
                        'розташування знака шорсткості
                        'на аркуші
Set myNote = Part.InsertNote("<FONT size =48 PTS
style=RI>(  )")         'дужки позначки невказаної
                        'шорсткості. У надписі
                        'використовується спеціальна
                        'розмітка: розмір шрифту
                        'size = 48PTS, шрифт звичайний
                        '(не курсив) style = RI
myNote.Angle = 0        'кут нахилу примітки
boolstatus = myNote.SetBalloon(0, 0)
                        'рамка навколо тексту примітки
Set myAnnotation = myNote.GetAnnotation()
                        'створення об'єкта Annotation
longstatus = myAnnotation.SetLeader3(False, 0, True,
False, False, False)    'покажчик з поличкою
boolstatus = myAnnotation.SetPosition(0.38, 0.28, 0)
                        'позиція примітки
```

```
boolstatus = Annotation.SetTextFormat(0, True, TextFormat)
                                    'шрифт примітки
End Sub                             'завершення головної програми
```

Приклад роботи проектної процедури наведено на рис 9.10.

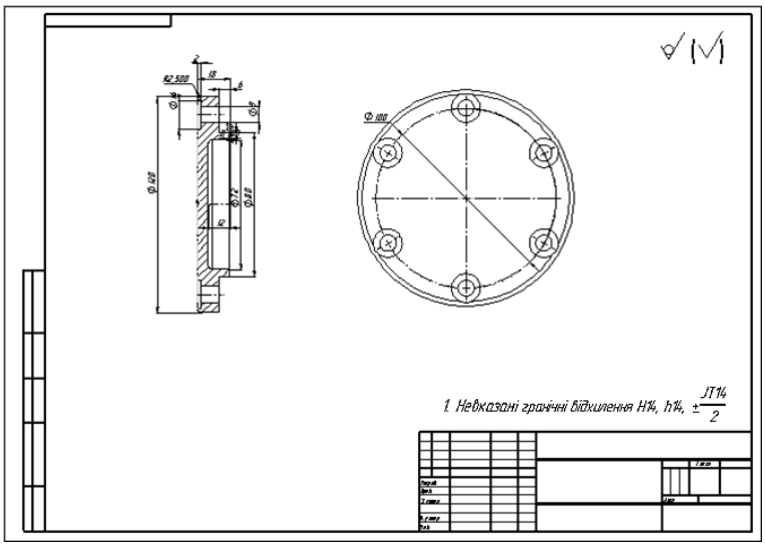


Рис. 9.10. Результат роботи проектної процедури

Контрольні запитання і завдання

1. Яка послідовність дій та які методи використовуються для побудови виду за моделлю на кресленні?
2. Яка послідовність дій та які методи використовуються для побудови проекційного виду на кресленні?
3. Назвіть послідовність дій та методи, що використовуються для побудови розрізу на кресленні.
4. Яка послідовність дій та які методи використовуються для побудови вирівняного розрізу?
5. Назвіть послідовність дій та методи, що використовуються для побудови місцевого виду на кресленні.
6. Яка послідовність дій та які методи використовуються для побудови трьох стандартних видів за моделлю на кресленні?

7. Назвіть послідовність дій та методи, що використовуються для побудови вириву деталі на кресленні.
8. Яка послідовність дій та які методи використовуються для побудови лінії розриву на кресленні?
9. Яка послідовність дій та які методи використовуються для побудови обрізаного виду на кресленні?
10. Назвіть послідовність дій та методи, що використовуються для переносу елементів моделі на креслення.
11. Яка послідовність дій та які методи використовуються для побудови текстової примітки?
12. Назвіть послідовність дій та методи, що використовуються для побудови покажчика шорсткості.
13. Яка послідовність дій та які методи використовуються для побудови штрихування?
14. Яка послідовність дій та які методи використовуються для побудови таблиці?
15. Побудуйте блок-схему проектної процедури для створення креслення кришки глухої підшипникового вузла.
16. Модифікуйте проектну процедуру для побудови твердотільних моделей косинок (див. п. 6.7), додавши до неї можливість побудови креслень.

10. Основні методи побудови складань

10.1. Додавання компонентів до складання

Послідовність додавання компонентів у складання наведено на рис. 10.1.

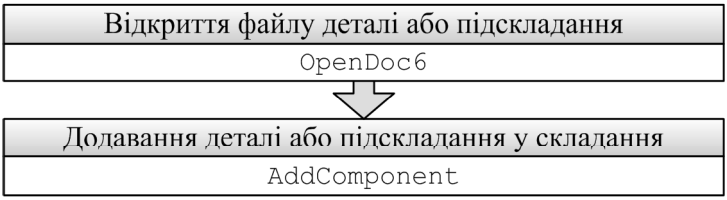


Рис. 10.1. Послідовність створення складання

Особливістю додавання компоненту до складання є необхідність попереднього відкриття файлу цього компоненту.

Синтаксис

```
result = DragOperator.AddComponent (CN,X,Y,Z)
```

де

CN – повний шлях та ім'я файлу деталі або підскладання, що додається.

X,Y,Z – координати точки центру деталі або складання, що додається.

Приклад

```
Set Part = swApp.OpenDoc6("C:\Деталь.prtdot", 1, 0, "",
longstatus, longwarnings) 'відкриття наявного документа
                          'Деталь.prtdot
Part.AddComponent ("D:\bolt.SLDPRT", 0, 0.08, 0.217)
                          'додати деталь bolt.SLDPRT до
                          'складання
```

У складанні одна з деталей (головна) має бути зафіксованою. Тому за замовчуванням перша додана у складання деталь фіксується. Для фіксування/звільнення компонентів використовується метод **FixComponent**. Метод не має аргументів.

Послідовність фіксування/звільнення компонентів у складанні наведено на рис. 10.2.

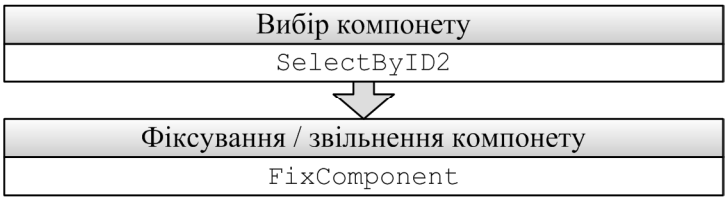


Рис. 10.2. Послідовність фіксування/звільнення компонентів

10.2. Спряження

Послідовність створення спряження між компонентами складання наведено на рис. 10.2.

Рис. 10.3. Послідовність створення спряження

Синтаксис

```
PMateOut = Part.AddMate3 ( MTFE, AFE,F,D,DAUL,DAL
L,GRN,GRD,A,AAUL,AALL,FPO,ES)
```

Аргументи методу **AddMate3** наведено у табл. 10.1.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "FACE",
-0. 14, 0.5, 0.24, True, 1, Nothing, 0)
                        'обрати грань одного
                        'компонента для спряження
boolstatus  =  Part.Extension.SelectByID2("",  "FACE",
0.74, -0.29, 0.1, True, 1, Nothing, 0)
                        'обрати грань другого
                        'компонента для спряження
Set Feature = Part.AddMate3(3, 0, False, 0.04, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'побудувати спряження
                        'Параллельно
```

Таблиця 10.1

Аргументи методу `AddMate3`

Аргумент	Тип	Призначення
`MTFE`	`integer`	Типи спряжень визначені системною змінною **`swMateType_e`** [10]: 0 – збіг; 1 – концентрично; 2 – перпендикулярно; 3 – паралельно; 4 – дотично; 5 – на відстані; 6 – під кутом; 7 – невідомий; 8 – симетрично; 9 – кулачкове; 10 – редуктор; 11 – ширина; 12 – прив'язати до ескізу; 13 – шестерня / рейка; 15 – шлях; 16 – заблокувати; 17 – гвинт; 18 – лінійний / лінійна муфта; 19 – універсальний шарнір; 20 – координата; 21 – проріз; 22 – шарнір; 23 – повзун; 24 – центр профілів.
`AFE`	`long`	Тип вирівнювання визначений системною змінною **`swMateAlign_e`** [10]: 0 – вирівняти; 1 – переставити вирівнювання; 2 – близькі.
`F`	`boolean`	`True` – повернути компонент; `False` – не повертати.
`D`	`double`	Відстань, що використовується для задавання спряжень.
`DAUL`	`double`	Абсолютне максимальне значення відстані.
`DALL`	`double`	Абсолютне мінімальне значення відстані.
`GRN`	`double`	Значення підвищення передаточного відношення для спряження **Редуктор**.
`GRD`	`double`	Значення зменшення передаточного відношення для спряження **Редуктор**.
`A`	`double`	Кут, що використовується для задавання спряжень.
`AAUL`	`double`	Абсолютне максимальне значення кута.
`AALL`	`double`	Абсолютне мінімальне значення кута.
`FPO`	`boolean`	`True` – тільки розташувати компоненти відповідно до спряження, але не створювати спряження; `False` – створити спряження.
`ES`	`long`	Код помилки при створенні спряження визначається системною змінною **`swAddMateError_e`** [10]..

10.3. Масиви компонентів

10.3.1. Лінійний масив компонентів

Послідовність створення лінійного масиву компонентів у складанні наведено на рис. 10.4.

Рис. 10.4. Послідовність створення лінійного масиву компонентів

Синтаксис:

```
result = FeatureManager.FeatureLinearPattern2(
n1, s1, n2, s2, fD1, fD2, dN1, dN2, gP)
```

Аргументи методу **FeatureLinearPattern2** наведено у табл. 10.2.

Таблиця 10.2

Аргументи методу FeatureLinearPattern2

Аргумент	Тип	Призначення
n1	`long`	Кількість екземплярів лінійного масиву в напряму 1, в тому числі оригіналу.
s1	`double`	Крок між екземплярами лінійного масиву в напряму 1 у метрах.
n2	`long`	Кількість екземплярів лінійного масиву в напряму 2, в тому числі оригіналу.
s2	`double`	Крок між екземплярами лінійного масиву в напряму 2 у метрах.
fD1	`boolean`	`True` – змінити напрям 1; `False` – не змінювати.
fD2	`boolean`	`True` – змінити напрям 2; `False` – не змінювати.
dN1	`string`	Ім'я напряму 1.
dN2	`string`	Ім'я напряму 2.
gP	`boolean`	`True` – використати геометричний шаблон; `False` – не використовувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("", "EDGE",
-0.5, 0.29, -0.31, False, 2, Nothing, 0)
                          'вибір кромки для напряму X
boolstatus = Part.Extension.SelectByID2("", "EDGE",
-0.36, 0.15, -0.31, True, 4, Nothing, 0)
                          'вибір кромки для напряму Y
boolstatus = Part.Extension.SelectByID2("shaiba-1@
Сборка1", "COMPONENT", 0, 0, 0, True, 1, Nothing, 0)
                          'вибір об'єкта для копіювання
Set Feature = Part.FeatureManager.FeatureLinearPattern2(2,
0, 1, 2, 0, 1, True, False, "NULL", "NULL", False)
                          'побудова масиву
```

10.3.2. Круговий масив компонентів

Послідовність створення кругового масиву компонентів у складанні наведено на рис. 10.5.

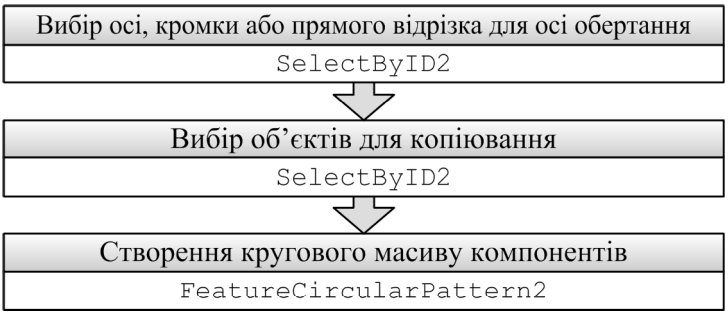


Рис. 10.5. Послідовність створення кругового масиву компонентів

Синтаксис:

```
result = FeatureManager.FeatureCircularPattern2 (
n, s, fD, dN, gP)
```

Аргументи методу **FeatureCircularPattern2** наведено у табл. 10.3.

Таблиця 10.3

Аргументи методу `FeatureCircularPattern2`

Аргумент	Тип	Призначення
n	`long`	Кількість екземплярів кругового масиву, включно з базовим.
s	`double`	Крок між екземплярами кругового масиву у радіанах.
fD	`boolean`	`True` – змінити напрям; `False` – не змінювати напрям.
dN	`string`	Ім'я напряму.
gP	`boolean`	`True` – використати геометричний шаблон; `False` – не використовувати.

Приклад

```
boolstatus = Part.Extension.SelectByID2("Line3@Эскиз1",
"EXTSKETCHSEGMENT", 0, 0, 0, False, 2, Nothing, 0)
                          'вибір прямого відрізка в
                          'ескізі як осі обертання
boolstatus = Part.Extension.SelectByID2("shaiba-1@
Сборка1", "COMPONENT", 0, 0, 0, True, 1, Nothing, 0)
                          'вибір об'єкта для копіювання
Set Feature = Part.FeatureManager.FeatureCircularPat-
tern2(5, 1, 256637061436#, False, "NULL", False)
                          'побудова масиву
```

10.4. Приклад розроблення підсистеми побудови складання ланцюга роликового

Розглянемо проектну об'єктну підсистему для побудови складання ланцюга роликового із вигнутими пластинами типу ПРИ 78,1 – 40000 згідно з ГОСТ 13568–97 або ISO 3512 (рис. 10.6).

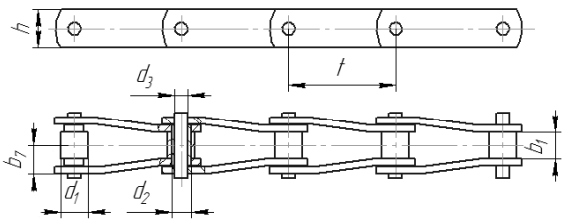


Рис. 10.6. Схема ланцюга ПРИ 78,1–40000

Параметри ланцюга ПРИ 78,1–40000: $d_1 = 40$ мм; $d_2 = 28$ мм; $d_3 = 19$ мм; $b_1 = 38{,}1$ мм; $b_7 = 51$ мм; $h = 56$ мм; $t = 78{,}1$ мм. Інші розміри визначаються виробником.

Складання виконується з 4 деталей ланцюга, побудованих раніше (рис. 10.7), а саме: з осі (рис. 10.7, *а*), втулки (рис. 10.7, *б*), ролика (рис. 10.7, *в*) та пластини. (рис. 10.7, *г*).

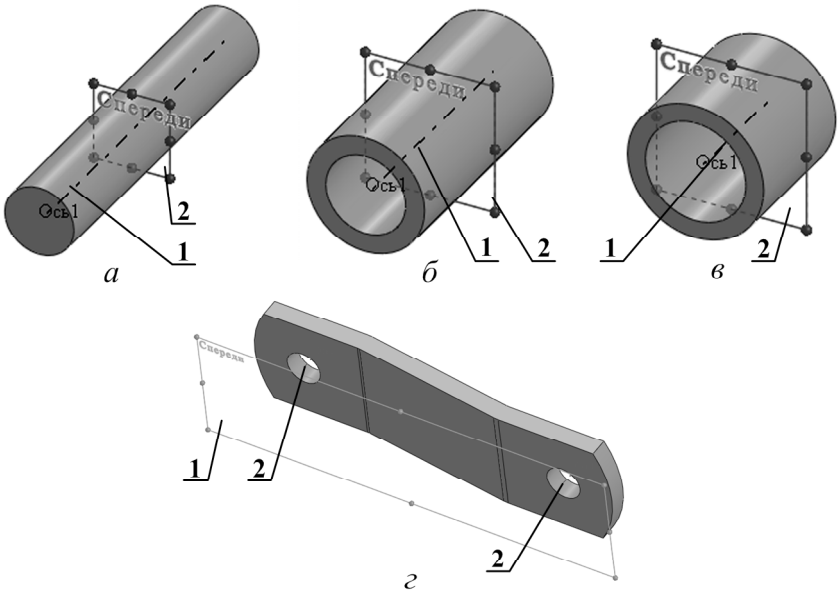


Рис. 10.7. Деталі складання ланцюга ПРИ–78,1–40000: *а* – вісь; *б* – втулка; *в* – ролик; *г* – пластина; *1* – площина, що проходить по середині ланцюга; *2* – довідкова вісь

Під час побудови складання при задаванні спряжень потрібно обирати грані деталей. Оскільки ці грані не мають власних імен, то обираються за точками розташованими на них, що є не зручним. Для спрощення встановлення спряжень на деталях будуються довідкові осі, що збігаються з осями обертання цих деталей.

Вихідні дані, що задаються користувачем – кількість ланок ланцюга.

Проектування форми користувача. На формі користувача розташовуються елементи управління згідно з рис. 10.8.

Рис. 10.8. Форма користувача та елементи управління: *1* – надпис **Label1**; *2* – поле для введення **TextBox1**; *3* – кнопка **CommandButton1**; *4* – кнопка **CommandButton2**

При розробленні форми користувача у менеджері властивостей встановлюються такі параметри:

– для форми **UserForm1**:

```
UserForm1.Caption = "Пластинчастий ланцюг"
```

– для кнопки **CommandButton1**:

```
CommandButton1.Caption = "Побудувати";
```

обробник події **CommandButton1_Click()**;

– для кнопки **CommandButton2**;

```
CommandButton2.Caption = "Вихід";
```

обробник події **CommandButton2_Click()**.

Проектна підсистема містить головну програму та 2 обробники подій форми та елементів керування:

– обробник події натиснення кнопки **CommandButton1** для побудови складання;

– обробник події натиснення кнопки **CommandButton2** для завершення роботи.

Головна програма

```
Sub main()
   User Form1.Show      'запуск форми користувача
End Sub
```

Обробник події натиснення кнопки **CommandButton1** для побудови складання ланцюга.

```
Private Sub CommandButton1_Click()
Dim swApp As Object, Part As Object
Dim boolstatus As Boolean, myMate As Object
Dim longstatus As Long, longwarnings As Long
Dim nErrors As Long, nWarnings As Long
Dim AT As String, N As Integer
                          'оголошення змінних
Set swApp = Application.SldWorks
                          'активація SolidWorks
```

```
Set Part = swApp.OpenDoc6("D:\Ось.SLDPRT", 1, 0, "",
longstatus, longwarnings)
                        'відкриття деталі Ось.SLDPRT
Set Part = swApp.OpenDoc6("D:\Втулка.SLDPRT", 1, 0,
"", longstatus, longwarnings)
                        'відкриття деталі Втулка.SLDPRT
Set Part = swApp.OpenDoc6("D:\Ролик.SLDPRT", 1, 0, "",
longstatus, longwarnings)
                        'відкриття деталі Ролик.SLDPRT
Set Part = swApp.OpenDoc6("D:\Пластина.SLDPRT", 1, 0,
"", longstatus, longwarnings)
                        'відкриття деталі Пластина.SLDPRT
Set Part = swApp.NewDocument("C:\ProgramData\
SolidWorks\SolidWorks 2011\templates\Сборка.asmdot",
0, 0, 0)                'створення документа складання
                        'з шаблону Сборка.asmdot
Set Part = swApp.ActiveDoc
                        'активація документа
AT = Part.GetTitle      'визначення назви складання,
                        'автоматично наданої документу
                        'системою
boolstatus = Part.AddComponent("D:\Ось.SLDPRT", 0, 0, 0)
                        'додавання деталі Ось у
                        'складання
boolstatus = Part.AddComponent("D:\Втулка.SLDPRT",0,0,0)
                        'додавання деталі Втулка у
                        'складання
boolstatus = Part.AddComponent("D:\Ролик.SLDPRT",0,0,0)
                        'додавання деталі Ролик у
                        'складання
boolstatus = Part.AddComponent("D:\Пластина.SLDPRT",0,0,0)
                        'додавання деталі Пластина у
                        'складання
boolstatus = Part.Extension.SelectByID2("Спереди@
Втулка-1@" &AT, "PLANE",0,0,0, False, 1, Nothing, 0)
                        'вибір площини
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
Set myMate = Part.AddMate3(0, 0, False, 0, 0, 0, 0, 0,
0, 0, 0, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Ось1@Ось-1@"
&AT, "AXIS", 0, 0, 0, False, 1, Nothing, 0)
                        'вибір осі
```

```
boolstatus = Part.Extension.SelectByID2("Ось1@Втулка-1@"
&AT, "AXIS", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір осі
Set myMate = Part.AddMate3(0, 0, False, 0.036, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, False, 1, Nothing, 0)
                        'вибір площини
boolstatus = Part.Extension.SelectByID2("Спереди@
Ролик-1@" &AT, "PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
Set myMate = Part.AddMate3(0, 0, False, 0.0031, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Ось1@Ролик-1@"
&AT, "AXIS", 0, 0, 0, False, 1, Nothing, 0)
                        'вибір осі
boolstatus = Part.Extension.SelectByID2("Ось1@Ось-1@"
&AT, "AXIS", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір осі
Set myMate = Part.AddMate3(0, 0, False, 0.03, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
boolstatus = Part.Extension.SelectByID2("Спереди@
Пластина-1@" &AT, "PLANE", 0, 0, 0, True, 1,Nothing,0)
                        'вибір площини
Set myMate = Part.AddMate3(0, 0, False, 0.016, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Ось1@
Пластина-1@" &AT, "AXIS", 0, 0, 0, True, 1, Nothing,0)
                        'вибір осі
```

```
boolstatus = Part.Extension.SelectByID2("Ось1@Ось-1@"
&AT, "AXIS", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір осі
Set myMate = Part.AddMate3(0, 1, False, 0.062, 0, 0,
0.001, 0.001, 0, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Сверху@
Пластина-1@" &AT, "PLANE", 0, 0, 0,False, 1,Nothing,0)
                        'вибір площини
boolstatus = Part.Extension.SelectByID2("Сверху",
"PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
Set myMate = Part.AddMate3(0, 0, False, 0, 0, 0,
0.001, 0.001, 0.1, 0.52, 0.52, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.AddComponent("D:\Пластина.SLDPRT",
0, 0, 0)                'додавання деталі Пластина у
                        'складання
boolstatus = Part.Extension.SelectByID2("Ось1@Ось-1@"
&AT, "AXIS", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір осі
boolstatus = Part.Extension.SelectByID2("Ось1@
Пластина-2@" &AT, "AXIS", 0, 0, 0, True, 1, Nothing,0)
                        'вибір осі
Set myMate = Part.AddMate3(0, 0, False, 0.0042, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Спереди@
Пластина-2@" &AT, "PLANE", 0, 0, 0, True, 1,Nothing,0)
                        'вибір площини
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
Set myMate = Part.AddMate3(0, 1, False, 0, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
```

```
boolstatus = Part.Extension.SelectByID2("Ось2@
Пластина-2@" &AT, "AXIS", 0, 0, 0, False, 1,Nothing,0)
                        'вибір осі
boolstatus = Part.Extension.SelectByID2("Ось2@
Пластина-1@" &AT, "AXIS", 0, 0, 0, True, 1, Nothing,0)
                        'вибір осі
Set myMate = Part.AddMate3(0, 1, False, 0, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                        'встановлення спряження
                        'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
Part.SaveAs "d:\Zveno.sldasm"
                        'збереження підскладання ланки у
                        'файл Zveno.sldasm
N = Val(TextBox1.Text)  'завантаження довжини ланцюга з
                        'елементу керування у змінну
If N> 1 Then            'якщо довжина ланцюга більше за
                        '1, то..
Set Part = swApp.NewDocument("C:\ProgramData\
SolidWorks\ SolidWorks 2011\templates\Сборка.asmdot",
0, 0, 0)                'створення документа складання з
                        'шаблону Сборка.asmdot
Set Part = swApp.ActiveDoc
                        'активація документа
AT = Part.GetTitle      'визначення назви складання,
                        'автоматично наданої документу
                        'системою
boolstatus = Part.AddComponent("d:\Zveno.sldasm", 0,0,0)
                        'додавання підскладання
                        'Zveno.sldasm у складання
For a = 1 To N- 1       'організація циклу додавання
                        'ланок у складання
boolstatus = Part.AddComponent("d:\Zveno.sldasm", a *
(156.2 / 1000), 0, 0)   'додавання підскладання
                        'Zveno.sldasm у складання
Next a                  'завершення циклу
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3       'перебудова моделі
For a = 2 To N          'організація циклу додавання
                        'спряжень між доданими ланками
boolstatus = Part.Extension.SelectByID2("Спереди@
Zveno-" & Trim(Str(a)) + "@" + AT, "PLANE", 0, 0, 0,
False, 1, Nothing, 0)   'вибір площини
boolstatus = Part.Extension.SelectByID2("Спереди",
"PLANE", 0, 0, 0, True, 1, Nothing, 0)
                        'вибір площини
```

```
Set myMate = Part.AddMate3(0, 0, False, 0, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                          'встановлення спряження
                          'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3         'перебудова моделі
boolstatus = Part.Extension.SelectByID2(("Ось1@Zveno-" &
Trim(Str(a)) + "@" + AT + "/Пластина-1@" &AT, "AXIS", 0,
0, 0, False, 1, Nothing, 0)
                           'вибір осі
boolstatus = Part.Extension.SelectByID2(("Ось2@Zveno-" &
Trim(Str(a - 1)) + "@" + AT + "/Ось-1@" &AT, "AXIS", 0,
0, 0, True, 1, Nothing, 0)
                          'вибір осі
Set myMate = Part.AddMate3(0, 1, False, 0, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                          'встановлення спряження
                          'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3         'перебудова моделі
boolstatus = Part.Extension.SelectByID2("Ось1@Zveno-"
&Trim(Str(a)) + "@" + AT + "/Ось-1@Zveno", "AXIS", 0,
0, 0, True, 1, Nothing, 0)
                          'вибір осі
boolstatus = Part.Extension.SelectByID2("Ось2@Zveno-"
& Trim(Str(a - 1)) + "@" + AT + "/Пластина-1@Zveno",
"AXIS", 0, 0, 0, True, 1, Nothing, 0)
    'вибір осі
Set myMate = Part.AddMate3(0, 1, False, 0, 0, 0,
0.001, 0.001, 0, 0, 0, False, longstatus)
                          'встановлення спряження
                          'Совпадение
Part.ClearSelection2 True 'скасування усіх виділень
Part.EditRebuild3         'перебудова моделі
Next a                    'завершення циклу
Part.SaveAs "d:\Cep.sldasm"
                          'збереження підскладання ланцюга
                          'у файл Cep.sldasm
End If                    'завершення блоку вибору
End Sub                   'завершення процедури
```

Обробник події натиснення кнопки **CommandButton** завершення роботи макросу

```
Private Sub CommandButton2_Click()
   Unload UserForm1       'закриття форми користувача
End Sub                   'завершення процедури
```

Приклад роботи проектної процедури наведено на рис. 10.9.

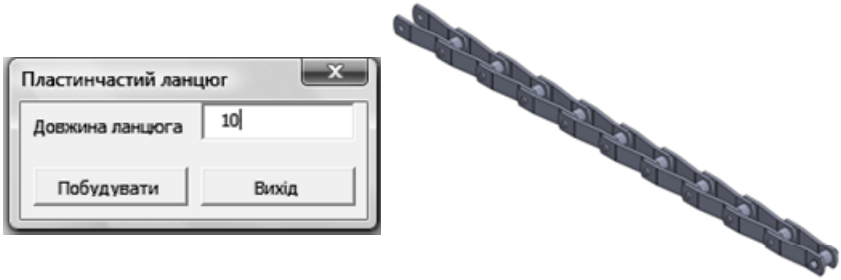


Рис. 10.9. Приклад робота процедури

Контрольні запитання і завдання

1. Яка послідовність дій та які методи використовуються для додавання деталей або підскладань у складання?
2. Яка існує особливість при додаванні деталі або підскладання у складання?
3. Назвіть особливості фіксування та звільнення компонентів у складаннях.
4. Назвіть послідовність дій та методи, що використовуються для встановлення спряжень.
5. Яка послідовність дій та які методи використовуються для побудови масиву лінійних компонентів?
6. Назвіть послідовність дій та методи, що використовуються для побудови масиву кругових компонентів.
7. Дослідіть, як спряжені деталі у складанні ланцюга у п. 10.4.
8. Створіть блок-схему обробника події створення складання ланцюга з п.10.4.
9. Створіть макрос для побудови складання кулькового підшипника кочення.

11. Методи оптимізації

Згідно зі схемою програмного забезпечення проектної процедури САПР (рис. 1.1), основним розрахунковим блоком є блок оптимізації (*4*). У ньому за встановленими вихідними даними та обмеженнями знаходиться оптимальне проектне рішення. Розглянемо основні завдання оптимізації конструкції, параметрів проектованих виробів та методи для їх розв'язання [2, 7].

11.1. Основні поняття і визначення

Оптимізація – знаходження екстремуму (мінімуму або максимуму) цільової функції в деякій області, яка може бути обмежена набором лінійних або нелінійних рівнянь або нерівностей.

З точки зору інженерних розрахунків, оптимізація – це процес вибору найкращого варіанта конструкції або найкращий розподіл ресурсів з усіх можливих.

Проектні параметри – параметри, оптимальні значення яких необхідно знайти в процесі розв'язання інженерного завдання оптимізації. Як проектні параметри можуть виступати, значення лінійних розмірів деталі, маса, температура та ін.

Розмірність завдання оптимізації – кількість n проектних параметрів $x_1, x_2,..., x_n$, що характеризує ступінь складності завдання оптимізації.

Цільова функція або ***критерій оптимальності*** – глобальний критерій оптимальності в математичних моделях, за допомогою якого описується інженерне завдання. Прикладами цільової функції, що трапляються в інженерних розрахунках, є міцність, маса конструкції, потужність установки тощо.

У разі одного проектного параметра ($n = 1$) цільова функція є функцією однієї змінної, а її графік – деяка крива на площині. При $n = 2$ цільова функція є функцією двох змінних, і її графік – поверхня в тривимірному просторі. При трьох і більше проектних параметрах поверхні, що задаються цільовою функцією, називаються *гіперповерхнями* і не піддаються зображенню звичайними засобами.

Цільова функція може бути представлена у вигляді формули або може набувати тільки деяких значень і задаватися у вигляді таблиці значень.

Цільових функцій може бути декілька. Наприклад, при проектуванні виробів машинобудування одночасно вимагається забезпечити максимальну надійність, мінімальну матеріаломісткість, максимальний корисний об'єм (чи вантажопідйомність). Деякі цільові функції можуть виявитися несумісними. У таких випадках необхідно вводити пріоритет тієї або іншої цільової функції – «функцію компромісу», що дозволяє в процесі оптимізації користуватися однією складеною цільовою функцією.

11.2. Постановка завдання оптимізації

Оптимізаційне завдання – завдання з визначення найкращих (у деякому розумінні) структури або значень параметрів об'єктів.

У процесі розв'язання завдання оптимізації мають бути встановлені такі значення проектних параметрів, при яких цільова функція має мінімум (чи максимум).

Оптимізація може бути:

– *параметричною*, коли визначаються оптимальні значення параметрів при завданій структурі об'єкта;
– *структурною*, коли обирається оптимальна структура об'єкта.

У досить загальному вигляді математичну задачу оптимізації можна сформулювати таким чином: мінімізувати (максимізувати) цільову функцію за наявності обмежень або без обмежень на керовані змінні.

Для запису математичних задач оптимізації в загальному вигляді використовується така символіка

$$f(x) \to \text{extr}\,(\max/\min),\ x \in \text{U},$$

де $f(x)$ – цільова функція; U – допустима множина, задана обмеженнями на керовані змінні.

Завдання оптимізації можна поділити на два типи:

– *безумовне завдання оптимізації* полягає у відшукуванні максимуму або мінімуму цільової функції на усій області проектування, тобто визначення глобального екстремуму;
– *умовне завдання оптимізації* (завдання з обмеженнями) – завдання, при формулюванні якого є деякі умови (обмеження) щодо проектних параметрів, які задаються сукупністю рівнянь або нерівностей.

Розв'язання оптимізаційного завдання має такі етапи:
- аналіз ситуації та формулювання завдання, визначення меж системи оптимізації для спрощення розв'язання;
- визначення параметрів розв'язання, що підлягають оптимізації, вибір керованих і постійних змінних;
- встановлення обмежень на керовані змінні у вигляді рівностей або нерівностей;
- вибирання критеріїв оптимальності (цільових функцій);
- побудова математичних моделей цільових функцій;
- вибирання математичного метода оптимізації;
- проведення розрахунків та оцінювання отриманих рішень за обраними критеріями;
- остаточне прийняття розв'язання з урахуванням невизначеності та ризиків.

11.3. Класифікація методів оптимізації

Методи оптимізації – методи пошуку екстремуму функції (у практичних задачах – критеріїв оптимальності) за наявності обмежень або без обмежень [2].

Методи оптимізації дозволяють виконувати оптимальне проектування (вибір найкращих номінальних технологічних режимів, елементів конструкцій, структури технологічних ланцюжків, умов економічної діяльності, підвищення доходності та ін.), оптимальне управління побудовою моделей об'єктів управління (мінімізації нев'язок різної структури моделі і реального об'єкту) і багато інших аспектів рішення економічних і соціальних проблем (наприклад, управління запасами, трудовими ресурсами, транспортними потоками та ін.).

Існує багато класів задач оптимізації і, відповідно, методів їх рішення.

Методи оптимізації класифікують таким чином:

1) ***за областю знаходження оптимуму:***
   - методи локальної оптимізації, що забезпечують відшукування одного локального мінімуму;
   - методи глобальної оптимізації, які спрямовані на встановлення усіх локальних мінімумів або найкращого з них;
2) ***за кількістю проектних параметрів:***
   - методи одновимірної оптимізації для цільової функції з одним проектним параметром;

– методи багатовимірної оптимізації для цільової функції з декількома проектними параметрами;

3) ***за кількістю цільових функцій:***
   – методи однокритерійної оптимізації, спрямовані на пошук оптимуму єдиної цільової функції;
   – методи багатокритерійної оптимізації, що забезпечують ухвалення рішення при декількох цільових функціях;
4) ***за видом цільової функції й обмежень:***
   – цільова функція й обмеження є лінійними функціями, розв'язуються методами лінійного програмування;
   – цільова функція й обмеження є нелінійними функціями, розв'язуються методами нелінійного програмування;
5) ***за ступенем математичної обґрунтованості:***
   – раціональні алгоритми зорієнтовані на деяку математичну модель функції, що оптимізується, і зазвичай мають строгі докази збіжності до стаціонарної точки й оцінки швидкості збіжності;
   – евристичні алгоритми засновані на формалізованій людській інтуїції та інших нестрогих, але розумних припущеннях;
6) ***за використанням похідних:***
   – методи прямого пошуку (методи нульового порядку), яким потрібні тільки значення цільової функції, тобто функція задана в табличному вигляді або може бути обчислена при деяких дискретних значеннях аргументу;
   – методи першого порядку (градієнтні методи), які використовують часткові похідні першого порядку цільової функції;
   – методи другого порядку (ньютонівські методи) використовують часткові похідні другого порядку.

Наведемо назви найбільш поширених методів оптимізації:

1) ***одновимірні методи:***
   метод золотого перерізу; дихотомія; метод парабол; перебір по сітці; метод Фібоначчі; трійковий пошук; метод Піявського; метод Стронгіна;
2) ***прямі методи:***
   метод Гауса; метод Нелдера – Міда; метод Хука – Дживса; метод конфігурацій; метод Розенброка;
3) ***методи першого порядку:***
   градієнтний спуск; метод Зойтендейка; покоординатний спуск; метод зв'язаних градієнтів; квазіньютонівські методи; алгоритм Левенберга – Марквардта;

4) ***методи другого порядку:***
метод Ньютона; метод Ньютона – Рафсона; алгоритм Бройдена – Флетчера – Гольдфарба – Шанно (BFGS);
5) ***стохастичні методи:***
метод Монте–Карло; імітація відпалу; еволюційні алгоритми; диференціальна еволюція; мурашиний алгоритм; метод рою часток; алгоритм бджолиної колонії;
6) ***методи лінійного програмування:***
симплекс–метод; алгоритм Гоморі; метод еліпсоїдів; метод потенціалів;
7) ***методи нелінійного програмування:***
послідовне квадратичне програмування.
8) ***методи багатокритерійної оптимізації:***
метод аналізу ієрархій, метод ефективних множин, метод уступок, метод головного критерію, метод згортки, метод цільового програмування.

11.4. Одновимірна оптимізація

11.4.1. Задачі на екстремум функції

Простим методом безумовної оптимізації (знаходження глобального екстремуму) є випадок функції *f(x)*, що задана у вигляді аналітичної залежності $y = f(x)$ і має явний вираз для своєї похідної [2]. Необхідною умовою екстремуму є нульове значення її похідної

$$\frac{\partial}{\partial x} f(x) = 0 . \qquad (11.1)$$

Приклад

Нехай після проведення досліджень встановлено залежність напруження σ у матеріалі деталі від деякого геометричного розміру L і зовнішнього навантаження F:

$$\sigma(L, F) = 1,25 + 0,038 \cdot F - 0,84 \cdot L + 0,0054 \cdot F^2 + 0,018 \cdot L^2 + 0,35 \cdot F \cdot L .$$

Визначимо таке значення геометричного розміру L при якому напруження у матеріалі деталі σ прагне до мінімуму.

Зауважимо, що при проектуванні значення зовнішнього навантаження F конструктор змінити не може. Ця змінна далі вважатиметься константою. Отже, це є завданням одновимірної оптимізації.

Згідно з (11.1) визначимо першу похідну наведеної вище залежності відносно розміру L

$$\frac{\partial}{\partial L}\sigma = 0,35 \cdot F + 0,036 \cdot L - 0,84 .$$

Складемо рівняння для пошуку екстремуму, прирівнявши цю похідну до 0

$$0,35 \cdot F + 0,036 \cdot L - 0,84 = 0 .$$

Після розв'язання рівняння отримаємо залежність для визначення геометричного розміру L, при якому напруження у матеріалі деталі σ буде мінімальним залежно від значення зовнішнього навантаження F

$$L = 23,3 - 9,72 \cdot F .$$

11.4.2. Метод золотого перерізу

З-поміж безлічі методів одновимірної оптимізації нульового порядку розглянемо один з найбільш простих і ефективних методів.

Метод золотого перерізу є прямим методом оптимізації і застосовується, коли аналітичної залежності для цільової функції немає, а є лише можливість визначення її значень у довільних точках даної області за допомогою деякого обчислювального алгоритму або фізичних вимірів [2].

Метод золотого перерізу належить до чисельних методів, для яких можна гарантувати, що необхідну точність буде досягнуто.

Розглянемо цей метод на прикладі знаходження мінімуму функції $f(x)$ на відрізку $[a, b]$. Припустимо, що цільова функція має один екстремум на цьому відрізку, тобто є *унімодальною функцією*.

Похибка наближеного розв'язання задачі визначається різницею між оптимальним значенням x проектного параметра і наближення до нього x_*.

Встановимо вимогу, щоб ця погрішність була за модулем менша заданого припустимого значення ε

$$\left|x - x_*\right| < \varepsilon . \qquad (11.2)$$

Процес розв'язання задачі пошуковими методами полягає у послідовному звуженні інтервалу зміни проектного параметра, що називається *інтервалом невизначеності*. На початку процесу оптимізації його довжина дорівнює α, а наприкінці вона повинна стати меншою за ε, тобто оптимальне значення проектного параметра повинне знаходитися в інтервалі невизначеності – на відрізку $[x_n, x_{n+1}]$, причому $x_{n+1} - x_n < \varepsilon$.

Метод золотого перерізу полягає в побудові послідовності відрізків $[a_0, b_0]$, $[a_1, b_1]$, ... , $[a_n, b_n]$, що стягуються до точки екстремуму функції $f(x)$. На кожному кроці, за винятком першого, обчислення значення функції $f(x)$ проводиться лише в одній точці. Ця точка називається *золотим перерізом* та обирається спеціальним чином.

***Алгоритм методу золотого перерізу*:**

1. Задаються початкові межі відрізку a, b і точність обчислення ε;

2. Заданий відрізок ділиться двома точками поділу, які симетричні відносно його центру (рис 11.1)

$$x_1 = b - \frac{b-a}{1{,}618}, \quad x_2 = a + \frac{b-a}{1{,}618},$$

де 1,618 – пропорція золотого перерізу.

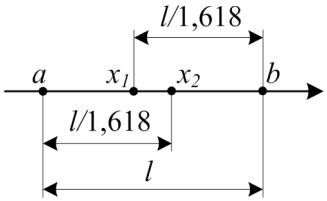


Рис 11.1. Точки золотого перерізу

Далі розраховуються значення цільової функції в цих точках

$$y_1 = f(x_1), y_2 = f(x_2).$$

Якщо $y_1 \geq y_2$ (для пошуку мінімуму), або $y_1 \leq y_2$ (для пошуку максимуму), то $a = x_1$, інакше $b = x_2$;

3. Процедура триває до тих пір, поки не буде досягнуто заданої точності ε.

Якщо $|b-a|<\varepsilon$, то $x=\dfrac{a+b}{2}$ і завершуємо роботу. Інакше повернення до кроку 2.

Приклад

Нехай після проведення досліджень встановлено залежність напруження σ у матеріалі деталі від деякого геометричного розміру L і зовнішнього навантаження F (рис 11.2)

$$\sigma(L,F)=10-L^{0,9}\cdot e^{-0,7\times L}\cdot F^{0,9}\cdot e^{-0,3\times F}.$$

Визначимо таке значення геометричного розміру L, при якому напруження у матеріалі деталі σ прагне до мінімуму.

Зауважимо, що при проектуванні значення зовнішнього навантаження F конструктор змінити не може. Ця змінна далі вважатиметься константою. Тому завдання зводиться до завдання одновимірної оптимізації.

Знайти розв'язок задачі на екстремум за допомогою похідної для представленої функції важко. Використаємо для пошуку мінімуму функції метод золотого перерізу.

Створимо функцію для визначення мінімуму згідно з наведеним вище алгоритмом.

```
Function MGS(a As Double, b As Double, e As Double, F
As Double) As Variant   'аргументи функції: значення
                        'початкових меж відрізку a, b,
                        'точність e та значення
                        'зовнішнього навантаження F.
                        'Функція повертає вектор зі
                        'значеннями x_min, f(x_min)
Dim Res(1) As Double    'оголошення вектора для
                        'повернення значень
Do                      'початок циклу
 x1 = b-(b-a)/1.618     'визначення першої точки поділу
 x2 = a+(b-a)/1.618     'визначення другої точки поділу
 y1 = 10-x1^0.9*Exp(-0.7*x1)*F^0.9*Exp(-0.3*F)
                        'визначення значення функції у
                        'першій точці поділу
 y2 = 10-x2^0.9*Exp(-0.7*x2)*F^0.9*Exp(-0.3*F)
                        'визначення значення функції у
                        'другій точці поділу
```

```
 If y1 >= y2 Then a = x1 Else b = x2
                          'визначення відрізка з меншим
                          'значенням функції
Loop Until Abs(b - a)< e
                          'умова завершення оптимізації
x = (a + b) / 2           'точка мінімального значення
                          'функції
Res (0) = x               'завантаження точки
                          'мінімального значення функції
                          'у вектор результатів
Res (1) = 10-x^0.9*Exp(-0.7*x)*F^0.9*Exp(-0.3*F)
                          'завантаження мінімального
                          'значення функції у вектор
                          'результатів
MGS = Res                 'повернення функції значення
End Function              'завершення функції
```

Приклад роботи функції

```
Dim V (1) As double
V = MGS (0.5,6,0.001,0.1)
```

У результаті отримано вектор V, що містить координати точки мінімуму функції L_{min} = 1,294, σ_{min} = 9,938 (рис 11.2).

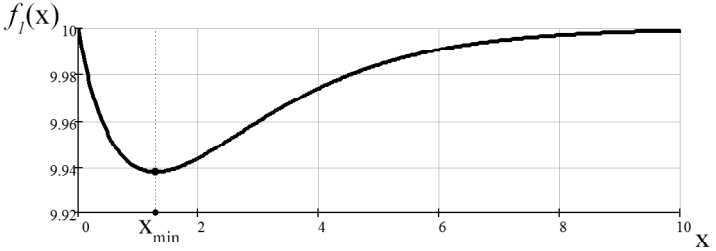


Рис 11.2. Графік залежності з указанням значення мінімуму

11.5. Багатовимірна оптимізація

11.5.1. Мінімум функції декількох змінних

Мінімум функції багатьох змінних $u=f(x_1, x_2, \ldots, x_n)$, що диференціюється, можна знайти, досліджуючи її значення у критичних точках, які визначаються розв'язанням системи рівнянь з частинних похідних [2]

$$\begin{cases} \dfrac{\partial}{\partial x_1} f(x_1, x_2, \ldots x_n) = 0, \\ \dfrac{\partial}{\partial x_2} f(x_1, x_2, \ldots x_n) = 0, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \dfrac{\partial}{\partial x_n} f(x_1, x_2, \ldots x_n) = 0. \end{cases} \qquad (11.3)$$

Приклад

Нехай після проведення досліджень встановлено залежність напруження σ у матеріалі деталі від деяких геометричних розмірів L_1, L_2 і зовнішнього навантаження F

$$\sigma\left(L_1, L_2, F\right) = 0,038 \cdot L_1 + 0,0054 \cdot L_1^2 + 0,018 \cdot L_2^2 + 0,35 \cdot F \cdot L_1 \cdot L_2 .$$

Визначимо таке значення геометричних розмірів L_1, L_2, при яких напруження у матеріалі деталі σ буде мінімальним.

Зауважимо, що при проектуванні значення зовнішнього навантаження F конструктор змінити не може. Ця змінна далі вважатиметься константою.

Згідно з (11.3), визначимо частинні похідні наведеної вище залежності відносно розмірів L_1 і L_2

$$\begin{cases} \dfrac{\partial}{\partial L_1} \sigma(L_1, L_2) = 0,0108 \cdot L_1 + 0,35 \cdot F \cdot L_2 + 0,038 ; \\ \dfrac{\partial}{\partial L_2} \sigma(L_1, L_2) = 0,036 \cdot L_2 + 0,35 \cdot F \cdot L_1 . \end{cases}$$

Складемо систему рівнянь, прирівнявши ці похідні до 0

$$\begin{cases} 0,0108 \cdot L_1 + 0,35 \cdot F \cdot L_2 + 0,038 = 0 ; \\ 0,036 \cdot L_2 + 0,35 \cdot F \cdot L_1 = 0 . \end{cases}$$

Після розв'язання системи рівнянь, отримаємо залежність для визначення значень геометричних розмірів L_1 і L_2, при яких напруження у матеріалі деталі σ буде мінімальним, залежно від значення зовнішнього навантаження F

$$\begin{cases} L_1 = -\dfrac{0,038}{0,0108 - 3,389 \cdot F^2}; \\ L_2 = \dfrac{0,369 \cdot F}{0,0108 - 3,389 \cdot F^2} \end{cases} .$$

11.5.2. Метод покоординатного спуску

Метод покоординатного спуску є одним з простіших методів нульового порядку для знаходження екстремуму функції декількох змінних [2]. Він спирається тільки на обчислення значень цієї функції і не використовує обчислення похідних.

Сутність методу така. Нехай потрібно знайти найменше значення цільової функції $u=f(x_1, x_2, \ldots, x_n)$. Як початкове наближення оберемо у n–мірному просторі деяку точку $M_0=f(x_1, x_2^{(0)}, \ldots, x_n^{(0)})$. Зафіксуємо усі координати функції, окрім першої. Тоді $u=f(x_1, x_2^{(0)}, \ldots, x_n^{(0)})$ – функція однієї змінної x_1. Розв'язуючи одновимірну задачу оптимізації для цієї функції, від точки M_0 перейдемо до точки $M_1=f(x_1^{(1)}, x_2^{(0)}, \ldots, x_n^{(0)})$, в якій функція і набуває найменшого значення по координаті x_1 при фіксованих інших координатах. У цьому спуску по координаті x_1 полягає перший крок процесу оптимізації.

Зафіксуємо тепер усі координати, окрім x_2, і розглянемо функцію цієї змінної $u=f(x_1^{(1)}, x_2^{(0)}, \ldots, x_n^{(0)})$. Знову вирішуючи одновимірну задачу оптимізації, знаходимо її найменше значення при $x_2 = x_2^{(1)}$, тобто у точці $M_2=f(x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(0)})$.

Аналогічно проводиться спуск по координатах $x_3, x_4, \ldots, x_n$, а потім процедура знову повторюється від x_1 до x_n. У результаті цього процесу виходить послідовність точок $M_0, M_1, \ldots, M_n$, в яких значення цільової функції складають монотонно спадаючу послідовність $f(M_0) \geq f(M_1) \geq \ldots \geq f(M_n)$. На будь-якому k–му кроці цей процес можна перервати, і значення $f(M_k)$ приймається як найменше значення цільової функції в даній області. На рис 11.3 наведено геометричну інтерпретацію методу покоординатного спуску.

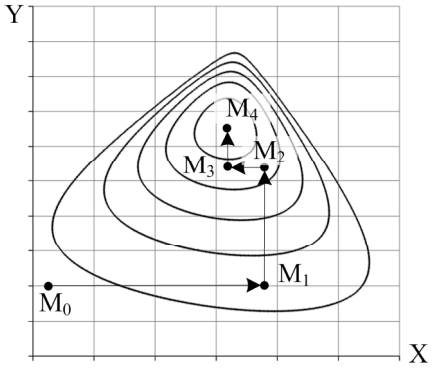


Рис 11.3. Геометрична інтерпретація методу покоординатного спуску для цільової функци $z = f(x, y)$

Таким чином, метод покоординатного спуску зводить задачу про знаходження екстремуму значення функції багатьох змінних до багатократного розв'язання одновимірних задач оптимізації з кожного з проектних параметрів.

Алгоритм методу покоординатного спуску:

1. Задати обмежувальну область визначення функції [a, b] для усіх аргументів та крок h, що забезпечує задану точність.
2. Почати змінювати перший аргумент, зберігаючи усі інші аргументи цільової функції як константи. Спочатку зробити крок праворуч і ліворуч від початкової точки і визначити знак перед h («+» чи «–»), тобто в яку сторону спадає (чи зростає) цільова функція, туди і рухати поточну точку.
3. У циклі пересунути поточну точку на величину кроку до тих пір, поки цільова функція не почне зростати (спадати) або ця координата не вийде за межі заданого діапазону.
4. Почати змінювати наступний аргумент та перейти до кроку 2.

При заданій обмежуючій області поточна точка обов'язково досягне або екстремуму, або межі області, оскільки знак кроку за цією координатою визначається тільки один раз і не змінюється до переривання циклу.

Приклад

Нехай після проведення досліджень встановлена залежність напруження σ у матеріалі деталі від деяких геометричних розмірів L_1 і L_2

$$\sigma(L_1, L_2) = (L_1 - 60)^2 + (0.5 \cdot L_2 - 70)^2.$$

Обмеженнями виступають області значень геометричних параметрів $L_1 \in [5, 40]$, $L_2 \in [10, 40]$. Точність розрахунку (крок) $\varepsilon = 0{,}5$.

Визначимо таке значення геометричних розмірів L_1 і L_2 при яких напруження σ у матеріалі деталі буде мінімальним.

Це є задача багатовимірної оптимізації. Скористаємося методом покоординатного спуску.

Згідно наведеного вище алгоритму створимо функцію, що реалізує метод покоординатного спуску.

```
Function KrSp(x0 As Double,xk As Double,y0 As
Double,yk As Double,e As Double) As Variant
                          'аргументи функції:область
                          'визначення функції за першим
                          'виміром x0 та xk, область
                          'визначення функції за другим
                          'виміром y0 та yk, точність e.
                          'Функція повертає вектор зі
                          'значеннями xmin, ymin, f(xmin, ymin)
Dim Res (2) As Double     'оголошення вектора для
                          'повернення значень
Zmin = 10000              'визначення первинного значення
                          'змінної мінімуму функції
X = x0                    'визначення додаткової змінної
Y = y0                    'визначення додаткової змінної
While X <= xk             'оголошення циклу руху по першій
                          'координатній осі
 Z = sqr(X - 60) + sqr(0.5 * Y - 70)
                          'визначення значення функції
 If Z < Zmin Then         'умова пошуку мінімуму функції
  X1 = X                  'запис координати мінімуму x
  Zmin = Z                'запис значення мінімуму функції
 Endif                    'кінець умовного блоку
 X = X + e                'пересування першої координати
                          'поточної точки на крок
Wend                      'завершення циклу
While Y <= yk             'оголошення циклу руху по другій
                          'координатній осі
 Z = sqr(X1 - 60) + sqr(0.5 * Y- 70)
                          'визначення значення функції
 If Z < Zmin Then         'умова пошуку мінімуму функції
  Y1 = X                  'запис координати мінімуму y
  Zmin = Z                'запис значення мінімуму функції
 End If                   'кінець умовного блоку
 Y = Y + e                'пересування другої координати
                          'поточної точки на крок
```

```
Wend                        'завершення циклу
Res(0) = X                  'завантаження точки X
                            'мінімального значення функції
                            'у вектор результатів
Res(1) = Y                  'завантаження точки Y
                            'мінімального значення функції
                            'у вектор результатів
Res(2) = Zmin               'завантаження  значення
                            'мінімуму функції
KrSp = Res                  'повернення функції значення
End Function                'завершення функції
```

Приклад роботи функції

```
Dim V (2) As double
V = KrSp (5,40,10,40,0.001)
```

У результаті отримано вектор V, що містить координати точки мінімуму функції $L_{1\,\min} = 40$, $L_{2\,\min} = 40$, $\sigma_{\min} = 29000$.

11.5.3. Метод сіток

Метод сіток, або метод повного перебору є не тільки універсальним і найпростішим з наближених методів за своєю ідеєю, але і самим ресурсоємним методом [2]. Причому цей метод можна використовувати як при безумовній, так і при умовній оптимізації (лінійному або нелінійному програмуванні).

Сутність методу така. Покриємо область визначення сіткою з кроком h (рис. 11.4) і визначимо значення функції в її вузлах. Порівнюючи отримані числа між собою, знайдемо з-поміж них найменше (або найбільше) і приймемо його приблизно за найменше значення функції для цієї області.

Метод сіток використовується в основному для розв'язання одновимірних і двовимірних задач. Для задач більшої розмірності він практично непридатний через тривалий час, необхідний для проведення розрахунків. Так для функції з 5–ти аргументів середній час розв'язання становитиме приблизно 10^5 секунд.

Приклад

Нехай після проведення досліджень встановлено залежність напруження σ у матеріалі деталі від деяких геометричних розмірів L_1 і L_2

$$\sigma(L_1, L_2) = 1 + L_1^{0,9} \cdot e^{-0,7 \cdot L_1} \cdot \left(-L_2^{0,9}\right) \cdot e^{-0,4 \cdot L_2}.$$

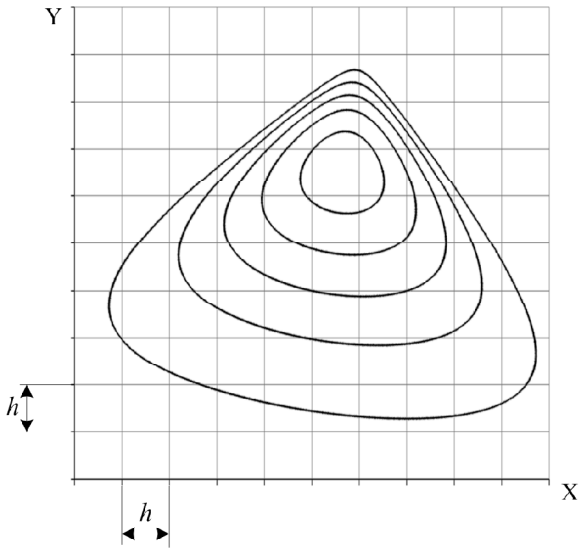


Рис.11.4. Покриття області сіткою з кроком *h*

Обмеженнями виступають області значень геометричних параметрів $L_1 \in [5, 40]$, $L_2 \in [10, 40]$. Точність розрахунку (крок) $\varepsilon = 0{,}5$.

Визначимо таке значення геометричних розмірів L_1 і L_2 при яких напруження σ у матеріалі деталі буде мінімальним.

Це є задача багатовимірної оптимізації. Скористаємося методом сіток.

Створимо функцію, що реалізує метод сіток.

```
Function LPr(L1p As Double, L1k As Double, L2p As
Double, L2k As Double, e As Double) As Variant
                        'аргументи функції:область
                        'визначення функції за першим
                        'виміром L1 п та L1 к, область
                        'визначення функції за другим
                        'виміром L2 п та L2 к, точність e.
                        'Функція повертає вектор зі
                        'значеннями L1 min, L2 min,
                        'σ (L1 min,L2 min)
Dim Res(2) As Double    'оголошення вектора для
                        'повернення значень
Zmin = 10000            'визначення первинного значення
                        'змінної мінімуму функції
X = L1p                 'визначення додаткової змінної
```

```
Y = L2p                    'визначення додаткової змінної
Nx = int(abs(L1p - L1k)/e)
                           'визначення кількості комірок
                           'сітки за першим аргументом
Ny = int(abs(L2p - L2k)/e)
                           'визначення кількості комірок
                           'сітки за першим аргументом
For a = 0 to Ny            'оголошення циклу руху по другій
                           'координатній осі
 For b = 0 to Nx           'оголошення циклу руху по першій
                           'координатній осі
  Z = 1 + X^0.9*exp(-0.7*X)*(-1)*Y^0.9*exp(-0.4*Y)
                           'визначення значення функції
  If Z < Zmin Then         'умова пошуку мінімуму функції
   Zmin = Z                'запис мінімального значення
   Xmin = X                'запис координати мінімуму x
   Ymin = Y                'запис координати мінімуму y
  End If                   'кінець умовного блоку
  X = X + e                'пересування першої координати
                           'поточної точки на крок
 Next b                    'кінець циклу b
 Y = Y + e                 'пересування другої координати
                           'поточної точки на крок
Next a                     'кінець циклу a
Res(0) = Xmin              'завантаження точки X
                           'мінімального значення функції
                           'у вектор результатів
Res(1) = Ymin              'завантаження точки Y
                           'мінімального значення функції
                           'у вектор результатів
Res(2) = Zmin              'завантаження  значення
                           'мінімуму функції
LPr = Res                  'повернення функції вектору
                           'результатів
End Function               'завершення функції
```

Приклад роботи функції

```
Dim V(2) As double
V = Lpr (1,40,1,40,0.01)
```

У результаті отримано вектор V, що містить координати точки мінімуму функції $L_{1\,min} = 1{,}29$, $L_{2\,min} = 1$, $\sigma_{min} = 0{,}658$.

11.6. Умовна оптимізація

Умовна оптимізація (оптимізація з обмеженнями) відрізняється від безумовної тим, що додатково до цільової функції

записуються функції обмежень, тобто відшукується не просто точка глобального екстремуму цільової функції, а точка екстремуму у середині області, заданої обмеженнями. Цим займається розділ математики, що називається математичне програмування [2].

Математичне програмування – це математична дисципліна, яка розробляє методи відшукування екстремальних значень цільової функції з-поміж безлічі її можливих значень, визначуваних обмеженнями.

Слід наголосити, що термін «програмування» немає відношення до терміна «програмування, як складання програми для ЕОМ», і виник у результаті неточного перекладу англійського «linear programming». Правильним перекладом цього терміна є «лінійне планування», оскільки одне зі значень слова «programming» – складання планів, планування. Таким чином, цей термін пов'язаний з поняттям оптимальної програми випуску продукції або оптимальної програми діяльності.

У теорії оптимізації під обмеженнями розуміють спеціальні функції, що містять комбінації змінних x. Ці функції записуються як рівність або як нерівності (строгі або нестрогі), які необхідно враховувати при визначенні точок екстремуму.

Залежно від властивостей цільової функції і функції обмежень усі задачі математичного програмування поділяються на два основні класи:

– задачі лінійного програмування, коли цільова функція і функції обмежень є лінійними функціями;
– задачі нелінійного програмування, коли хоч одна з вказаних функцій є нелінійною.

11.6.1. Лінійне програмування

Лінійне програмування – розділ математичного програмування, що вивчає завдання оптимізації, в яких цільова функція є лінійною функцією проектних параметрів, а обмеження задаються у вигляді лінійних рівнянь і нерівностей [2]. Воно застосовується для роботи з математичними моделями тих процесів і систем, в основу яких може бути покладено гіпотезу лінійного представлення реального світу.

Характерним для лінійного програмування є те, що оптимальною виявляється одна з крайніх (кутових) точок області допустимих розв'язків. При великій кількості обмежень виходить багато точок (тисячі, десятки і сотні тисяч). Тому завданнями лінійного програмування є мінімізація кількості обчислень. Для цього розроблено багато конкретних методів.

Стандартна (канонічна) постановка задачі лінійного програмування формулюється таким чином: знайти значення змінних, які

1) задовольняють системі лінійних рівнянь

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1, \\ &a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_n; \end{aligned} \tag{11.4}$$

2) є не невід'ємними

$$x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0; \tag{11.5}$$

3) забезпечують найменше (або найбільше) значення лінійної цільової функції

$$f(x_1, x_2, \ldots, x_n) = c_0 + c_1x_1 + c_2x_2 + \ldots + c_nx_n. \tag{11.6}$$

Будь-яке розв'язання системи рівнянь (11.4), що задовольняє системі нерівностей (11.5), називається *допустимим розв'язком*. Допустимий розв'язок, який мінімізує цільову функцію (11.6), називається *оптимальним розв'язком*.

Обмеження використовують для того, щоб зменшити зону пошуку екстремуму, для чого виділяють *область допустимих розв'язків*.

Областю розв'язків системи є перетин кінцевого числа напівплощин, що описуються кожною окремою нерівністю виду (11.5). Цей перетин є багатокутною областю допустимих рішень (рис 11.5). Вона може бути як обмеженою, так і необмеженою і навіть порожньою.

Можна виокремити три основні варіанти області допустимих розв'язків залежно від типу обмежень, які її визначають:

– ***обмеження суперечливі***, тобто $x_1, x_2, \ldots, x_n$ не можуть одночасно задовольнити усім заданим обмеженням, тому задача оптимізації не має розв'язання (рис 11.5, *а*). Область допус-

тимих розв'язків розірвана і складається з двох частин: одна область обмежена осями координат x_1 і x_2та прямими q_1 і q_2, друга – віссю координат x_1 і прямими q_1 і q_3. Очевидно, що шукана точка не може бути одночасно в цих двох областях;

– ***обмеження несуперечливі, але область допустимих розв'язків нескінченна*** (рис 11.5, *б*), тому завдання оптимізації також не має ні постановки, ні розв'язання. Область допустимих розв'язків відкрита (показана жирною ламаною лінією) і спрямована у нескінченність;

– ***обмеження несуперечливі і область допустимих розв'язків обмежена*** (рис 11.5, *в*), тому задача оптимізації має розв'язання. Область допустимих рішень є опуклим багатокутником.

Отже, оптимізація лінійної цільової функції на багатокут-

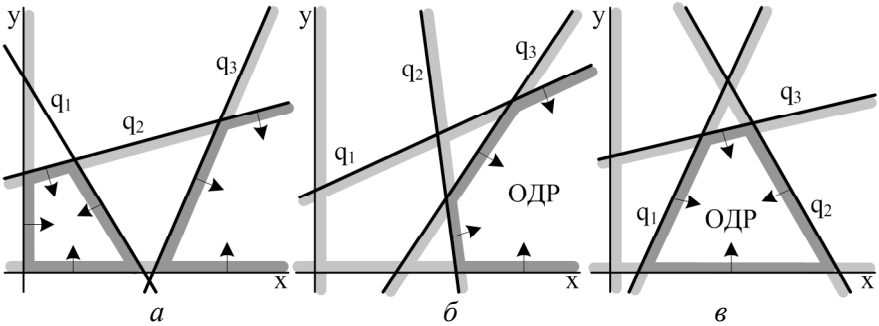


Рис 11.5. Типи області допустимих розв'язків: *а* – області допустимих розв'язків не існує; *б* – необмежена область допустимих розв'язків; *в* – обмежена область допустимих розв'язків

нику допустимих розв'язків відбувається в точках перетину цього багатокутника з опорними прямими, що відповідають цій цільовій функції. При цьому перетин може бути в одній точці (у вершині багатокутника) або в нескінченній безлічі точок (на ребрі багатокутника). В останньому випадку є нескінченна безліч оптимальних розв'язків.

Приклад

Розглянемо приклад задачі лінійного програмування цільової функції двох змінних з п'ятьма умовами.

Цільова функція

$$I(\overline{x}) = 5 \cdot x_1 + 4 \cdot x_2 \to \max,$$

Обмеження

$$\left.\begin{aligned} q_1(\overline{x}) &= 6 \cdot x_1 + 4 \cdot x_2 \le 24 \\ q_2(\overline{x}) &= x_1 + 2 \cdot x_2 \le 6; \\ q_3(\overline{x}) &= -x_1 + x_2 \le 1; \\ q_4(\overline{x}) &= x_2 \le 2; \\ x_1, x_2 &\ge 0. \end{aligned}\right\}$$

Геометричне представлення задачі (рис 11.6) містить у координатах x_1, x_2 лінії обмежень q_1, q_2, q_3, q_4, що обмежують область допустимих розв'язків ABCDEF. Ця область – опукла і рішення знаходиться в одній з точок: A, B, C, D, E або F.

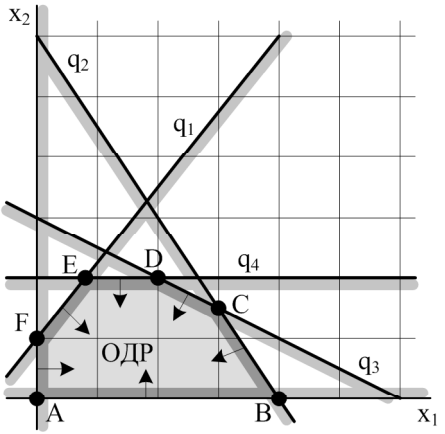


Рис. 11.6. Графічне представлення задачі оптимізації

Для розв'язання подібного типу задач існує низка методів, що дозволяють значно скоротити обсяг обчислень та підвищити точність результатів. Для розв'язання представленої задачі скористаємося найпростішим та універсальним методом сіток.

Створимо функцію для розв'язання поставленої задачі лінійного програмування.

```
Function LPrMesh (x1_0 As Double,x1_k As Double,x2_0
As Double,x2_k As Double,e As Double) As Variant
                        'аргументи функції:область
                        'визначення функції за першим
                        'виміром x1_0 та x1_k, область
                        'визначення функції за другим
                        'виміром x2_0 та x2_k, точність e.
                        'Функція повертає вектор зі
                        'значеннями x1 min, x2 min,
                        'I(x1 min,x2 min)
Dim Res(2) As Double    'оголошення вектора для
                        'повернення значень
Zmax = 0                'визначення первинного значення
                        'змінної максимуму функції
X1 = x1_0               'визначення додаткової змінної
X2 = x2_0               'визначення додаткової змінної
Nx1 = int(abs(x1_0 - x1_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
Nx2 = int(abs(x2_0 - x2_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
For a = 0 to Nx2        'оголошення циклу руху за
                        'другою координатною віссю
 For b = 0 to Nx1       'оголошення циклу руху за
                        'першою координатною віссю
  Z = 5*X1 + 4*X2       'визначення значення функції
  If (6*X1+4*X2<=24) and (X1+2*X2<=6) and
(-1*X1+X2<=1) and (X2<=2) Then
                        'обмеження згідно з умовою
   If Z > Zmax Then     'умова пошуку максимуму функції
    Zmax = Z            'максимальне значення функції
    X1max = X1          'значення аргументу x1 при
                        'максимумі функції
    X2max = X2          'значення аргументу x2 при
                        'максимумі функції
   End If               'кінець умовного блоку
  End If                'кінець умовного блоку
  X1 = X1 + e           'пересування першої координати
                        'поточної точки на крок
 Next b                 'кінець циклу b
 X2 = X2 + e            'пересування другої координати
                        'поточної точки на крок
Next a                  'кінець циклу a
Res(0) = X1min          'завантаження точки X1
                        'мінімального значення функції
                        'у вектор результатів
```

```
Res(1) = X2min          'завантаження точки X2
                        'мінімального значення функції
                        'у вектор результатів
Res(2) = Zmin           'завантаження значення
                        'мінімуму функції
LPrMesh = Res           'повернення функції вектору
                        'результатів
End Function            'завершення функції
```

Приклад роботи функції

```
Dim V(2) As Double
V = LPrMesh (0,10,0,2,0.01)
```

У результаті отримано вектор **V**, що містить координати точки мінімуму функції $x_{1\,min} = 0$, $x_{2\,min} = 0$, $I_{min} = 0$.

Розглянемо характерні завдання лінійного програмування.

Задача лінійного програмування про ресурси

Деяка організація виробляє продукцію двох видів: $П_1$ і $П_2$. При цьому використовуються три типи вихідної сировини: C_1, C_2 і C_3. Відомо, що для виробництва 1 тонни продукції $П_1$ потрібно: $a_{1,1}$ = 1 тонна сировини C_1, $a_{2,1}$ = 6 брикетів сировини C_2 і $a_{3,1}$ = 32 літри сировини C_3. Для виробництва однієї тонни продукції $П_2$ потрібно: $a_{1,2}$ = 2 тонни сировини C_1, $a_{2,2}$ = 3 брикети сировини C_2 і $a_{3,2}$ = 36 літрів сировини C_3. Обсяг використовуваної сировини обмежений (обмежені можливості складських приміщень): сировини C_1 може використовуватися не більше 20 тонн ($C_{1max} \le 20$), сировини C_2 – не більше 60 брикетів ($C_{2max} \le 60$), сировини C_3 – не більше 400 л ($C_{3max} \le 400$). При реалізації ціна 1 тонни продукції $П_1$ становить b_1 = 30000 грн., а продукції $П_2$ – b_2 = 20000 грн. Вся продукція продається без обмежень.

Потрібно визначити, скільки якої продукції слід виробити, щоб отримати найбільший прибуток.

Умови задачі можна записати у вигляді таблиці 11.1.

У таблиці змінними x_1 і x_2 позначені маси відповідних типів продукції (у тонах), які необхідно знайти у результаті розв'язання задачі.

Задача формулюється завжди таким чином, щоб всі інші параметри були ваговими коефіцієнтами, які множаться на ці змінні. Причому вагові коефіцієнти можуть мати різну фізичну природу і, відповідно, різні одиниці виміру.

Таблиця 11.1

Вагові коефіцієнти завдання лінійного програмування при виробництві продукції

№	Тип функції	Позначення функції	Вагові коефіцієнти на 1 тонну продукції		Гранична величина
			Продукція $П_1$: x_1	Продукція $П_2$: x_2	
1	Цільова функція, грн.	$I(x)$	$b_1 = 20000$	$b_2 = 30000$	max
2	Використання сировини C_1, тонни	$q_1(x)$	$a_{1,1} = 1$	$a_{1,2} = 2$	$C_{1\,max} \le 20$
3	Використання сировини C_2, брикети	$q_2(x)$	$a_{2,1} = 6$	$a_{2,2} = 3$	$C_{2\,max} \le 60$
4	Використання сировини C_3, літри	$q_3(x)$	$a_{3,1} = 32$	$a_{3,2} = 36$	$C_{3\,max} \le 400$

Наприклад, коефіцієнти у виразі q_1 вимірюються у тоннах, у виразі q_2 – у брикетах, а у виразі q_3 – у літрах. Розмірність змінних x_1 і x_2 також може бути різною, але добуток вагового коефіцієнта на змінну повинен призводити до однієї і тієї ж фізичної величини в одних і тих же одиницях виміру. Це пов'язано з тим, що добутки вагових коефіцієнтів на змінні підсумовуються, а підсумовувати можна тільки однорідні складові.

Загальний прибуток при продажу обох типів продукції, який слід максимізувати

$$I(x) = b_1 \cdot x_1 + b_2 \cdot x_2 = 30000 \cdot x_1 + 20000 \cdot x_2 \to \max, \quad (11.7)$$

З фізичної суті задачі випливає, що x_1, x_2 повинні бути невід'ємними, тобто більше або дорівнювати нулю. Негативність x_1 і x_2 означала б, що продукція не виробляється, а споживається.

Якщо прагнути до максимізації функції (11.7) без будь-яких обмежень, то це буде досягнуто при нескінченно великих значеннях x_1 і x_2.

Але в умові задачі встановлено обмеження, пов'язані з обсягами доступної сировини, які призводять до таких виразів:

$$\left.\begin{aligned} q_1(\overline{x}) &= a_{1,1}\cdot x_1 + a_{1,2}\cdot x_2 \le C_{1\max};\\ q_2(\overline{x}) &= a_{2,1}\cdot x_1 + a_{2,2}\cdot x_2 \le C_{2\max};\\ q_3(\overline{x}) &= a_{3,2}\cdot x_1 + a_{3,2}\cdot x_2 \le C_{3\max};\\ x_1, x_2 &\ge 0. \end{aligned}\right\}$$

або

$$\left.\begin{aligned} q_1(\overline{x}) &= 1\cdot x_1 + 2\cdot x_2 \le 20;\\ q_2(\overline{x}) &= 6\cdot x_1 + 3\cdot x_2 \le 60;\\ q_3(\overline{x}) &= 32\cdot x_1 + 36\cdot x_2 \le 400;\\ x_1, x_2 &\ge 0. \end{aligned}\right\} \qquad (11.8)$$

Вирази (11.7), (11.8) є формулюванням завдання у вигляді задачі лінійного програмування.

Для наочності відобразимо поставлену задачу графічно у площині двох змінних x_1 і x_2.

Спочатку слід відобразити всі лінії обмежень. Для цього нерівності (11.8) замінюються рівняннями, які будуються на площині

$$\left.\begin{aligned} q_1(\overline{x}) &= 1\cdot x_1 + 2\cdot x_2 = 20;\\ q_2(\overline{x}) &= 6\cdot x_1 + 3\cdot x_2 = 60;\\ q_3(\overline{x}) &= 32\cdot x_1 + 36\cdot x_2 = 400;\\ x_1, x_2 &= 0. \end{aligned}\right\} \qquad (11.9)$$

Їх побудова наведена на рис. 11.7. У результаті сформовано область допустимих значень (позначена всередині точками), обмежена зверху і праворуч лініями обмежень, а знизу і ліворуч – умовою невід'ємності шуканих змінних. На границі цієї області у точках перетину обмежень і повинна розташовуватися точка оптимуму.

Розв'язати поставлену задачу, як і у попередньому прикладі, можна за допомогою методу сіток.

Функція буде повністю аналогічна наведеній вище за винятком формули обчислення значення функції та умови, пов'язаної з обмеженнями.

Створимо функцію для розв'язання поставленої задачі лінійного програмування.

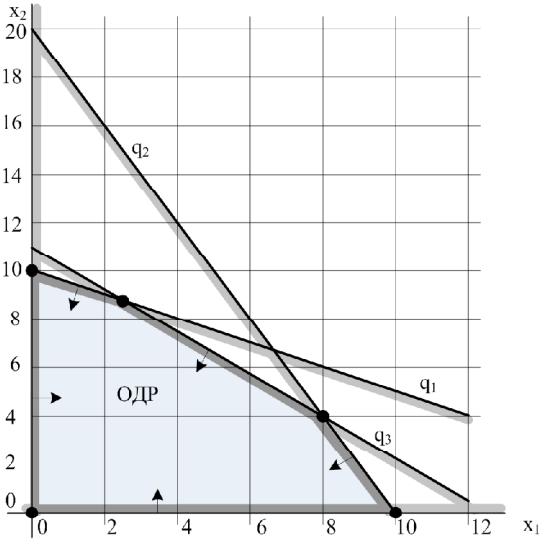


Рис 11.7. Графічне представлення задачі

```
Function LPrMesh1 (x1_0 As Double,x1_k As Double,x2_0
As Double,x2_k As Double,e As Double) As Variant
                        'аргументи функції:область
                        'визначення функції за першим
                        'виміром x1_0 та x1_k, область
                        'визначення функції за другим
                        'виміром x2_0 та x2_k, точність e.
                        'Функція повертає вектор зі
                        'значеннями x1 min, x2 min,
                        'I(x1 min,x2 min)
Dim Res(2) As Double    'оголошення вектора для
                        'повернення значень
Zmax = 0                'визначення первинного значення
                        'змінної максимуму функції
X1 = x1_0               'визначення додаткової змінної
X2 = x2_0               'визначення додаткової змінної
Nx1 = int(abs(x1_0 - x1_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
Nx2 = int(abs(x2_0 - x2_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
For a = 0 to Nx2        'оголошення циклу руху за
                        'другою координатною віссю
```

```
 For b = 0 to Nx1          'оголошення циклу руху за
                           'першою координатною віссю
  Z = 30000*X1 + 20000*X2  'визначення значення функції
  If (X1+2*X2<=20) and (6*X1+3*X2<=60) and
(32*X1+36*X2<=400) Then    'обмеження згідно з умовою
   If Z > Zmax Then        'умова пошуку максимуму функції
    Zmax = Z               'максимальне значення функції
    X1max = X1             'значення аргументу x1 при
                           'максимумі функції
    X2max = X2             'значення аргументу x2 при
                           'максимумі функції
   End If                  'кінець умовного блоку
  End If                   'кінець умовного блоку
  X1 = X1 + e              'пересування першої координати
                           'поточної точки на крок
 Next b                    'кінець циклу b
 X2 = X2 + e               'пересування другої координати
                           'поточної точки на крок
Next a                     'кінець циклу a
Res(0) = X1min             'завантаження точки X1
                           'мінімального значення функції
                           'у вектор результатів
Res(1) = X2min             'завантаження точки X2
                           'мінімального значення функції
                           'у вектор результатів
Res(2) = Zmin              'завантаження  значення
                           'мінімуму функції
LPrMesh = Res              'повернення функції вектору
                           'результатів
End Function               'завершення функції
```

Приклад роботи функції

```
Dim V(2) As double
V = LPrMesh1 (0,10,0,2,0.01)
```

У результаті отримано вектор **V**, що містить координати точки мінімуму функції $x_{1\,\min} = 8$, $x_{2\,\min} = 4$, $I_{\min} = 320000$.

Знайдене розв'язання можна використовувати для додаткового аналізу наявних запасів сировини.

Для виробництва оптимальної кількості продукції потрібно:

– сировини $C_1 = a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 = 1 \cdot 8 + 2 \cdot 4 = 8 + 8 = 16$ тонн;

– сировини $C_2 = a_{2,1} \cdot x_1 + a_{2,2} \cdot x_2 = 6 \cdot 8 + 3 \cdot 4 = 48 + 12 = 60$ брикетів;

– сировини $C_3 = a_{3,1} \cdot x_1 + a_{3,2} \cdot x_2 = 32 \cdot 8 + 36 \cdot 4 = 256 + 144 = 400$ літрів.

Видно, що сировину C_2 і C_3 буде витрачено повністю ($C_2 = C_{2\,max}$, $C_3 = C_{3\,max}$), а сировина C_1 ще залишиться, при цьому надлишок сировини C_1 становитиме $C_{1max} - C_1 = 20 - 16 = 4$ тонни. Отже, при плануванні виробництва можна врахувати це і закупити рівно стільки сировини, скільки потрібно.

Задача лінійного програмування про розкроювання

Потрібно розробити оптимальний план розкрою стандартних листів сталі, який забезпечить вихід планової кількості заготовок різного виду при мінімальних сумарних відходах, якщо відомо, що з партії листової сталі необхідно нарізати чотири види різних заготовок у кількості b_i (i = 1, 2, 3, 4) штук. Лист сталі стандартних розмірів може бути розкроєно чотирма способами. Кожному можливому способу розкрою відповідає карта розкрою. З карт розкрою відомий вихід заготовок у штуках різних видів $a_{i,j}$ (i = 1, 2, 3, 4; j = 1, 2, 3, 4), а також площа відходів c_j (j = 1, 2, 3, 4) при розкрої одного листа сталі за j–м способом розкрою. Отже, необхідно визначити, яку кількість листів сталі необхідно розкроїти тим чи іншим способом, щоб відходи були мінімальними (див. табл. 11.2).

Складемо економіко-математичну модель задачі. Позначимо через x_j – кількість вихідного матеріалу (листів сталі), які необхідно розкроїти за одним із способів j.

Таблиця 11.2

Варіанти розкрою листів

Види заготовок	План-завдання за кількістю заготовок (b_i)	Вихід заготовок (шт.) різних видів з карт розкрою ($a_{i,j}$)			
		1	2	3	4
1	240	1	4	0	1
2	200	1	0	4	0
3	120	1	0	0	3
4	140	1	1	0	3
Площа відходів, м^2 (c_j)		1,4	0,1	2,1	0,1

Цільова функція зводиться до знаходження мінімуму відходів при розкрої

$$F = 1{,}4 \cdot x_1 + 0{,}1 \cdot x_2 + 2{,}1 \cdot x_3 + 0{,}1 \cdot x_4 \to \min.$$

Обмеження у задачі повинні відповідати плановому виходу заготовок *i*–го виду за всіма *j*–ми способами розкрою:

$$\begin{cases} x_1 + 4 \cdot x_2 + x_4 \geq 240; \\ x_1 + 4 \cdot x_3 \geq 200; \\ x_1 + 3 \cdot x_4 \geq 120; \\ x_1 + x_2 + 3 \cdot x_4 \geq 140; \\ x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0. \end{cases}$$

Отже, задачу зведено до звичайної задачі лінійного програмування, яку може бути розв'язано за допомогою методу сіток, як у попередньому прикладі.

11.6.2. Нелінійне програмування

Задачами нелінійного програмування називаються задачі математичного програмування, в яких нелінійними є або цільова функція, або обмеження у вигляді нерівностей чи рівнянь [2].

Спосіб розв'язання обирається залежно від виду функції $f(x)$, що оптимізується.

Приклад

Нехай задано цільову функцію такого виду

$$I(\overline{x}) = 10 - x_1^{0.9} \cdot e^{-0,7 \cdot x_1} \cdot x_2^{0.7} \cdot e^{-0,3 \cdot x_2} \rightarrow max\,.$$

Обмеження

$$\left.\begin{array}{l} q_1(\overline{x}) = 2 \cdot x_1 + 2 \cdot x_2 \leq 10; \\ q_2(\overline{x}) = 5 \cdot x_1^2 + 20 \cdot x_2 \leq 43; \\ x_1, x_2 \geq 0. \end{array}\right\}$$

Як видно з умови, і сама функція, і одне з обмежень є нелінійними. Причому з графічного представлення задачі (рис. 11.8) видно, що точка екстремуму лежить у середині області допустимих значень, а не на її границі.

Для визначення точки екстремуму застосуємо найпростіший та універсальний метод сіток, розглянутий раніше. Оскільки у цільової функції тільки два аргументи, то час на розв'язання буде прийнятним.

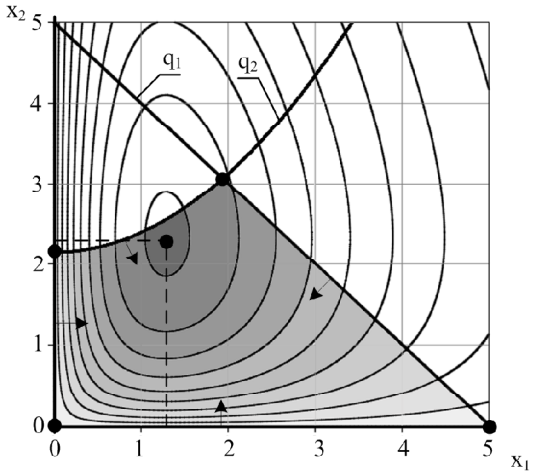


Рис 11.8. Графічне представлення задачі

Граничними значеннями для змінних будуть точки з мінімальними координатами перетину, але оскільки друга умова нелінійна, то слід використати точки перетину із лінією іншої умови: $x_1 = 0\ldots5$, $x_2 = 0\ldots4$.

Створимо функцію визначення максимуму функції.

```
Function ResMeshNl(x1_0 As Double,x1_k As Double,x2_0
As Double,x2_k As Double,e As Double) As Variant
                        'аргументи функції: область
                        'визначення функції за першим
                        'виміром x1_0 та x1_k, область
                        'визначення функції за другим
                        'виміром x2_0 та x2_k, точність e.
                        'Функція повертає вектор зі
                        'значеннями x1 min, x2 min,
                        'I(x1 min,x2 min)
Dim Res(2) As Double    'оголошення вектора для
                        'повернення значень
Zmax = 0                'визначення первинного значення
                        'змінної максимуму функції
X1 = x1_0               'визначення додаткової змінної
X2 = x2_0               'визначення додаткової змінної
Nx1 = int(abs(x1_0 - x1_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
Nx2 = int(abs(x2_0 - x2_k)/e)
                        'визначення кількості комірок
                        'сітки за першим аргументом
```

```
For a = 0 to Nx2          'оголошення циклу руху по другій
                          'координатній осі
 For b = 0 to Nx1         'оголошення циклу руху по першій
                          'координатній осі
  Z = 10+X1^0.9*exp(-0.7*X1)*(-1)*X2^0.9*exp(-0.4*X2)
                          'визначення значення функції
  If (2*X1+2*X2<=10) and (5*X1*X1+20*X2<=43) Then
                          'обмеження згідно з умовою
   If Z > Zmax Then       'умова пошуку максимуму функції
    Zmax = Z              'максимальне значення функції
    X1max = X1            'значення аргументу x1 при
                          'максимумі функції
    X2max = X2            'значення аргументу x2 при
                          'максимумі функції
   End If                 'кінець умовного блоку
  End If                  'кінець умовного блоку
  X1 = X1 + e             'пересування першої координати
                          'поточної точки на крок
 Next b                   'кінець циклу b
 X2 = X2 + e              'пересування другої координати
                          'поточної точки на крок
Next a                    'кінець циклу a
Res(0) = X1min            'завантаження точки X1
                          'мінімального значення функції
                          'у вектор результатів
Res(1) = X2min            'завантаження точки X2
                          'мінімального значення функції
                          'у вектор результатів
Res(2) = Zmin             'завантаження  значення
                          'мінімуму функції
ResMeshNl = Res           'повернення функції вектору
                          'результатів
End Function              'завершення функції
```

Приклад роботи функції

```
Dim V(2) As double
V = ResMeshNl (0,5,0,4,0.001)
```

У результаті отримано вектор **V**, що містить координати точки мінімуму функції $x_{1\,\min} = 1{,}3$, $x_{2\,\min} = 2{,}2$, $I_{\min} = 10{,}458$.

11.7. Багатокритерійна оптимізація

Багатокритерійна оптимізація – це процес одночасної оптимізації двох або більше цільових функцій у заданій області визначення [7].

Постановлення задачі багатокритерійної оптимізації: вимагається знайти значення аргументів $x_1, x_2,..., x_n$, такі, що

$$F_1\left(x_1, x_2, ..., x_n\right) \to extr(\min/\max);$$

.....

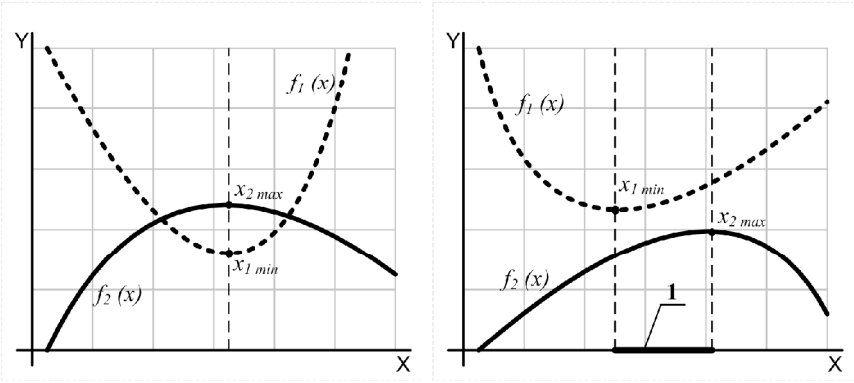
$$F_n\left(x_1, x_2, ..., x_n\right) \to extr(\min/\max),$$

які задовольняють системі обмежень

$$g_i\left(x_1, x_2, ..., x_n\right) \le b_i, \quad i = 1, 2, ..., m\ .$$

Безліч аргументів $x_1, x_2,..., x_n$, що задовольняють системі обмежень, утворює *допустиму область,* елементи якої називаються *допустимими розв'язками* або *альтернативами,* а цільові функції $f_1(x_1, x_2,..., x_n),..., f_n(x_1, x_2,..., x_n)$ – *критеріями*.

Якщо функції $f_1, f_2,..., f_n$ мають екстремум в одній точці, то задача має *ідеальний розв'язок* (рис 11.9, *а*). Однак існування такого ідеального розв'язку, що максимізував або мінімізував одразу декілька цільових функцій, украй рідкісне (рис 11.9, *б*). У разі відсутності ідеального розв'язку задачі, визначається *компромісний розв'язок,* максимально наближений до ідеального. При цьому основна проблема при розв'язанні багатокритерійних завдань – визначення у якому сенсі оптимальний розв'язок краще за інші.



а *б*

Рис 11.9. Розв'язок багатокритерійної задачі оптимізації: *а* – ідеальний розв'язок; *б* – компромісний розв'язок; *1* – область компромісних розв'язків

Наприклад, при пошуку оптимальної конструкції деталі конструктор намагається максимізувати запас міцності і мінімізувати масу. Вочевидь цього неможливо досягнути одночасно, оскільки, чим більше запас міцності, тим масивнішою має бути деталь і тим більше її маса.

Як і у випадку з нелінійним програмуванням, методів пошуку рішень багатокритерійних задач багато, оскільки сфера їх застосування обмежена. Адже кожен метод розроблено під конкретну задачу.

11.7.1. Оптимальність за Парето

Один із розповсюджених підходів до необхідності одночасного врахування декількох критеріїв належить італійському економісту Вільфредо Парето (1848–1923) [7]. Він полягає у відшукуванні непоганих рішень.

Оптимальність за Парето – такий стан системи, при якому значення кожного частинного показника, що характеризує систему, не може бути покращено без погіршення стану інших.

У завданні багатокритерійної оптимізації точка $x_0 \in \mathrm{D}$ називається оптимальною за Парето, якщо не існує іншої точки $x \in \mathrm{D}$, яка була б переважнішою за x_0.

Точки, оптимальні за Парето, утворюють множину непокращуваних або ефективних точок $\mathrm{D_p} \subset \mathrm{D}$, яка має назву *множина Парето*. Оптимальні рішення багатокритерійної задачі слід шукати тільки серед елементів цієї множини альтернатив $\mathrm{D_p}$. У ній жоден критерій не може бути покращено без погіршення хоча б одного з інших.

Приклад

Розглянемо багатокритерійну задачу оптимізації. Нехай є дві цільові функції (два критерії)

$$\begin{cases} f_1(x) = 20 + (2 - 0,5 \cdot x)^2 \to \min; \\ f_2(x) = 10 - (4 - 0,5 \cdot x)^2 \to \max. \end{cases}$$

Обмеження

$$0 \le x \le 10.$$

З графіку (рис 11.10) видно, що задача не є ідеальною – мінімум функції f_1 не збігається з максимумом функції f_2.

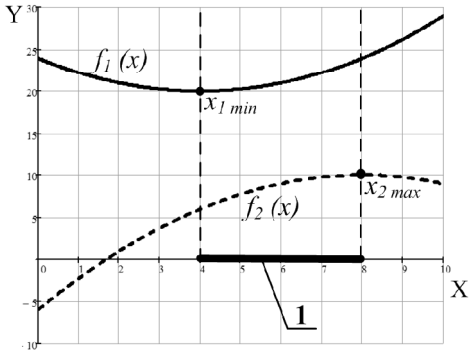


Рис 11.10. Графічне представлення задачі: *1* – область компромісних розв'язків

Екстремуми функцій знаходяться у наступних точках

$$\begin{cases} x_{f_1\,\min} = 4; \\ x_{f_2\,\max} = 8. \end{cases}$$

Побудуємо графік залежності значень однієї функції від іншої (рис 11.11). На графіку Позначимо точки оптимуму для однієї та для іншої функції.

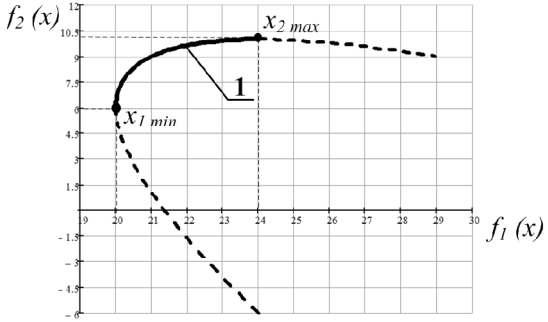


Рис 11.11. Множина Паретто: *1* – область оптимуму за Паретто

Частина графіка між цими точками (*1*) і буде множиною Парето у якій потрібно шукати компромісне рішення. Тобто од-

ного рішення задача не має. Використовуючи той чи інший метод багатокритерійної оптимізації остаточно визначають компромісне значення $x_к$, що задовольняє умовам посталеної задачі.

Як що додати до умови задачі обмеження, то задача може мати одне рішення (рис. 11.12, 11.13).

$$x \cdot 3 + y \le 30,$$

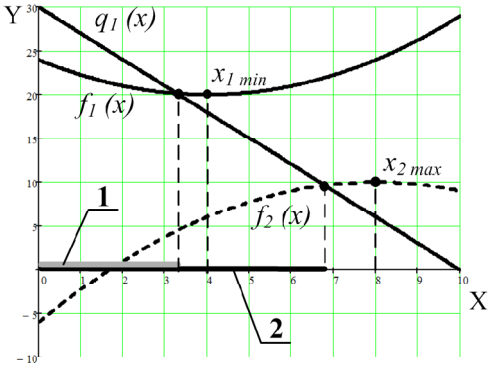


Рис 11.12. Графічне представлення задачі з обмеженням: *1* – область значення аргументу x для функції f_1; *2* – область значення аргументу x для функції f_2

Завдяки доданому обмеженню зменшилась область значень аргументу x:

– для функції f_1

$$0 \le x \le 3{,}291;$$

– для функції f_2

$$0 \le x \le 6{,}79.$$

Загальна область значень аргументу x становитиме $0 \le x \le 3{,}291$.

На графіку залежності значень однієї функції від іншої на основі загальної області значень аргументу x (множини Парето) (рис. 11.13) чітко видно єдине рішення задачі багатокритерійної оптимізації у точці

$$x = 3{,}291.$$

При цьому значення функцій становитимуть

$$f_1(3{,}291) = 20{,}126;$$
$$f_2(3{,}291) = 4{,}456.$$

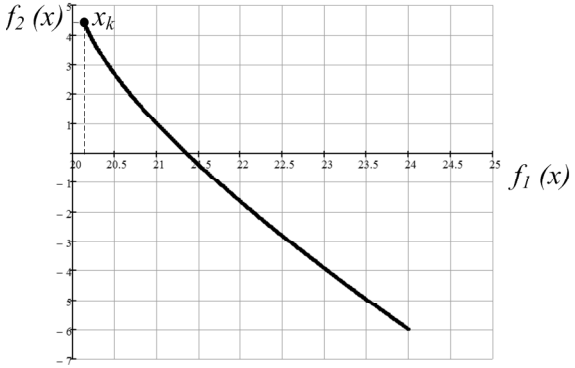


Рис 11.13. Множина Паретто для задачі з обмеженням: x_k – точка оптимуму

11.7.2. Метод пошуку компромісного розв'язку на основі зваженого середньоарифметичного

Розглянемо один з простих та універсальних методів пошуку компромісних розв'язків багатокритерійних задач.

Представлений метод підходить для пошуку компромісного розв'язку цільових функцій явно заданих у вигляді формули.

Компромісною у даному випадку вважається точка, координати якої визначаються як середнєарифметичне від координат точок екстремумів цільових функцій. Тобто метод відшукує середину області компромісних розв'язків. Крім того метод дає змогу використовувати вагові коефіцієнти *k*.

Вагові коефіцієнти враховують важливість того чи іншого критерію, тобто зміщують компромісне розв'язання у бік більш важливого критерію. Вони повинні задовольняти двом умовам

$$\begin{cases} 0 < k_i < 1; \\ \sum k = 1. \end{cases}$$

Алгоритм методу

1. Постановлення завдання оптимізації. Визначення цільових функцій $f_1, f_2, \ldots, f_n$, типу екстремуму (max/min), обмежень та вагових коефіцієнтів $k_1, k_2, \ldots, k_n$.

2. Визначення допустимої області аргументів для кожної з цільових функції на основі обмежень. Встановлення загальної допустимої області аргументів.
3. Визначення точок екстремуму для кожної з цільових функцій $x_{1\ extr}, x_{2\ extr}, \ldots, x_{n\ extr}$.
4. Визначення координати компромісної точки

$$\vec{X}_k = \frac{\sum_{i=1}^{N} \vec{X}_{extr\ i} \cdot k_i}{\sum_{i=1}^{N} k_i}.$$

Приклад

Розглянемо багатокритерійну задачу оптимізації з п.11.7.1 з двома цільовими функціями (рис. 11.10)

$$\begin{cases} f_1(x) = 20 + (2 - 0,5 \cdot x)^2 \to \min; \\ f_2(x) = 10 - (4 - 0,5 \cdot x)^2 \to \max. \end{cases}$$

і обмеженням

$$0 \le x \le 10$$

Допустима область аргументів для обох функції однакова. Мінімум для $f_1(x_{max}) = 4$, максимум для $f_2(x_{max}) = 8$. Задамо вагові коефіцієнти обох функцій однаковими $k_1 = k_2 = 0,5$.

Визначення компромісної точки наведено на рис. 11.14.

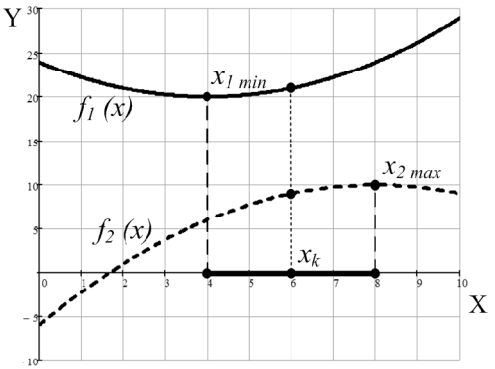


Рис. 11.14. Компромісне рішення

Отже, координати точки компромісного розв'язання

$$x_k = \frac{x_1 \cdot k_1 + x_2 \cdot k_2}{k_1 + k_2} = \frac{4 \cdot 0,5 + 8 \cdot 0,5}{0,5 + 0,5} = 6 .$$

11.7.3. Метод багатокритерійної оптимізації для неявнозаданих функцій

Представлений метод є модифікацією попереднього методу (п.11.7.2) і використовується для пошуку компромісного розв'язку цільових функцій неявно заданих у вигляді таблиць значень. Він, як і базовий метод, відшукує середину області компромісних рішень з урахуванням вагових коефіцієнтів.

Алгоритм методу

1. Постановлення завдання оптимізації. Визначення цільових функцій $f_1, f_2, \ldots, f_n$, типу екстремуму (max/min), обмежень та вагових коефіцієнтів $k_1, k_2, \ldots, k_n$.
2. Визначення допустимої області аргументів для кожної з цільових функції на основі обмежень. Встановлення загальної допустимої області аргументів.
3. Побудова таблиць значень цільових функцій на основі аргументів з допустимої області з урахуванням обмежень. Кожна така таблиця містить стовпці аргументів і стовпець значень функції.
4. Сортування таблиць значень цільових функції згідно із задачею оптимізації (у порядку спадання при максимізації, або у порядку збільшення при мінімізації) (рис. 11.15).

max

x_1	x_2	x_n	$f_1(x_1, x_2, \ldots, x_n)$
1,13	809	70	0,0934
5,86	577	82	0,082
6,91	575	15	0,0804
5,29	878	36	0,0793
1,38	275	68	0,0703
1,24	812	52	0,0634
7,83	513	13	0,0611
0,56	875	44	0,0592
4,76	782	88	0,0544
2	619	71	0,0434
4,69	594	72	0,0334
1,34	423	26	0,0279
6,16	897	88	0,0246
1,32	415	32	0,0201

min

max

x_1	x_2	x_n	$f_2(x_1, x_2, \ldots, x_n)$
5,29	878	36	123
6,08	691	57	112
6,51	562	59	110
7,77	323	41	109
2,88	396	18	103
2,35	715	28	98
0,13	854	12	96
3,33	231	45	92
5,5	895	59	90
4,98	462	64	89
7,74	435	24	85
1,47	485	84	83
0,83	571	15	73
5,18	627	85	67

min

max

x_1	x_2	x_n	$f_m(x_1, x_2, \ldots, x_n)$
7,57	310	63	23,943
6,71	281	14	23,932
4,33	686	30	23,912
3,07	251	41	23,832
5,29	878	36	23,81
1,14	895	32	23,801
4,37	384	54	23,793
0,12	417	75	23,754
8,91	747	73	23,712
5,41	787	38	23,523
8,28	895	38	23,511
5,41	658	53	23,493
7,96	320	15	23,432
8,32	331	10	23,422

min

Рис. 11.15. Сортування таблиць значень цільових функцій

5. З таблиць значень у межах заданого інтервалу пошуку Δ обираються співпадаючі набори аргументів з урахуванням вагових коефіцієнтів (рис 11.16)

$$x_1 = x_2\left(\frac{k_2}{k_1}\right) = \ldots = x_n\left(\frac{k_n}{k_1}\right).$$

x_1	x_2	x_n	$f_1(x_1, x_2, \ldots, x_n)$
1,13	809	70	0,0934
5,86	577	82	0,082
6,91	575	15	0,0804
5,29	878	36	0,0793
1,38	275	68	0,0703
1,24	812	52	0,0634
7,83	513	13	0,0611
0,56	875	44	0,0592
4,76	782	88	0,0544
2	619	71	0,0434
4,69	594	72	0,0334
1,34	423	26	0,0279
6,16	897	88	0,0246
1,32	415	32	0,0201

x_1	x_2	x_n	$f_2(x_1, x_2, \ldots, x_n)$
5,29	878	36	123
6,08	691	57	112
6,51	562	59	110
7,77	323	41	109
2,88	396	18	103
2,35	715	28	98
0,13	854	12	96
3,33	231	45	92
5,5	895	59	90
4,98	462	64	89
7,74	435	24	85
1,47	485	84	83
0,83	571	15	73
5,18	627	85	67

x_1	x_2	x_n	$f_m(x_1, x_2, \ldots, x_n)$
7,57	310	63	23,943
6,71	281	14	23,932
4,33	686	30	23,912
3,07	251	41	23,832
5,29	878	36	23,81
1,14	895	32	23,801
4,37	384	54	23,793
0,12	417	75	23,754
8,91	747	73	23,712
5,41	787	38	23,523
8,28	895	38	23,511
5,41	658	53	23,493
7,96	320	15	23,432
8,32	331	10	23,422

Рис. 11.16. Визначення оптимального варіанту

Інтервал пошуку Δ на початку становить 1. Потім він збільшується до повного розміру таблиці

У результаті визначаються аргументи компромісного рішення, що відповідають середині області компромісних рішень з урахуванням вагових коефіцієнтів.

11.8. Приклад проектної процедури оптимізації

Розглянемо розроблення проектної процедури для визначення оптимального варіанту за трьома критеріями конструкції машини з урахуванням вагових коефіцієнтів.

Задачі такого типу дуже часто зустрічаються при проектуванні, коли у результаті розрахунків є декілька варіантів машини, що відрізняються параметрами. Потрібно знайти найкращий варіант. Причому за одними критеріями потрібно шукати максимум, а за іншими – мінімум.

Для розв'язання поставленого завдання використаємо метод багатокритерійної оптимізації для неявнозаданих цільових функцій.

У розглянутому прикладі проектна процедура працює з трьома цільовими функціями, що мають тільки один аргумент.

Проектна процедура у якості вихідних даних використовує таблицю, записану у CSV-файл. Таблиця має 4 стовпці: перший – значення аргументу або номер варіанту конструкції, другий, третій та четвертий містять відповідні значення цільових функцій.

Проектування форми користувача. На формі користувача розташовуються елементи управляння згідно з рис. 11.17.

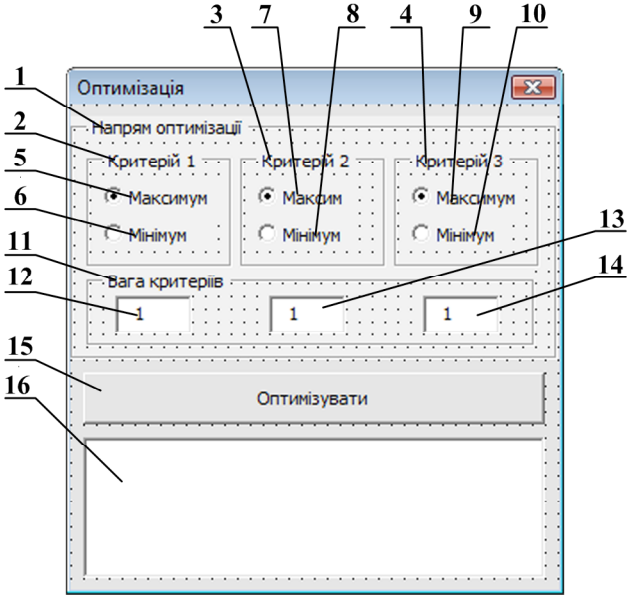


Рис 11.17. Форма користувача та елементи управління: *1* – рамка **`Frame1`**; *2–4* – рамки **`Frame2`**–**`Frame4`**; *5–10* – кнопки–перемикачі **`OptionButton1`** – **`OptionButton6`**; *11* – рамка **`Frame5`**; *12–14* – поля для введення **`TextBox1`**–**`TextBox3`**; *15* – кнопка **`CommandButton1`** з надписом **`"Оптимізувати"`**; *16* – список для виведення результатів **`ListBox1`**

При розробці форми користувача у менеджері властивостей встановлюються наступні параметри:

– для форми **`UserForm1`**:

```
UserForm1.Caption = "Оптимізація"
```

– для кнопки **`CommandButton1`**:

```
CommandButton1.Caption = "Оптимізувати";
```

обробник події **CommandButton1_Click()**;

– для надпису **Label1**:

```
Label1.Caption = "Введіть число для округлення";
Label1.Font.Size = 12;
```

– для рамки **Frame1**:

```
Frame1.Caption = "Напрям оптимізації"
```

– для рамки **Frame1**:

```
Frame1.Caption = "Критерій 1"
```

– для рамки Frame2:

```
Frame2.Caption = "Критерій 2"
```

– для рамки Frame3:

```
Frame3.Caption = "Критерій 3"
```

– для кнопок–перемикачів **OptionButton1**, **OptionButton3**, **OptionButton5**:

```
OptionButton1.Caption = "Максимум";
OptionButton1.Value = True;
```

– для кнопок–перемикачів **OptionButton2**, **OptionButton4**, **OptionButton6**:

```
OptionButton2.Caption = "Мінімум";
OptionButton2.Value = False;
```

– для рамки Frame4:

```
Frame1.Caption = "Вага критеріїв"
```

– для полів введення **TextBox1, TextBox2, TextBox3**:

```
TextBox1.Text = "1".
```

Проектна підсистема містить головну програму, дві функції та 1 обробник події натиснення кнопки **CommandButton1** для оптимізації даних.

У підсистемі використано функцію **CSVLoad**, яка розглянута раніше (див. п.5.2.4).

Головна програма та функції

```
Public DB() As Double     'оголошення глобального
                          '(доступного усім процедурам і
                          'функціям) динамічного масиву.
Function Sort_max(V As Variant, k As Integer) As Variant
                          'оголошення функції сортування
                          'за спаданням таблиці значень
                          'функції. Аргумент V – масив,
                          'що містить значення аргументу
                          'та цільової функції, k –
                          'кількість значень аргументу.
Dim n As Double           'оголошення допоміжної змінної
For A = 0 To k – 1        'оголошення циклу
 For B = 0 To k – 2       'оголошення циклу
```

```
  If V(B, 1) < V(B + 1, 1) Then
                        'якщо поточний елемент масиву
                        'більший за наступний, то
                        'потрібно змінити місцями ці
                        'елементи. Для цього...
   n = V(B + 1, 1)      'змінній N присвоїти значення
                        'наступного елементу масиву
   V(B + 1, 1) = V(B, 1)'наступному елементу масиву
                        'присвоїти значення поточного
                        'елементу масиву
   V(B, 1) = n          'поточному елементу масиву
                        'присвоїти значення змінної N
   n = V(B + 1, 0))     'змінній N присвоїти значення
                        'наступного елементу масиву
   V(B + 1, 0) = V(B, 0)'наступному елементу масиву
                        'присвоїти значення поточного
                        'елементу масиву
   V(B, 0) = n          'поточному елементу масиву
                        'присвоїти значення змінної N
  End If                'завершення умови
 Next B                 'завершення циклу
Next A                  'завершення циклу
Sort_max = V            'повернення відсортованого масиву
End Function            'завершення функції

Function Sort_min(V As Variant, k As Integer) As Variant
                        'оголошення функції сортування
                        'за зростанням таблиці значень
                        'функції. Аргумент V – масив,
                        'що містить значення аргументу
                        'та цільової функції, k –
                        'кількість значень аргументу.
Dim n As Double         'оголошення допоміжної змінної
For A = 0 To k – 1      'оголошення циклу
 For B = 0 To k – 2     'оголошення циклу
  If V(B, 1) > V(B + 1, 1) Then
                        'якщо поточний елемент масиву
                        'більший за наступний, то
                        'потрібно змінити місцями ці
                        'елементи. Для цього...
   n = V(B + 1, 1)      'змінній N присвоїти значення
                        'наступного елементу масиву
   V(B + 1, 1) = V(B, 1)'наступному елементу масиву
                        'присвоїти значення поточного
                        'елементу масиву
   V(B, 1) = n          'поточному елементу масиву
                        'присвоїти значення змінної N
   n = V(B + 1, 0))     'змінній N присвоїти значення
                        'наступного елементу масиву
```

```
   V(B + 1, 0) = V(B, 0)  'наступному елементу масиву
                          'присвоїти значення поточного
                          'елементу масиву
   V(B, 0) = n            'поточному елементу масиву
                          'присвоїти значення змінної N
  End If                  'завершення умови
 Next B                   'завершення циклу
Next A                    'завершення циклу
Sort_min = V              'повернення відсортованого масиву
End Function              'завершення функції

    Sub main()            'головна програма
      UserForm1.Show      'запуск форми користувача
    End Sub
```

Обробник події натиснення кнопки **CommandButton1** для оптимізації даних.

```
Private Sub CommandButton1_Click()
Dim V1() As Double, V2() As Double, V3() As Double
                          'оголошення динамічних масивів
                          'для завантаження таблиць
                          'цільових функцій
CSVLoad ("c:\Data.csv")   'завантаження таблиці з
                          'CSV-файлу
Dim M_l As Integer        'оголошення допоміжної змінної
M_l = FileLen("c:\Data.csv")
                          'визначення кількості рядків у
                          'таблиці
ReDim V1(M_l - 1, 1)
ReDim V2(M_l - 1, 1)
ReDim V3(M_l - 1, 1)      'задавання розміру масивів
                          'для кожної цільової функції
                          'таблиці
For A = 0 To M_l - 1      'оголошення циклу завантаження
                          'значення аргументу і цільових
                          'функцій
 V1(A, 0) = DB(0, A): V1(A, 1) = DB(1, A)
 V2(A, 0) = DB(0, A): V2(A, 1) = DB(2, A)
 V3(A, 0) = DB(0, A): V3(A, 1) = DB(3, A)
                          'завантаження значення
                          'аргументу і цільових функцій
Next A                    'завершення циклу
If OptionButton1.Value = True Then V1 = Sort_max(V1,
M_l) Else V1 = Sort_min(V1, M_l)
                          'якщо кнопка-перемикач
                          'OptionButton1 обрано, то
```

```
                        'відбувається сортування
                        'таблиці першої цільової
                        'функції за зростанням, інакше
                        'сортування за спаданням
If OptionButton3.Value = True Then V2 = Sort_max(V2,
M_l) Else V2 = Sort_min(V2, M_l)
                        'якщо кнопка-перемикач
                        'OptionButton3 обрано, то
                        'відбувається сортування
                        'таблиці другої цільової
                        'функції за зростанням, інакше
                        'сортування за спаданням
If OptionButton5.Value = True Then V3 = Sort_max(V3,
M_l) Else V3 = Sort_min(V3, M_l)
                        'якщо кнопка-перемикач
                        'OptionButton5 обрано, то від
                        'бувається сортування
                        'таблиці третьої цільової
                        'функції за зростанням, інакше
                        'сортування за спаданням
k1 = Val(TextBox1.Text)
k2 = Val(TextBox2.Text)
k3 = Val(TextBox3.Text)  'завантаження вагових
                        'коефіцієнтів
f = 0                   'задавання початкового значення
                        'змінної-прапорця визначення
                        'рішення
For A = 0 To M_l - 2    'оголошення циклу
 For Delta = 0 To M_l   'оголошення циклу
  For B = 0 To Delta    'оголошення циклу
   For C = 0 To Delta   'оголошення циклу
    For D = 0 To Delta  'оголошення циклу
     If V1(A + B, 0) = V2(A + C, 0) * (k2 / k1) And
V1(A + B, 0) = V3(A + D, 0) * (k3 / k1) Then
                        'якщо аргумент з першої
                        'таблиці дорівнює аргументу з
                        'другої та третьої таблиць, то
                        'рішення знайдене
      ListBox1.AddItem "Оптимальний варіант " +
Str(V1(A + B, 0))       'повідомлення про знайдене рішення
      ListBox1.AddItem "Допуск " + Str(Delta + 1)
                        'повідомлення про допуск рішення
      f = 1             'значення змінної-прапорця про
                        'знайдене рішення
      GoTo Opt          'дострокове завершення циклу
                        'пошуку
     End If             'завершення умови
    Next D              'завершення циклу
```

```
   Next C                    'завершення циклу
  Next B                     'завершення циклу
 Next Delta                  'завершення циклу
Next A                       'завершення циклу
Opt:                         'мітка для безумовного переходу
If f = 0 Then ListBox1.AddItem "Допустимого рішення не
знайдено"                    'якщо прапорець не змінено, то
                             'видається повідомлення про
                             'відсутність рішення
End Sub                      'завершення обробника події
```

Розглянемо роботу створеної програми на прикладі вибору оптимальної марки автомобіля за трьома критеріями (табл. 11.3). Потрібно визначити який автомобіль обрати, щоб він був достатньо потужним, з невеликою витратою палива та коштував недорого.

Таблиця 11.3

Параметри автомобілів

Критерій	VW Golf	Opel Astra	Ford Focus	Toyota Corolla
Ціна, 1000 Euro	16.2	14.9	14.0	15.2
Витрата палива на 100 км, л	7.2	7.0	7.5	8.2
Потужність, кВт	66.0	62.0	55	71

Отже, завдання оптимізації має наступний вид:

– ціна $\rightarrow$ min;

– витрата палива на 100 км $\rightarrow$ min;

– потужність $\rightarrow$ max.

Попередньо проаналізуємо цю задачу за допомогою критерію Парето. На рис. 11.18 наведено точки, які відповідають маркам автомобілів. З графіку випливає, що однозначного розв'язку завдання не має. Автомобілі марки Opel (*2*) і Ford (*3*) є Парето-оптимальним розв'язанням.

З таблиці 11.1 формуємо CSV-файл даних. Марки машин виступають у ролі аргументу. Шифруємо їх за порядковими номерами. CSV-файл даних

```
1;16.20;7.20;66.00
2;14.90;7.00;62.00
3;14.00;7.50;55.00
4;15.20;8.20;71.00
```

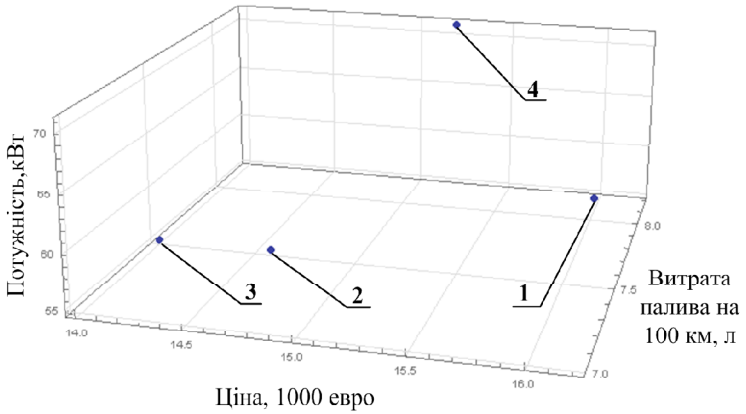


Рис 11.18. Оптимальність за Парето: *1* – VW Golf; *2* – Opel Astra; *3* – Ford Focus; *4* – Toyota Corolla

Запускаємо програму та задаємо відповідні вихідні дані (рис.11.19).

У результаті роботи проектної процедури визначено оптимальну марку автомобіля за трьома критеріями. Найкращим рішенням виявився автомобіль марки Opel Astra (варіант 2). Це співпадає з Парето-оптимальним розв'язанням.

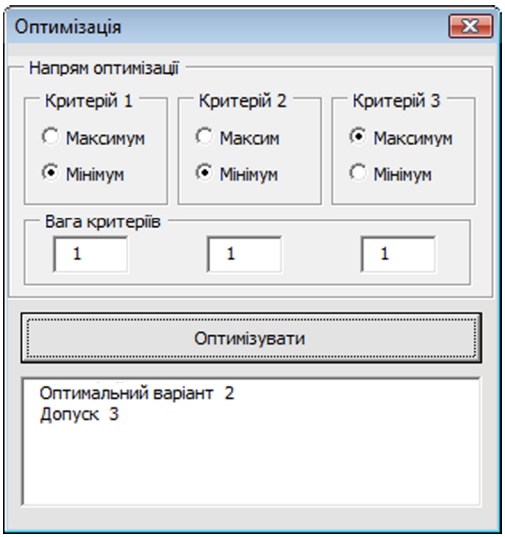


Рис. 11.19. Результат роботи проектної процедури

Контрольні запитання і завдання

1. Дайте визначення оптимізації.
2. Сформулюйте задачу оптимізації у загальному вигляді.
3. Класифікуйте методи оптимізації.
4. Які методи оптимізації використовуються коли цільову функцію задано таблицею значень?
5. Опишіть порядок розв'язання задач одновимірної оптимізації методом золотого перерізу.
6. Опишіть порядок розв'язання задач багатовимірної оптимізації методом покоординатного спуску.
7. Опишіть порядок розв'язання задач одновимірної оптимізації методом сіток.
8. Дайте визначення математичного програмування.
9. Вкажіть особливості лінійного програмування.
10. Вкажіть особливості нелінійного програмування.
11. Дайте визначення багатокритерійної оптимізації.
12. Опишіть порядок розв'язання задач багатокритерійної оптимізації методом пошуку компромісного рішення на основі зваженого середньоарифметичного.
13. Опишіть порядок розв'язання задач багатокритерійної оптимізації методом пошуку компромісного рішення для неявнозаданих функцій.
14. Створіть блок-схему функції для визначення мінімуму методом золотого перерізу (див. п.11.4.2).
15. Створіть блок-схему функції для визначення екстремуму методом покоординатного спуску (див. п.11.5.2).
16. Створіть блок-схему функції для визначення екстремуму методом сіток (див. п.11.5.3).
17. Створіть блок-схему обробника події натиснення кнопки **`CommandButton1`** (див. п.11.8).
18. Створіть функцію для розв'язання задачі про оптимальне розкроювання
19. Створіть проектну процедуру для визначення оптимального варіанту методом багатокритерійної оптимізації для неявнозаданих цільових функцій, як у п.11.8. Додайте можливість пошуку оптимуму не за одним, як у п. 11.8, а за трьома аргументами.

Використана література

1. Бондарик В.М. Системы автоматизированного проектирования. Курс лекций: Уч. пособие для студ. спец. «Медицинская электроника», «Электронно–оптические системы и технологии» дневной и заочной форм обуч. – Мн. БГУИР, 2006. – 272 с.

2. Васильев Ф.П. Методы оптимизации. – М.:Издательство «Факториал Пресс», 2002. – 824 с.

3. Гарбер Г.З. Основы программирования на Visual Basic и VBA в Excel 2007. – М.: СОЛОН–ПРЕСС, 2008. – 92 с.: ил.

4. ГОСТ 23501.101–87. Системы автоматизированного проектирования. Основные положения. – М.: Государственный комитет СССР по стандартам, 1988. – 11с.

5. Кен Гети, Майк Гилберт. Программирование на Visual Basic 6 и VBA. Руководство разработчика: Пер. с англ. – К.: Издательская группа BHV, 2001. – 912 с., ил.

6. Ли К. Основы САПР (CAD/CAM/CAE). – СПб.: Питер, 2004. – 560 с.

7. Лотов А.В., Поспелова И.И. Многокритериальные задачи принятия решений: Учебное пособие. – М.: МАКС Пресс, 2008. – 197 с.

8. Зуев С.А., Полещук Н.Н. САПР на базе AutoCAD – как это делается. – СПб.: БХВ–Петербург, 2004. – 1168 с.

9. Норенков И.П. Основы автоматизированного проектирования: Учеб.для вузов. 2–е изд., перераб. и доп. – М.: МГТУ им. Н.Э. Баумана, 2002. – 336 с.

10. SolidWorks API Help. – Режим доступа: http://help.solidworks.com/2012/English/api/SWHelp_List.html?id=f1c9cb18f6d245cdb312b56985ca6199#Pg0&ProductType=&ProductName=

Навчальне видання

Хруцький Андрій Олександрович

ОСНОВИ РОЗРОБКИ ПРОЕКТНИХ ПІДСИСТЕМ НА БАЗІ SOLIDWORKS API

Навчальний посібник

Видавничий центр ДВНЗ «КНУ»
В.о. директора О.В. Соломенний
Технічний редактор М.С. Куций
Редактор І.В. Ланова
Комп'ютерний набор та верстка Л.В. Беспалова

Підписано до друку 01.09.2016
Формат 60/84 1/32. Папір офсетний
Ум. друк. арк. 00,0 Обл. вид. арк. 00,0
Тираж 300 прим.

Видавничий центр ДВНЗ
«Криворізький національний університет»
Свідоцтво суб'єкта видавничої справи
ДК№4328 від 24.05.2015 р.,
вул. XXII партз'їзду, 11, Криврй Ріг, 50027
тел. 409-17-23

Надруковано з оригинал-макету в друкарні
видавець ФОП Озеров Г.В.
м. Харків, вул. Університетська, 3, кв. 9
Свідоцтво про державну реєстрацію
№818604 від 02.03.2000 р.