МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

О. Є. Поморцева

ПРОГРАМУВАННЯ ГЕОІНФОРМАЦІЙНИХ ЗАДАЧ

КОНСПЕКТ ЛЕКЦІЙ

(для студентів першого (бакалаврського) рівня вищої освіти спеціальності 193 – Геодезія та землеустрій)

Харків
ХНУМГ ім. О. М. Бекетова
2021

Поморцева О. Є. Програмування геоінформаційних задач : конспект лекцій для студентів першого (бакалаврського) рівня вищої освіти спеціальності 193 – Геодезія та землеустрій) / О. Є. Поморцева ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2021. – 86 с.

Автор

Поморцева *Олена Євгенівна*, кандидат технічних наук, доцент

Рецензент

О. Б. Костенко, кандидат фізико-математичних наук, доцент кафедри комп'ютерних наук та інформаційних технологій ХНУМГ ім. О. М. Бекетова

Рекомендовано кафедрою земельного адміністрування та геоінформаційних систем, протокол № 1 від 31.08.2021.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ПК – персональний комп’ютер
МП – мова програмування
ЕОМ – електрона обчислювальна машина
МВР – мова програмування високого рівня
МНВР – мова програмування надвисокого рівня
ООП – об’єктно-орієнтоване програмування
БД – база даних
ОС – операційна система
VBA – мова програмування Visual Basic for Applications

1 ІСТОРІЯ РОЗВИТКУ МОВ ПРОГРАМУВАННЯ

Розвиток обчислювальної техніки супроводжується створенням нових і вдосконаленням існуючих засобів спілкування програмістів з персональним комп'ютером (ПК) – мов програмування (МП). Під ЯП розуміють правила подання даних і запису алгоритмів їх обробки, які автоматично виконуються ПК. У більш абстрактному вигляді МП є засобом створення програмних моделей об'єктів і явищ зовнішнього світу. До теперішнього часу створені десятки різних МП – від найпримітивніших до близьких до природної мови людини (рис. 1.1).

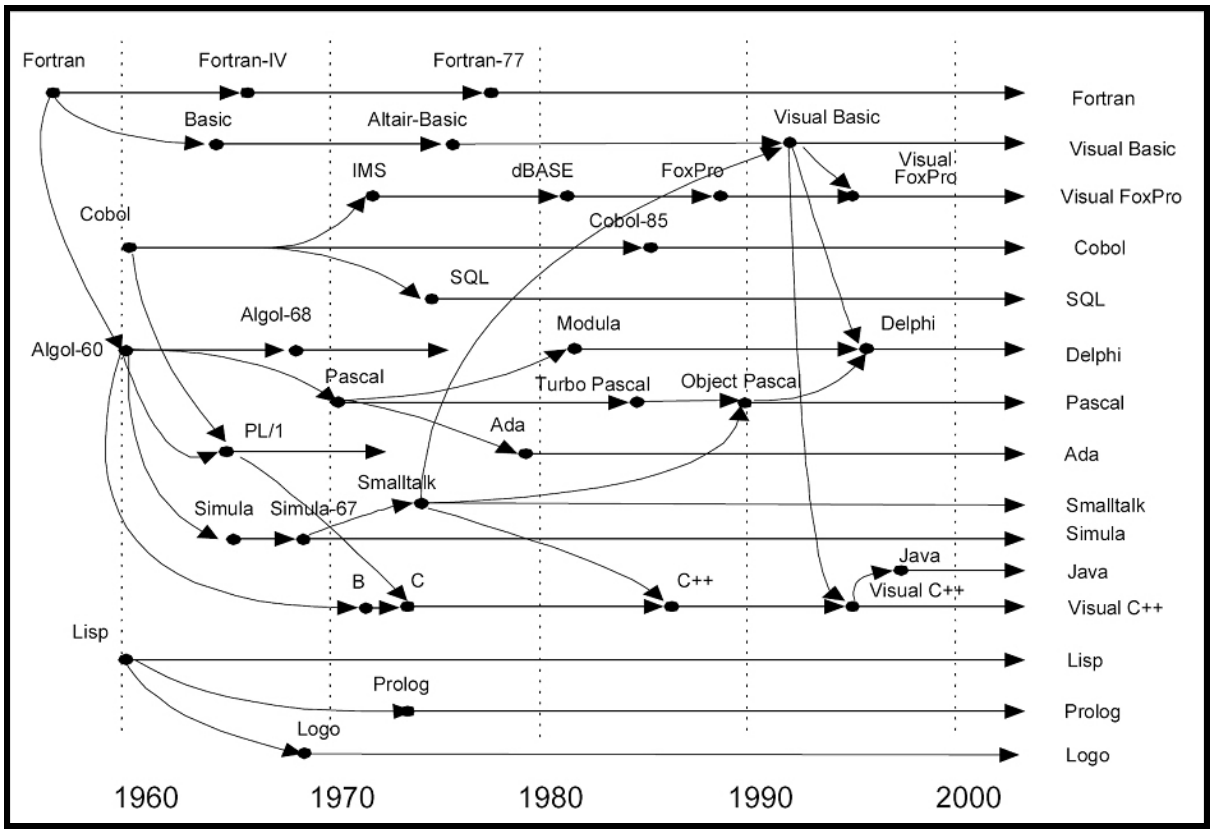


Рисунок 1.1 – Хронологія розвитку мов програмування

Щоб розібратися у всьому різноманітті МП, потрібно знати їх класифікацію, а також історію створення, еволюцію і тенденції розвитку.

Рушійні сили еволюції мов програмування

По-перше, МП є інструментом програміста для створення програм. Для створення хороших програм потрібні хороші МП. Тому однією з рушійних сил еволюції програмування є прагнення розробників до

створення більш досконалих програм. По-друге, процес розробки програми можна порівнювати з промисловим виробництвом, в якому визначальними чинниками є продуктивність праці колективу програмістів, собівартість і якість програмної продукції. Створюються різні технології розробки програм (структурне, модульне, об'єктно-орієнтоване програмування), які повинні підтримуватися МП.

Отже, другою рушійною силою еволюції МП є прагнення до підвищення ефективності процесу виробництва програмної продукції. По-третє, програми можна розглядати як аналог пристроїв-конструкторів, в яких замість окремих деталей використовують конструкції МП (елементна база програми). Як і електронні пристрої, програми можуть бути найпростішими (рівня детекторного приймача) і дуже складними (рівня автоматичної космічної станції), при цьому рівень інструменту повинен відповідати складності виробу. Крім того, людині зручніше описувати модельований об'єкт в термінах предметної області, а не мовою цифр.

Таким чином, третьою рушійною силою, яка веде до створення нових, спеціалізованих, орієнтованих на проблемну область і більш потужних МП, є збільшення різноманітності і підвищення складності завдань, що вирішуються за допомогою ПК.

По-четверте, вдосконалення самих ПК призводить до необхідності створення мов, які максимально реалізують нові можливості ПК.

По-п'яте, програми є інтелектуальним продуктом, який потрібно накопичувати і примножувати. Але програми, як і технічні вироби, мають властивість морального старіння, однією з причин якого є їх залежність від типу ПК і операційного середовища. З моральним старінням програм борються шляхом їх модернізації та випуску нових версій, однак при високій динаміці зміни типів ПК і операційних середовищ розробники будуть тільки тим і займатися, що модернізувати старі програми. Тому, МП повинні забезпечувати тривалий життєвий цикл програми, і прагнення до цього є п'ятою рушійною силою розвитку МП.

Відомо, що першим програмістом була жінка – леді Ада Лавлейс, дочка лорда Байрона. Вона розробляла програми для одного з перших механічних комп'ютерів, створеного на початку XIX століття англійським вченим Чарльзом Беббіджем. Однак програмування в сучасному розумінні почалося з моменту створення першої електронної обчислювальної машини. Але тим не менше, ім'я цієї чудової жінки – «**Ada**» – присвоєно

одній з найпотужніших сучасних МП, яка є базовою для міністерства оборони США.

Перші електронні обчислювальні машини (ЕОМ), створені людиною, мали невеликий набір команд і вбудованих типів даних, але дозволяли виконувати програми на машинній мові. Машинна мова (ММ) – єдина мова, зрозуміла ЕОМ. Він реалізується апаратно: кожну команду виконує деякий електронний пристрій. Програма на МП є послідовність команд і даних, заданих в цифровому вигляді. Наприклад, команда вигляду «**1A12**» у 16-річному вигляді або «**0001101000010010**» у двійковому вигляді означає операцію додавання (1A) вмісту регістрів 1 і 2.

Дані на МП представлені числами і символами. Операції є елементарними і з них будується вся програма. Введення програми в цифровому вигляді проводиться безпосередньо в пам'ять з клавіатури ЕОМ або з примітивних пристроїв введення. Природно, що процес програмування був дуже трудомістким, розібратися в програмі навіть автору було досить складно, а ефект від застосування ЕОМ був досить низьким. Цей етап у розвитку МП показав, що програмування є складною проблемою, важко піддається автоматизації, але саме програмне забезпечення визначає в кінцевому рахунку ефективність застосування ЕОМ. Тому на всіх наступних етапах зусилля спрямовувалися на вдосконалення інтерфейсу між програмістом і ЕОМ , тобто самої мови програмування.

Прагнення програмістів оперувати не цифрами, а символами, призвело до створення мнемонічної мови програмування, яку називають асемблером, мнемокодом, автокодом. Ця мова має певний синтаксис запису програм, у якому, зокрема, цифровий код замінений мнемонічним кодом. Наприклад, команда додавання записується у вигляді «**AR 1,2**» і означає додавання (**Addition**) типу регістр-регістр (**Register**) для регістрів 1 і 2. Тепер програма має більш легку для читання форму, але її не розуміє ЕОМ. Тому знадобилося створити спеціальну програму транслятор, яка перетворює програму з мови асемблера на МП. Ця проблема призвела до глибоких наукових досліджень і розробки різних теорій, наприклад теорії формальних мов, які лягли в основу створення трансляторів. Практично будь-який клас ЕОМ має свою мову асемблера. На сьогоднішній день мова асемблера використовується для створення системних програм, що використовують специфічні апаратні можливості даного класу ЕОМ.

Наступний етап характеризується створенням мов високого рівня (МВР). Ці мови є універсальними (на них можна створювати будь-які прикладні програми) і алгоритмічно повними, мають більш широкий спектр типів даних і операцій, підтримують технології програмування. На цих мовах створюється сила-силенна різних прикладних програм.

Принциповими відмінності МВР від мов низького рівня:

– використання змінних;

– можливість запису складних виразів;

– розширюваність типів даних за рахунок конструювання нових типів з базових;

– розширюваність набору операцій за рахунок підключення бібліотек підпрограм; слабка залежність від типу ЕОМ.

З ускладненням МП ускладнюються і транслятори для них. Тепер в набір інструментів програміста, окрім транслятора, входить текстовий редактор для введення тексту програм, відладчик для усунення помилок, бібліотека для створення чи вибору програмних модулів і безліч інших службових програм. Всі разом це називається системою програмування. Найбільш яскравими представниками МВР є **FORTRAN**, **PL/1, Pascal, C, Basic, Ada**.

Може виникнути питання: чому створено стільки різних мов одного класу? Чому не можна створити одну мову на всі випадки життя? Відповідь на це питання може бути таким же, як і на питання про різні мови народів світу: так вже вийшло. Кожен з розробників МВР прагнув створити найкращу і саму універсальну мову, яка б дозволяла швидко отримувати найефективніші, надійні і безпомилкові програми. Однак в процесі цього пошуку з’ясувалося, що справа не в самій мові, а в технології її використання, тому подальший розвиток мов став визначатися новими технологіями програмування (рис. 1.2).

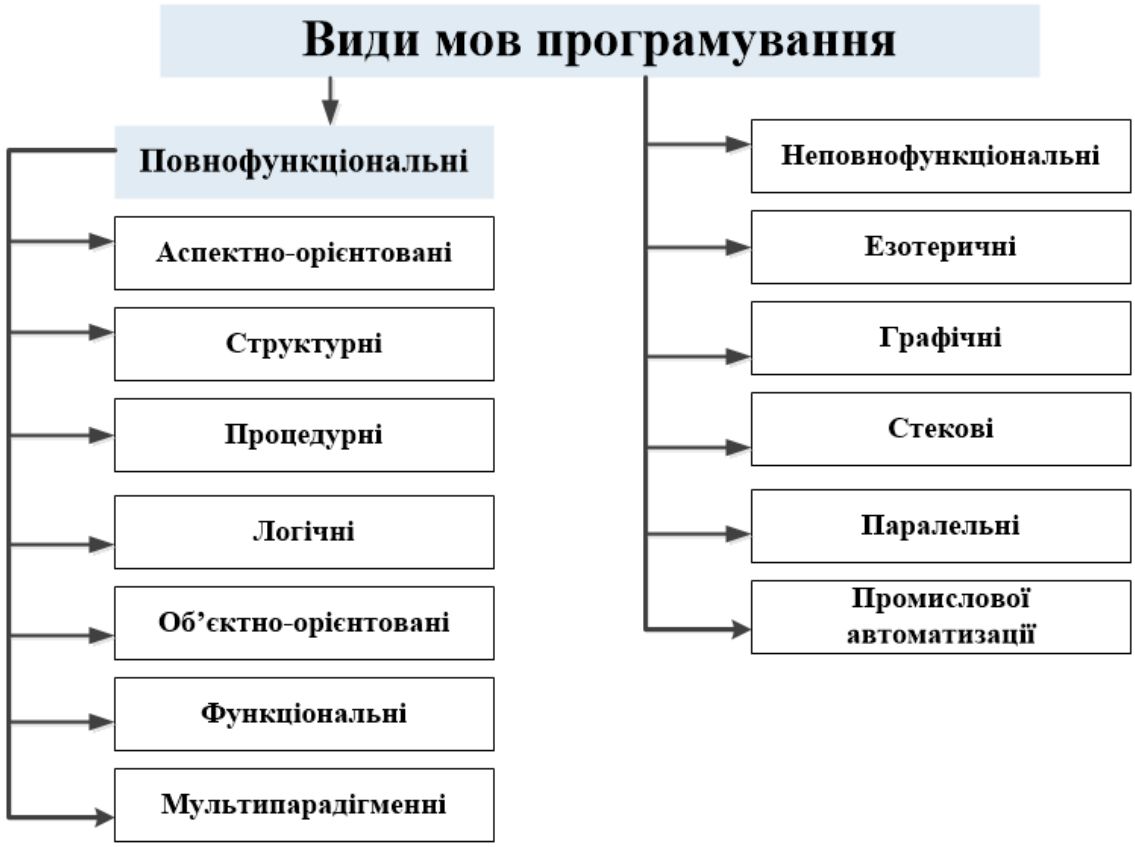


Рисунок 1.2 – Нові технології програмування

Одночасно з розвитком універсальних МВР стали розвиватися проблемно-орієнтовані МП, які вирішували економічні завдання (**COBOL**), задачі реального часу (**Modula-2, Ada**), символьної обробки (Snobol), моделювання (**GPSS, Simula, SmallTalk**), чисельно-аналітичні завдання (**Analitic**) та інші. Ці спеціалізовані мови дозволяли більш адекватно описувати об'єкти і явища реального світу, наближаючи мову програмування до мови фахівця в проблемній області.

Іншим напрямом розвитку МП є створення мов надвисокого рівня (МНВР). Мовою високого рівня програміст задає процедуру (алгоритм) одержання результату по відомим вихідним даним, тому вони називаються процедурними МП. На МНВР програміст задає відношення між об'єктами в програмі, наприклад систему лінійних рівнянь, і визначає, що потрібно знайти, але не задає як отримати результат. Такі мови ще називають непроцедурними, тому що сама процедура пошуку рішення вбудована в мову (в його інтерпретатор). Такі мови використовуються, наприклад, для вирішення завдань штучного інтелекту (**Lisp, Prolog**) і дозволяють моделювати розумову діяльність людини в процесі пошуку рішень. До

непроцедурних мов відносять і мови запитів систем управління базами даних (QBE, SQL).

Класифікація мов програмування

На підставі зазначеного вище, МП можна класифікувати за такими ознаками.

1. За ступенем орієнтації на специфічні можливості ЕОМ, МП діляться так:
   - машинно-залежні;
   - машинно-незалежні.

До машинно-залежних МП відносяться машинні мови, асемблерні і автокоди, які використовуються в системному програмуванні. Програма на машинно-залежному ЯП може виконуватися тільки на ЕОМ даного типу. Програма на машинно-незалежній МП після трансляції на машинну мову стає машинно-залежною. Ця ознака МП визначає мобільність одержуваних програм (можливість перенесення на ЕОМ іншого типу).

2. За ступенем деталізації алгоритму отримання результату МП діляться так:
   - мови низького рівня;
   - мови високого рівня;
   - мови надвисокого рівня.
3. За ступенем орієнтації на рішення певного класу задач:
   - проблемно-орієнтовані;
   - універсальні.
4. За можливістю доповнення новими типами даних і операціями:
   - які розширяються;
   - які не розширяються.
5. За можливістю управління реальними об’єктами і процесами:
   - мови систем реального часу;
   - мови систем умовного часу.
6. За способом отримання результату:
   - процедурні;
   - непроцедурні.
7. За типом вирішуваних завдань:
   - мови системного програмування;

– мови прикладного програмування.

8. Непроцедурні мови за типом вбудованої процедури пошуку рішень діляться на:

– реляційні;
– функціональні;
– логічні.

Розглянута схема класифікації дозволяє кожній МП привласнити одну з ознак кожного класу.

Тенденції розвитку мов програмування

Розглянута схема класифікації МП дозволяє зробити висновок про те, що МП володіють певною спеціалізацією, тому необхідно розглянути тенденції розвитку класів МП. Мови системного програмування, на яких створюються операційні системи, транслятори і інші системні програми, розвиваються в напрямку підвищення їх рівня та незалежності від ЕОМ. На сьогоднішній день майже 90 % системного програмного забезпечення створюється не на мові асемблера, а на мові «**C**». Наприклад, операційна система Unix практично повністю написана на цій мові. Мова «**C**» дозволяє отримувати програми, які можна порівняти за своєю ефективністю з програмами, написаними на мові асемблера. Правда, обсяг програм виходить більше, але зате ефективність їх створення набагато вища.

Машинна незалежність досягається використанням стандарту мови, що підтримується всіма розробниками трансляторів, і використанням так званих крос-систем для еквівалентного перетворення програм з однієї мови низького рівня на іншу.

Іншим напрямком є підвищення рівня самої машинного мови. Наприклад, відомі Lisp-машини, в яких машинною мовою є мова Lisp (реалізована апаратно). Іншим прикладом є ЕОМ п’ятого покоління з машинною мовою штучного інтелекту Prolog.

МВР розвиваються в напрямі підтримки технологій програмування, забезпечення низькорівневих операцій (рівня асемблера), забезпечення нових інформаційних технологій і незалежності від середовища реалізації. Слід сказати, що за своїми можливостями МВР поступово зближуються і програмістові на «**C**» все важче стає сперечатися про переваги цієї мови з програмістом, працюючим на мові Basic.

Тотальний бум переживає технологія об’єктно-орієнтованого програмування (ООП) у наш час: практично всі сучасні МВР підтримують ООП. Та й всі сучасні програмні системи побудовані на принципах ООП, і сьогодні кожен студент, який програмує знає, що таке інкапсуляція, успадкування і поліморфізм. Для позначення факту підтримки ООП мови отримують приставку **Object (ObjectPascal**) або інші (**C ++).**

Windows, мережі ЕОМ, сервери, бази даних і Internet, як основа нових інформаційних технологій, чинять сильний вплив на сучасні МП. Для підтримки **Windows** створюються системи візуального програмування з приставкою **Visual**, наприклад: **Visual C ++, Visual Basic** та інші. Для роботи з базами даних (БД), мережами і Інтернет в МП включаються спеціальні внутрішні або зовнішні кошти.

З появою технології **World Wide Web (WWW)** Інтернет набув великої популярності. Було створено багато допоміжних мов для роботи з Інтернетом, оформлення сайтів, доступу до ресурсів. Набула поширення мова, що інтерпретується, проста та легко переносима на інші платформи мова **Perl**. Вона використовується для написання різних за складністю додатків і скриптів. Значну роль у розвитку і функціонуванні Інтернету відіграє мова **Java**. Для написання основи сторінок Web-сайтів використовується **HTML** (мова розмітки гіпертекстів). Вона має всі можливості розмітки і оформлення тексту, графіки. Для більшої привабливості та функціональності сайту використовуються сценарії, написані на **JavaScript**, які виконуються на стороні Web-браузера користувача. В основному сценарії використовуються для поліпшення зовнішнього вигляду сайту та вирішення дрібних завдань. Сценарії, написані на мові PHP, виконуються на стороні сервера і посилають в браузер користувача вже оброблену інформацію. Вони використовуються для створення динамічних HTML-сторінок, гостьових книг, форумів, опитувань. Скрипти, написані переважно на **Perl**, **C/C ++,** виконуються на стороні сервера і залежать від конкретних дій користувача. Вони мають призначення подібне до сценаріїв **PHP**.

Популярність мов штучного інтелекту за останні десять років, на жаль, помітно впала. Це пов’язано, насамперед із психологічними проблемами, яких зазанають програмісти при використанні цих мов. Наприклад, у найпотужнішій мові **Lisp** програма має дуже складну для розуміння списочну структуру і невеликий за обсягом проєкт дуже швидко

виходить з під контролю. У мові Prolog програміст повинен точно знати логіку роботи вбудованої машини логічного висновку, а робота програми залежить від структури і вмісту бази знань (БЗ). Якщо з проектуванням програми і структури БЗ програміст справляється, то для заповнення БЗ він повинен бути експертом в предметній області або тісно контактувати з експертом і отримувати у нього підказки, а те й інше є складним завданням.

Макрос

Мова, що є засобом для заміни послідовності символів, які описують виконання необхідних дій ЕОМ на більш стислу форму, називається макрос (засіб заміни). Макрос здебільшого призначений для того, щоб скоротити запис вихідної програми. Компонент програмного забезпечення, що забезпечує функціонування макросів, називається «**Макропроцесор**». На макропроцесор надходить макровизначальний і вихідний текст. Макрос однаково може працювати, як з програмами, так і з даними.

Історія створення і розвитку VBA

Мова Basic була розроблена професорами Дартмутського коледжу Дж. Кемені і Т. Курцем у 1965 році як засіб навчання і роботи непрофесійних програмістів. Його призначення визначено в самій назві, яке є абревіатурою слів **Beginner's All-purpose Symbolic Instruction Code** (багатоцільова мова символічних інструкцій для початківців) і при цьому в дослівному перекладі означає «Базовий». Однак парадокс полягав у тому, що, будучи дійсно вельми простим засобом програмування, абсолютно непридатним в ті часи для вирішення серйозних завдань, Basic представляв собою якісно нову технологію створення програм в режимі інтерактивного діалогу між розробником і комп'ютером. Вірніше, Basic була праобразом сучасних систем програмування. Інша справа, що рішення такого завдання на техніці тих років було можливо тільки за рахунок максимального спрощення мови програмування і використання транслятора типу «інтерпретатор».

Із цих самих причин **Basic** здебільшого застосовувався на міні та мікро-ЕОМ, які в 70-ті роки XX сторіччя мали оперативну пам'ять, обсяг якої здається сьогодні просто нереальним (4–32 тис. байт). Швидкий розвиток систем на основі **Basic** розпочався з появою на початку 80-х років

персональних комп'ютерів, продуктивність і популярність яких росте останнім часом небаченими темпами. На початку 90-х років XX сторіччя компанія Microsoft почала активну боротьбу за просування на ринок своєї нової операційної системи (**ОС**) **Windows** (замість **MS-DOS**). Однак, як відомо, користувачі працюють не з ОС, а з програмами, які працюють в її середовищі, тому швидкість зміни платформи в основному визначається темпами появи відповідних прикладних програм. Однак зміна операційних систем являє серйозну проблему і для програмістів, так як їм потрібно було освоювати нову технологію розробки програм. На той момент популярною думкою було те, що **ОС Windows** пред'являє більш високі вимоги до кваліфікації програміста.

У 1991 році під гаслом «тепер і починаючі програмісти можуть легко створювати додатки для **Windows**» з'явилася перша версія нового інструментального засобу **Microsoft Visual Basic (VB).** В той момент Microsoft досить скромно оцінювала можливості цієї системи, орієнтуючи її, перш за все, на категорію початківців і непрофесійних програмістів. Основним завданням було випустити на ринок простий і зручний інструмент розробки в тоді ще досить новому середовищі Windows, програмування в якому було проблемою і для досвідчених фахівців. Тим більше що конкуренція з боку інших розробників мов програмування під ОС Windows на той момент була не висока.

Дійсно, VB 1.0 на той момент був більше схожий не на робочий інструмент, а на діючий макет майбутнього середовища розробки. Його принципове нововведення полягало в реалізації ідей подієво-керованого і візуального програмування в середовищі Windows, які досить радикально відрізнялися від класичних схем розробки програм. За загальним визнанням VB став родоначальником нового покоління інструментів, званих сьогодні засобами швидкої розробки програм (**Rapid Application Development – RAD**). Сьогодні ця ідеологія вважається звичною, але тоді вона створювала серйозні проблеми (в тому числі чисто психологічного плану) для програмістів. Проте, число VB-користувачів зростала, причому багато в чому за рахунок величезної популярності попередника даної мови програмування – **QuickBasic**. При цьому **VB** швидко посилювався як за рахунок розвитку середовища програмування, так і за рахунок включення професійних елементів мови і проблемно-орієнтованих засобів. І до моменту випуску в 1995 році **VB** версії 4.0 ця система була вже визнаним і

одним з найпоширеніших інструментів створення широкого класу додатків. На початку 90-х років намітилася чітка тенденція включення в програми, призначені для кінцевого користувача, засобів внутрішнього програмування, які повинні були вирішувати завдання настройки і адаптації цих пакетів для конкретних умов їх застосування. Наприкінці 1993 р **Microsoft** оголосила про намір створити на основі **VB** нову універсальну систему програмування для прикладних програм, яка отримала назву **Visual Basic for Applications (VBA**) – **VB** для додатків. Цілком зрозуміло, що реалізацію цього проекту вона почала з власних офісних пакетів.

Перший варіант **VBA** 1.0 з'явився в складі **MS Office** 4.0, але лише в програмах **Excel 4.0 і Project** 6.0. В інших же додатках – **Word** 6.0 і **Access** 2.0 були власні варіанти Basic. Більш того, VBA 1.0 досить сильно відрізнявся (причому маючи ряд істотних переваг) від універсальної тоді тоді системи VB 3.0. Якісний перелом настав у кінці 1996 року з випуском **MS Office** 97, в якому була реалізована єдине середовище програмування **VBA** 5.0, включене в програми **Word, Excel** і **PowerPoint**. Крім того, **VBA** 5.0 використовувала той же самий мовний механізм і середовище розробки, що й універсальна система **VB** 5.0. До складу випущеного **MS Office** 2000 увійшла відповідно версія **VBA** 6.0, яка використовувалася в шести програмах – **Word, Excel, PowerPoint, Access, Outlook, FrontPage**. В результаті останнім часом Microsoft позиціонує свій пакет **MS Office** не просто як набір прикладних програм, а як комплексну платформу для створення бізнес-додатків, які вирішують широке коло спеціалізованих завдань користувачів. Одночасно **Microsoft** активно просуває **VBA** в якості галузевого стандарту для управління програмованими додатками, оголосивши про можливість його ліцензування. Сьогодні вже більше ста провідних світових фірм-розробників прикладних програм придбали ліцензії на нього і включають **VBA** до складу своїх програмних продуктів, в тому числі і американська компанія **ESRI** – виробник геоінформаційних систем. Освоєння механізму програмування **VBA**, реалізованого в офісному додатку відкриває користувачам можливість використання отриманих знань і навичок при роботі з десятками і сотнями інших програм, в тому числі і тих, яких поки що не існує. Почавши зі створення найпростіших макрокоманд, при бажанні можна в рамках одного інструментарію стати професіоналом, які розробляють програмні системи

будь-якої складності. Буквально десять – двадцять років тому в усьому світі було не більше двох мільйонів програмістів. Сьогодні їх налічується близько десяти мільйонів, з них не менше 70 % використовують хоча б один з інструментів VB або VBA.

2 АЛГОРИТМІЗАЦІЯ У ПРОГРАМУВАННІ

Одним з базових понять інформатики та обчислювальної техніки є поняття алгоритму як деякого правила перетворення інформації. Алгоритм містить рекомендації щодо того, які операції по обробці даних і в якій послідовності треба виконати, щоб отримати рішення задачі.

Алгоритм – це точне розпорядження, яке визначає обчислювальний процес, що веде від змінюваних початкових даних до вишукуваного результату. Тобто це послідовність дій, спрямованих на досягнення чи на вирішення поставленого завдання за кінцеве число кроків.

Детермінованість (визначеність) – однозначність результату обчислювального процесу при заданих початкових даних

Дискретність – розчленованість обчислювального процесу на окремі елементарні дії (кроки), можливість виконання яких не викликає сумніву

Масовість – забезпечення вирішення будь-якої задачі з класу однотипних

Результативність – забезпечення отримання результату через кінцеве число кроків.

Для запису алгоритмів використовуються різні способи:

- словесно-формульний опис;
- алгоритмічний запис на умовній мові;
- графічна схема алгоритму (блок-схема).

Причина різноманітності засобів запису алгоритмів – відмінності в системах команд виконавців і в способах сприйняття виконавцем команд алгоритму.

Мова програмування – це спосіб запису алгоритму, орієнтований на виконання його системою програмування комп’ютера. Найбільш часто алгоритми зображуються у вигляді графічних або блок-схем.

Алгоритм великої складності зазвичай представляється за допомогою схем двох видів:

- узагальненої схеми алгоритму, яка розкриває загальний принцип функціонування алгоритму і основні логічні зв’язки між окремим модулями;
- детальної схеми алгоритму, яка зображує зміст кожного елемента

узагальненої схеми.

Типи алгоритмів і їх графічне зображення

Типові структуру основних типів алгоритмів:

– **лінійний** (на основі структури слідування), характеризується послідовнім виконання команд;

– **розгалужений** (на основі структур розгалуження і вибору), характеризується тим, що в ході виконання, рішення задачі йде тільки по одному з напрямків, вибір якого залежить від виконання заданої умови;

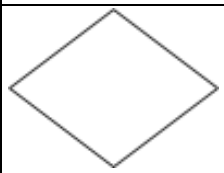
– **циклічній** (на основі структури цикл), характеризується багаторазовим повторення певної групи дій.

Типові структури хороші тим, що всі вони мають одну точку входу и одну точку виходу. Типові структури можуть буті вкладені один в одного. Зупинимося на графічному описі алгоритму, який називається **блок-схемою**.

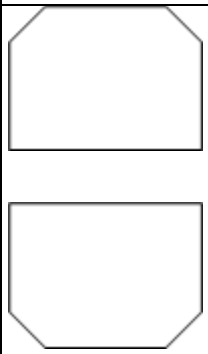
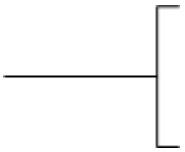
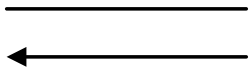
Блок-схема алгоритму – графічне зображення алгоритму у вигляді пов'язаних між собою за допомогою стрілок (ліній переходу) і блоків – графічних символів, кожен з яких відповідає одному кроку алгоритму. Усередині блоку дається опис відповідної дії.

Тобто блок-схема – графічне представлення алгоритму. Для позначення різних дій в блок-схемі використовують різні геометричні фігурі. У таблиці 2.1 наведені найбільш уживані фігурі блок-схем.

Таблиця 2.1 – Основні символи схем алгоритмів

Назва	**Графічне зображення**	**Функція символа**
1	2	3
Процес		Виконання операції або групи операцій, в результаті яких змінюється значення, форма подання або розташування даних
Рішення		Вибір напряму виконання алгоритму або програми, залежно від деяких змінних умов
Введення – виведення		Перетворення даних у форму, придатну для обробки (введення) або відображення результатів обробки (виведення)

Продовження таблиці 2.1

1	2	3
Границі циклу		Відображення початку та закінчення циклу
Початок / закінчення		Початок, закінчення, переривання процесу обробки даних
Коментар		Зв’язок між елементом схеми та поясненням
Лінія потоку		Покажчик послідовності зв’язків між символами

Цей спосіб має низку переваг завдяки наочності, що забезпечує, зокрема, високу «читаність» алгоритму і явне відображення управління в ньому. Цей спосіб виявився дуже зручним засобом зображення алгоритмів і набув широкого поширення в науковій та навчальній літературі.

Структурна схема алгоритму – графічне зображення алгоритму у вигляді схеми пов’язаних між собою за допомогою стрілок (ліній переходу) блоків – графічних символів, кожен з яких відповідає одному кроку алгоритму. Усередині блоку дається опис відповідної дії. Графічне зображення алгоритму широко використовується перед програмуванням завдання внаслідок його наочності, тому що зорове сприйняття зазвичай полегшує процес написання програми, її коригування при можливих помилках, осмислення процесу обробки інформації.

Принцип програмування «зверху вниз» вимагає, щоб блок-схема поетапно конкретизувалася і кожен блок «розписувався» до елементарних операцій. Але такий підхід можна здійснити при вирішенні нескладних завдань. При вирішенні скільки-небудь серйозного завдання блок-схема «розповзеться» до такої міри, що її неможливо буде охопити одним поглядом. Блок-схеми алгоритмів зручно використовувати для пояснення

роботи вже готового алгоритму, при цьому в якості блоків беруться дійсно блоки алгоритму, робота яких не вимагає пояснень. Блок-схема алгоритму повинна служити для спрощення зображення алгоритму, а не для ускладнення.

Основні поняття і класифікація обчислювальних процесів

При вирішенні завдання на цифровий ЕОМ зазвичай виконуються наступні основні етапи:

– постановка завдання, де формулюється мета дослідження і пред'являються вимоги до її вирішення по точності і часу;

– розробка математичної моделі розв'язуваної задачі; вибір чисельного методу для наближеного розв'язання задачі з урахуванням сформульованих вимог;

– розробка обчислювального алгоритму;

– програмування, тобто запис алгоритму розв'язання задачі на певній алгоритмічній мові;

– налагодження програми;

– обчислення;

– обробка результатів обчислення.

З перелічених етапів найбільш трудомісткими є четвертий і п'ятий, тобто розробка обчислювального алгоритму та програмування, тому їм буде приділено особливу увагу.

Під обчислювальним алгоритмом розуміють послідовну сукупність формалізованих вказівок, що переводять вихідні дані в процесі обчислення у вишукувані результати.

До алгоритму завдання висуваються такі вимоги:

– детермінованість, тобто алгоритм повинен бути чітким, зрозумілим і однозначним;

– масовість – можливість використання алгоритму для вирішення типових завдань;

– результативність, тобто алгоритм повинен складатися з кінцевої кількості арифметичних і логічних дій, що призводять до вишукуваного результату. Результативність багато в чому залежить від правильного завдання вихідних даних.

Алгоритм зазвичай задається в змістовному, блок-схемному або операторному вигляді. При блок-схемному описі обчислювального

алгоритму слід користуватися умовними графічними позначеннями.

Лінійні обчислювальні процеси

Принцип розв'язання задач на ЕОМ полягає в поданні задачі у вигляді програми обробки даних і реалізації її на ЕОМ з метою отримання кінцевих результатів при заданих початкових даних. Розробка програми заснована на можливості представить будь-яку задачу у математичній формі: математичними формулами та іншими співвідношеннями, якими задаються правила обробки даних. Програма визначає послідовність реалізації цих правил на ЕОМ.

Алгоритм – це чітко задана послідовність дій, які повинні бути виконані для вирішення задачі. Тобто це план роботи комп'ютера. Одну ціль можна досягти різними способами (алгоритмами).

Словесний опис алгоритму

1. Уведення
2. Обчислення
3. Виведення

Програма на алгоритмічній мові є формою представлення алгоритму. Вона включає опис алгоритму і даних на алгоритмічній мові. Опис даних – це вказівки на виділення певного об'єму пам'яті і форму подання даних для їхнього збереження і обробки.

Алгоритмічна мова – програма для комп'ютера.

Таким чином, для розв'язання задач на ЕОМ необхідно: розробити математичну модель задачі, визначити початкові дані і кінцеві результати, розробити алгоритм і програму на алгоритмічній мові, виконати переклад програми з алгоритмічної на машинну мову за допомогою транслятора, виконати машинну програму на ЕОМ та отримати результати.

Обчислювальний процес називається лінійним, якщо усі його операції виконуються послідовно в порядку їхнього запису (рис. 2.1).

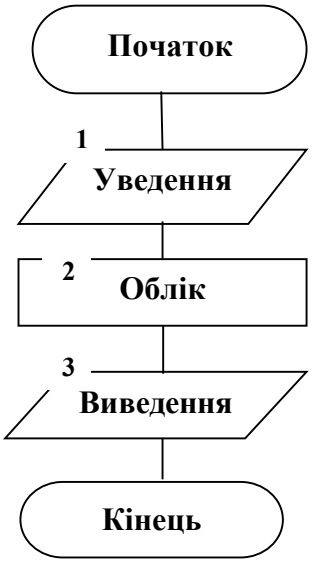


Рисунок 2.1 – Алгоритм лінійного обчислювального процесу

Лінійний алгоритм (лінійна структура) – це такий алгоритм, в якому всі дії виконуються послідовно один за одним і тільки один раз. Блок-схема являє собою послідовність блоків, які розташовуються зверху вниз у порядку їх виконання. Первинні і проміжні дані не впливають на напрямок процесу обчислення. Лінійні обчислювальні процеси характеризуються послідовним виконанням операторів програми і блоків обчислювального алгоритму. Вони, зазвичай, є складовою частиною циклічного або розгалуженого обчислювального процесу.

При освоєнні матеріалу по розробці структурних схем складну математичну залежність доцільно розбивати на окремі частини і оформляти їх у вигляді самостійних блоків. Наприклад, при обчисленні функції за наступною формулою:

$$a = arcsin(AC - \sqrt{A^2 + B^2 - C^2} /(A^2 + B^2)) \qquad (2.1)$$

Частина (фрагмент) обчислювального процесу ($A^2 + B^2$) доцільно оформити у вигляді самостійного блоку. При розробці обчислювального алгоритму доцільно прагнути до мінімізації кількості обчислювальних операторів і до мінімального використання обсягу пам’яті, не погіршуючи при цьому точність обчислення обраної послідовності обчислювальних дій.

Лінійним називається такий обчислювальний процес, в якому самостійні етапи обчислень виконуються в послідовності їх запису, тобто в

природному порядку. Кожна операція є самостійною, незалежною від будь-яких умов. Лінійні обчислювальні процеси мають місце при обчисленні арифметичних виразів. Наприклад – скласти схему алгоритму роботи найпростішого касового апарату при визначенні вартості товару, яка визначається за формулою (2.2), а блок-схема цього процесу подана на рисунку 2.2.

$$\text{Сума} = \text{Ціна} * \text{Кількість} \qquad (2.2)$$

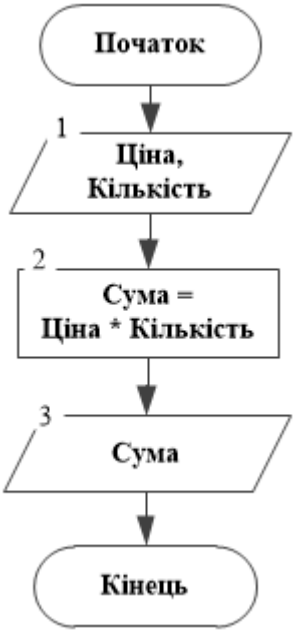


Рисунок 2.2 – Схема алгоритму роботи касового апарату

Принципи візуального програмування

Революційним кроком у програмуванні, який кардинально полегшив життя програмістів, стала поява візуального програмування, що виник внаслідок появи Visual Basic і знайшов втілення в системах **C ++ Builder, Microsoft Visual C ++, Delphi** фірми **Borland** і інших. Візуальне програмування дозволило звести проєктування призначеного для користувача інтерфейсу до простих і наочних процедур, які дають можливість за хвилини або години зробити те, на що раніше йшли місяці роботи. Додаток, побудований за принципами об’єктної орієнтації – це не послідовність якихось операторів, додаток перестав бути якимось жорстким алгоритмом. Об’єктно-орієнтована програма – це сукупність об’єктів і способів їх взаємодії.

Об’єкт – це сутність, здатна зберігати свій стан і забезпечує набір операцій (поведінку) для перевірки і зміни цього стану. Зовнішнє управління об’єктом здійснюється через обробники подій. Ці обробники звертаються до методів і властивостей об’єкта. Початкові значення даних об’єкта можуть задаватися також у процесі проектування установкою різних властивостей. В результаті виконання методів об’єкта можуть генеруватися нові події, які сприймаються іншими об’єктами програми або користувачем. Окремим і головним об’єктом при такому підході в багатьох випадках можна вважати користувача програми. Він же служить основним, але не єдиним, джерелом подій, які керують додатком.

Середовищем взаємодії об’єктів є повідомлення, що генеруються в результаті різних подій. Події наступають, перш за все, внаслідок дій користувача – переміщення курсору миші, натискання кнопок миші або клавіш клавіатури. Але події можуть наступати і в результаті роботи самих об’єктів. У кожному об’єкті визначено безліч подій, на які він може реагувати. У конкретних примірниках об’єкта можуть бути визначені обробники якихось із цих подій, які і визначають реакцію даного екземпляра об’єкта. До написання цих обробників, часто вельми простих, і зводиться основне програмування при розробці графічного інтерфейсу користувача за допомогою Visual Basic.

Створення обробників подій у Visual Basic

Користувач працюєте в інтегрованому середовищі розробки (ICP або Integrated development environment – IDE). Середовище надає вам форми (в додатку їх може бути кілька), на яких розміщуються компоненти. Зазвичай це віконні форми, хоча вони можуть бути невидимими. На форму за допомогою миші переносяться і розміщуються піктограми елементів управління (рис. 2.3).

За допомогою простих маніпуляцій мишею можна змінювати розміри і розташування цих елементів. При цьому весь час в процесі проектування видно результат – зображення форми і розташованих на ній компонентів. Не потрібно багато разів запускати додаток і вибирати найбільш вдалі розміри вікна і компонентів для перегляду форми. Результати проектування видно, навіть не компілюючи програму (транслюючи програму, складену на мові високого рівня, в еквівалентну програму на низкорівневій мові, близькій до машинного коду) негайно

після виконання якоїсь операції за допомогою миші.

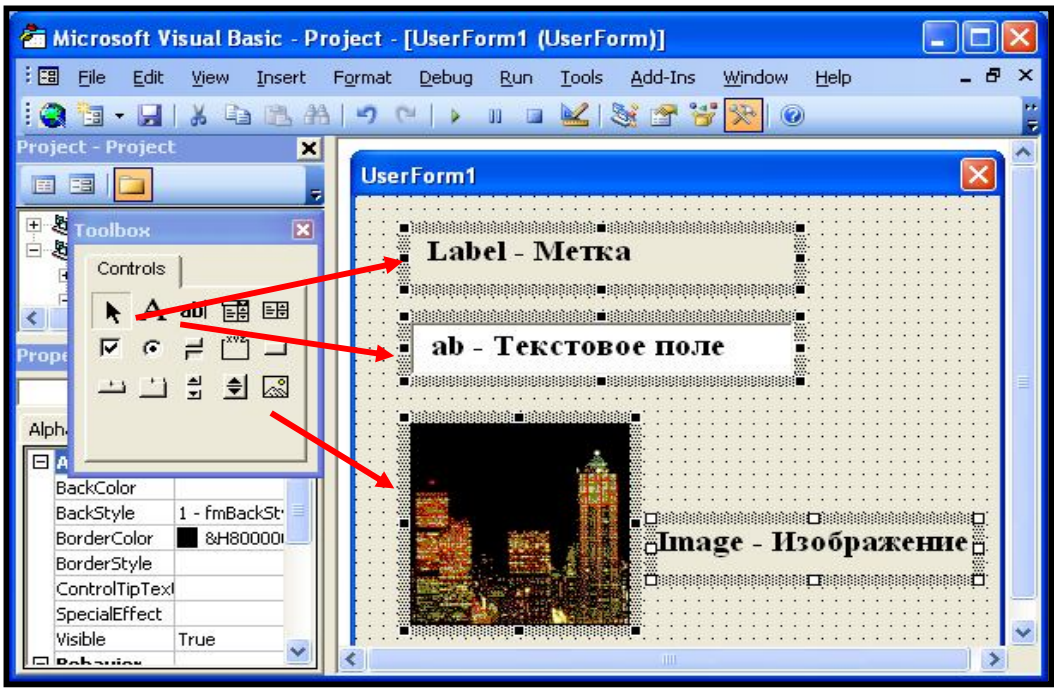


Рисунок 2.3 – Створення віконної форми

Однак переваги візуального програмування не зводяться до цього. Найголовніше полягає в тому, що під час проектування форми і розміщення на ній компонентів VB автоматично формує коди програми, включаючи в неї відповідні фрагменти, що описують даний компонент. Потім у відповідних діалогових вікнах користувач може змінити задані за замовчуванням значення властивостей цих компонентів, а при необхідності написати обробники подій.

Таким чином, проєктування зводиться фактично до розміщення компонентів на формі, завдання деяких властивостей і написання, при необхідності, обробників подій. Компоненти можуть бути візуальні, видимі при роботі додатка, і невізуальні, що виконують ті чи інші службові функції. Візуальні компоненти відразу видно на екрані в процесі проєктування в такому самому вигляді, у якому їх побачить користувач під час виконання програми. Це дозволяє дуже легко вибрати місце їх розташування та їх дизайн – форму, розмір, оформлення, текст, колір. Невізуальні компоненти видно на формі в процесі проектування у вигляді піктограм, але користувачеві під час виконання програми вони не видимі, хоча і виконують для нього за кадром вельми корисну роботу.

Принципи розробки інтерфейсу

Основна перевага Windows-додатків – їх стандартний вид. Якщо користувач навчився працювати в одному з них, то можна вважати, що він освоїть будь-який інший. Всі додатки Windows використовують одні і ті ж принципи роботи (рис. 2.4). Існує кілька загальних принципів розробки додатків під Windows. Дотримуючись їх, розробник отримує низку істотних переваг:

– по-перше, розроблений додаток виглядає професійно;

– по-друге, воно легко освоюється кінцевим користувачем і узгоджується з іншими додатками;

– по-третє, додаток має сучасний дизайн.

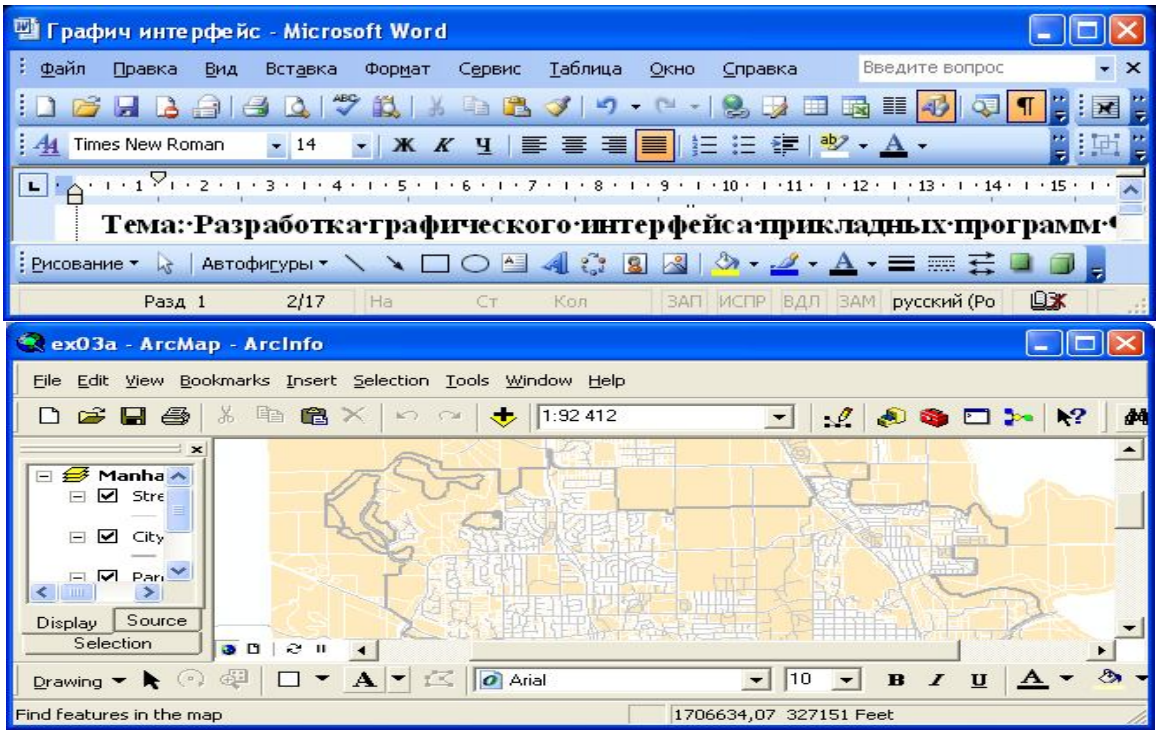


Рисунок 2.4 – Приклад схожості виду графічних інтерфейсів **MS Word** і **ArcMap**

Інтерфейс – це зовнішня оболонка додатку разом з програмами управління доступом та іншими прихованими від користувача механізмами управління, яка надає можливість працювати з документами, даними і іншою інформацією, яка зберігається в комп'ютері.

Головна мета будь-якого додатка – забезпечити максимальну зручність і ефективність роботи з інформацією: базами геоданих, графікою

або зображення, тому інтерфейс є найважливішою частиною будь-якого додатку.

Якісно розроблений інтерфейс гарантує зручність роботи користувача з додатком. Проектування інтерфейсу – процес циклічний. На цьому етапі розробки додатку бажано частіше спілкуватися з користувачами і замовниками додатку для розробки найприйнятніших по ефективності та зручності і зовнішнього вигляду інтерфейсного рішення.

Під час роботи в **ArcMap**, як і в більшості інших додатків, доводитися стикатися з такими елементами інтерфейсу, як діалогові вікна. Діалогові вікна використовуються для отримання інформації для введення і виведення повідомлень та даних. Одним з об'єктів візуалізації **VBA** є форма (Userform), який призначений для користувача. Призначені для користувача форми – це діалогові вікна інтерфейсу процедур **VBA**. За допомогою діалогових вікон користувач може ефективно передавати дані в процедури і одержувати результати їхньої роботи. Призначені для користувача форми дають можливість створювати діалогові вікна в додатках, що розробляються, і розміщувати у вікнах елементи управління. В **VBA** є дві функції – **Msgbox** та **Inputbox**, які дозволяють відображати прості діалогові вікна, не створюючи форму користувача. Ці вікна можна видозмінювати, використовуючи керовані ними параметри, але вони не мають тих широких і ефективних можливостей та опцій, які надають форми користувача.

VBA пропонує великі можливості, які можна використовувати при створенні призначених для користувача діалогових вікон та програмування елементів управління.

Саме тому що проектування інтерфейсу процес ітераційний, на цьому етапі розробки програми бажано частіше спілкуватися з користувачами і замовниками додатку для вибору найбільш прийнятних по ефективності, зручності і зовнішньому вигляду інтерфейсних рішень. Вибір того чи іншого типу інтерфейсу залежить від складності додатка, оскільки кожен з них має свої недоліки, обмеження і призначений для вирішення певних завдань. При цьому, необхідно відповідати на ряд питань: яка кількість документів буде оброблятися в проектованому додатку і їх тип, чи мають дані деревоподібну структуру, чи буде потрібна панель інструментів, яка кількість документів обробляється за певний інтервал часу (наприклад, за день) тощо. Тільки після цього можна вибирати конкретний тип інтерфейсу.

При розробці інтерфейсу необхідно керуватися такими принципами:

1. **Стандартизація**. Слід використовувати стандартні, перевірені багатьма програмістами і користувачами інтерфейсні рішення. Для **Visual Basic** це рішення **Microsoft**. Під рішеннями маються на увазі дизайн форми, розташування елементів управління на формі, їх взаємне розташування, значки на кнопках управління, назви команд меню.

2. **Зручність і простота роботи**. Інтерфейс повинен бути інтуїтивно зрозумілим. Бажано, щоб всі дії легко запам'ятовувалися і не вимагали тривалих процедур: виконання додаткових команд, зайвих натискань на кнопки, виклику проміжних діалогових вікон.

3. **Зовнішній дизайн**. Не можна, щоб інтерфейс стомлював зір. Він повинен бути розрахований на тривалу роботу користувача з додатком протягом дня.

4. **Неперевантаженість форми**. Форми повинні бути оптимально завантажені елементами управління. При необхідності можна використовувати вкладки або додаткові сторінки форм.

5. **Групування**. Елементи управління в формі необхідно групувати за змістом, використовуючи елементи групування: рамки, фрейми.

6. **Розрідженість об'єктів форми**. Елементи управління слід розташовувати на певній відстані, а не впритул один до одного. Для виділення елементів управління можна організувати порожні ділянки в формі.

У наш час для додатків, які розробляються в середовищі **Windows** за допомогою **Visual Basic**, використовуються три типи інтерфейсу:

1. Однодокументний **SDI** (**Single-Document Interface**).
2. Багатодокументний **MDI** (**Multiple-Document Interface**).
3. Інтерфейс типу «Провідник» (**Explorer**).

Однодокументний інтерфейс

Однодокументний інтерфейс – це тип інтерфейсу, в якому надається можливість роботи тільки з одним документом в одному вікні. Прикладом може служити редактор **Microsoft Paint**. Інтерфейс включає такі елементи:

- головне меню;
- панелі інструментів з елементами управління;
- вікно додатка для розміщення елементів управління даними;
- елементи управління для роботи з даними;

– рядок стану.

Багатодокументний інтерфейс

Головна особливість MDI інтерфейсу – можливість багаторазово відкривати форму одного виду документа для декількох різних за змістом документів. Прикладом може служити редактор Microsoft Word. Інтерфейс включає такі елементи:

– головне меню;

– панелі інструментів з елементами управління;

– головне вікно програми (MDI-вікно);

– дочірні вікна програми;

– елементи управління для роботи з даними, розташованими в дочірніх вікнах;

– рядок стану.

Інтерфейс типу «Провідник»

Інтерфейс типу **Провідник** розробляється для доступу до ієрархічних деревовидних структур, тобто до таких, де зустрічається вкладеність. Прикладом вкладеності можуть служити папки і файли. Файли розташовані в папках, які, в свою чергу, лежать у вищестоящих папках і так далі. Прикладом такого інтерфейсу є провідник **Windows**. У цьому файловому менеджері наочно видно структуру зберігання папок і файлів, що утворює ієрархічне дерево. За своєю суттю це аналог інтерфейсу SDI, розроблений спеціально для деревоподібних структур. Інтерфейс програми типу провідник містить наступні елементи:

– головне меню;

– вікно додатка для розміщення елементів управління даними;

– ієрархічний список елементів деревоподібної структури. Це можуть бути папки і файли, документи, якщо вони організовані в ієрархічну структуру;

– елементи управління для роботи з даними: кнопки, поля, прапорці;

– рядок стану.

Засоби проєктування інтерфейсу

Важко переоцінити роль інтерфейсу користувача в сучасному програмному забезпеченні. Вдалий чи невдалий інтерфейс багато в чому визначає успіх чи невдачу всієї розробки. Зручність інтерфейсу – поняття досить суб'єктивне. Те, що подобається одному користувачеві, зовсім не підійде для іншого. Те, що забезпечує комфортну роботу службовців одного відділу, буде зустрінуте недоброзичливо в іншому. Тому при проектуванні системи найкраще заздалегідь включати в неї можливість налаштування інтерфейсу під потреби різних категорій користувачів.

Створення додатків практично неможливо без використання елементів управління, так як вони дозволяють користувачеві взаємодіяти з цими додатками. За допомогою елементів управління (меню **Вид / Панелі інструментів / Елементи управління**) можна автоматизувати роботу з документами. Представлені елементи управління дозволяють розробити практично будь-які інтерфейси для користувачів. Панель елементів управління – основний робочий інструмент при візуальній розробці форм додатка у VBA (рис. 2.5).

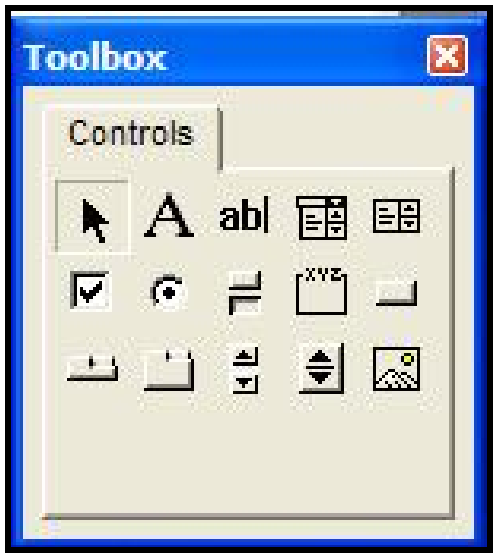


Рисунок 2.5 – Панель елементів управління в VBA

Елементи управління

У складі панелі елементів управління містяться основні елементи управління форм – написи, текстові поля, кнопки, списки та інші елементи інтерфейсу (об'єкти), для швидкого візуального проектування макета форми. У таблиці 2.2 представлені назви і зовнішній вигляд об'єктів

(елементів управління), які вбудовуються у форму.

Таблиця 2.2 – Назви і зовнішній вигляд елементів управління

Тип об'єкта	Призначення	Графічний вигляд
Label	Напис	
TextBox	Текстове поле	
CommandButton	Кнопка	
ChekBox	Прапорець	
OptionButton	Перемикач	
Frame	Група (рамка)	
ListBox	Список	
ComboBox	Поле зі списком	
Image	Рисунок	
ShipButton	Лічильник	
ScrollBar	Полоса прокрутки	

Кнопка – елемент, при натисканні на який виконується та чи інша команда.

Прапорець – елемент, який незалежно від інших може перебувати в трьох станах: включеному, вимкненому (стани можуть бути визначені як користувачем, так і програмістом) і неактивному (визначається програмою).

Поле (для введення тексту) – дає користувачеві можливість ввести текстову інформацію з метою подальшої обробки в програмі.

Перемикач – елемент, який може знаходитися у включеному, вимкненому і неактивному станах. На відміну від прапорця, якщо один з перемикачів в групі включений, решта включені бути не можуть.

Список – засіб вибору одного варіанта з декількох представлених. Допускається прокручування списку, якщо не всі його елементи видно одночасно.

Поле зі списком – вставляє об'єкт, що є поєднанням списку і поля. Користувач може або вибрати потрібне значення зі списку, або ввести його в поле самостійно.

Лічильник – може передавати в програму своє значення від 1 до 100.

Полоса прокрутки – може передавати в програму число, що дорівнює відстані в пунктах від її початку. Верхня межа відстані не обмежена.

Напис (поле відображення тексту) – відображає незмінний текст, наприклад підпис до малюнка.

Рисунок – у цей елемент можна вставити рисунок з файлу.

Розміщення елементів управління на формі здійснюється наступним чином:

– на панелі **Toolbox** натиснути кнопку необхідного елемента управління. Наприклад, для розміщення на формі текстового напису необхідно додати елемент **Label**;

– клацанням-протяжкою лівою кнопкою миші вибраний елемент перемістити з панелі на форму у місце можливого розміщення об’єкта;

– для деяких елементів, для створення потрібного розміру, необхідно не відпускаючи кнопку миші, намалювати рамку необхідного розміру на формі і відпустити кнопку миші.

У разі необхідності задані при розміщенні елемента управління на формі розміри можна змінити пізніше, використовуючи мишу або вікно властивостей об’єкта.

Деякі загальні властивості елементів управління

Нижче представлені загальні властивості, які притаманні більшості елементів управління (ЕУ).

Позиція. Позицію ЕУ визначають чотири властивості **Left, Top, Height, Width**. Властивості **Left, Top** задають координати верхнього лівого кута елементу управління. Властивості **Height** і **Width** – його висоту і ширину. Відлік в системі координат ведеться зверху вниз по осі Y і зліва направо по осі X.

Колір. Управління кольором здійснюється за допомогою властивостей **BackColor, FillColor, ForeColor**, яким за замовчуванням призначаються стандартні кольори. Властивість **BackColor** відповідає за колір фону і вибирається в діалоговому вікні настройки кольору. Властивість **ForeColor** визначає колір, який використовується для відображення тексту і графіки в елементі управління, а властивість **FillColor** встановлює колір заповнення нарисованих об’єктів.

Параметри шрифта. Вид шрифта в елементах управління

вибирається за допомогою установки значень властивості **Font**.

Доступність і видимість ЕУ. Для цього використовуються дві властивості – **Enabled** і **Visible**. Властивість **Enabled** визначає, чи буде елемент управління реагувати на події (**True**) чи ні (**False**). Властивість **Visible** дозволяє зробити ЕУ невидимим – необхідно зробити значення **False**.

Властивість **Name** є ідентифікатором елемента управління. Воно відіграє особливу роль. Помилка при його завданні призводить до серйозних наслідків (невиконання коду програми). Тому спочатку завжди задається ім’я ЕУ, і лише потім пишеться для нього процедура обробки події.

3 ТИПИ ДАНИХ У ПРОГРАМУВАННІ

Змінні, типи даних і константи

Операція – конструкція в мовах програмування, аналогічна за записом математичним операціям, тобто спеціальний спосіб запису деяких дій. Найбільш часто застосовуються арифметичні, логічні та рядкові операції. На відміну від функцій, операції часто є базовими елементами мови і позначаються різними символами пунктуації, а не алфавітно-цифровими, вони мають спеціальний синтаксис і нестандартні правила передачі аргументів.

Фактично операція – це та сама функція, але записується вона особливим чином, тому необхідно мати можливість визначати операції для довільних типів даних таким же чином, як і методи – щоб можна було працювати з ними таким же чином, як і з елементарними типами. Ця можливість називається «перевантаження операцій» і присутня в більшості мов 4–5 поколінь. У таких мовах транслятор фактично підставляє замість виконання операції виклик відповідної їй функції.

Операції поділяються за кількістю прийнятих аргументів на:

- унарні – один аргумент (заперечення, унарний мінус);
- бінарні – два аргументи (додавання, віднімання, множення);
- тернарні – три аргументи («умова: Вираз 1, Вираз 2»).

Арифметичні операції задаються наступними символами (табл. 3.1).

Таблиця 3.1 – Перелік арифметичних операцій

Знак	Операція
+	додавання
–	віднімання
*	множення
/	ділення
\	ділення без залишку
mod	ділення по модулю

Для запису операторів в мові VBA використовуються ключові слова (зарезервовані слова), такі як **Dim, As, New, If, Then, Else, While, End**. При явному оголошенні змінні визначаються за допомогою оператора **Dim**.

Dim Name As String – явне оголошення типу даних. Типи даних, які використовуються в Visual Basic, наведені в таблиці 3.2.

Таблиця 3.2 – Перелік типів даних

Тип даних	**Розмір у байтах**	**Опис типу**	**Діапазон значень**
1	**2**	**3**	**4**
Byte	1	Однобайтове ціле число	від 0 до 255
Integer	2	Коротке ціле число	від -32768 до 32767
Long	4	Довге ціле	± 21474833648
Double	8	Подвійне з плаваючою точкою	від $5{,}0\times10^{-324}$ до $1{,}7\times10^{308}$
Single	4	Десяткові числа одинарної точності	від $1{,}5\times10^{-45}$ до $3{,}4\times10^{38}$
String	1 байт на каждый символ	Рядок змінної довжини	від 0 до 2×10^{9}
Boolean	2	Логічне значення **True** або **False**	**True** або **False**
Currency	8	Число в грошовому форматі	від 922 337 203 685 477.5808 до +922 337 203 685 477.5807
Date	8	Дата і час – дані типу **Date**, представлені числом із плаваючою крапкою. Ціла частина цього числа показує дату, а дрібна частина – час	Дата від 1 січня 100 г. до 31 грудня 9999 г.
Variant	>=16 байтов	Будь-який тип даних	Число – **Double**, рядок – **String**
Object	4	Посилання на об’єкт	Будь-яке посилання на об’єкт

У мові VBA є універсальні типи даних, тобто здатні зберігати як число, так і рядок, дату і будь-який інший тип інформації. Наприклад, клітинка в таблиці може містити що завгодно і спочатку, програма не знає який в ній тип даних зберігається. Крім того, в самій програмі може використовуватися тип даних **Variant**, який так само може містити будь-яке значення будь-якого типу.

Щоб визначити, який тип даних в клітинці або в змінній типу

Variant, можна скористатися кількома способами. Головне призначення **VBA** – обробка даних. Деякі дані зберігаються в об'єктах, наприклад, діапазонах робочих аркушів. Інші дані зберігаються в створених користувачем змінних.

Змінна є іменованим місцем зберігання даних в пам'яті комп'ютера. Змінні можуть містити дані різних типів – від простих логічних, або булевих значень (**True** або **False**) до великих значень з подвійною точністю. Значення присвоюється змінній за допомогою оператора рівності.

VBA підтримує кілька обмежень в іменуванні змінних:

– можна використовувати в назвах символи букв, числа і деякі розділові знаки, але першою в імені змінної завжди повинна вводитися буква;

– **VBA** не розрізняє регістри;

– не можна використовувати в іменах пробіли або точки.

Щоб зробити імена змінних легкими для читання, використовують змішаний регістр (**InterestKate** а не **interestkate**) або вводять символ підкреслення (**Interest_Kate**).

Спеціальні символи оголошення типу **#, $,%, &, !** не застосовуються в імені змінної. Назви змінних обмежені довжиною 254 символа. Не допускається застосовувати в якості назв змінних або процедур зарезервовані слова, тобто такі слова, які використовуються **VBA**.

У **VBA**, як і в будь-якій іншій мові програмування, змінні і константи використовуються для зберігання будь-яких значень. Як і випливає з назви, змінні можуть змінюватися, константи ж зберігають фіксовані значення.

Наприклад, константа π зберігає значення 3,14159265 ... Число π не змінюватиметься в процесі виконання програми, але все ж зберігати таке значення зручніше як константу. У той же час можна використовувати змінну **sVAT_Rate** для зберігання ставки ПДВ на товари, що купувалися. Величина змінної **sVAT_Rate** може змінюватися залежно від того, що який товар купувався.

Типи даних

Всі змінні і константи відносяться до певного типу даних. У таблиці нижче наведені типи даних, що використовуються у **VBA**, з описом і діапазоном можливих значень: Очевидно, що правильно вибираючи тип даних, можна використовувати пам'ять більш економно (наприклад, вибрати тип даних **Integer** замість **Long** або **Single** замість **Double**). Однак, використовуючи більш компактні типи даних, потрібно уважно стежити за тим, щоб в коді не було спроб вмістити в них не пропорційно великі значення.

Оголошення змінних і констант

У програмі, перш ніж використовувати змінну, її треба оголосити. У мові Visual Basic для опису змінних використовуються спеціальні оператори. Змінними позначаються значення, що змінюються в ході виконання програми. У мові VBA кожна змінна характеризується своїм типом (**type**). Від типу залежить, які значення приймає змінна. Оголошення змінної визначає її тип. У найпростішому випадку синтаксис оголошення змінної складається з одного виразу, що включає:

- ключове слово **Dim**;
- ім'я змінної;
- ключове слово **As**;
- назва типу змінної.

Оголошення змінної в кодах програми виглядає так.

Dim *Ім'яЗмінної* **As** тип

В одному рядку можна оголосити відразу кілька змінних.

Dim *Ім'яЗмінної1* **As** тип1, *Ім'яЗмінно2* **As** тип2, ...

Ім'я змінної має бути унікальним у межах програмного коду.

ПРИКЛАД.

Dim *intЧисло* **As Integer**, *Рядок* **As String**

де змінна *intЧисло* – змінна цілого типу, може приймати значення від -32 768 до 32 767;

змінна *Рядок* – змінна строкового типу.

Dim *strA* (1 to 33) **As String**

Оголошення строкового масиву, що містить 33 елементи.

У разі якщо тип змінної не оголошений, то змінна вважається змінною універсального типу **Variant** і для її зберігання відводиться в пам'яті максимальна кількість байтів (16), що призводить до неефективного використання пам'яті і уповільнення роботи програми.

Недоліком мови **VBA** є те, що вона дозволяє програмістам обходитися без оголошення змінних. Такий порядок використовується в мові **VBA** за замовчуванням. У міру необхідності в програму додаються нові змінні без їх оголошення. Однак додавання змінних без їх оголошення призводить до виникнення ряду помилок. Наприклад, якщо випадково неправильно набрано ім'я змінної, програма просто сприйме це як додавання нової змінної з іншим ім'ям. У процесі виконання програми можуть виникнути проблеми. Крім того, якщо змінна не оголошена, для неї автоматично буде обраний тип **Variant**. В результаті доступні системні ресурси будуть використані вкрай нераціонально.

При оголошенні змінних **VBA** автоматично розпізнає неправильно набрані імена змінних, що допомагає при усуненні помилок. Крім того, для оголошених змінних можна підібрати такі типи даних, які найкращим чином будуть підходити для збереження оброблюваних значень.

Говорячи про змінні у **VBA**, варто згадати ще один дуже важливий момент. Якщо оголошено змінну, але не присвоєно їй якесь значення, то вона ініціалізується значенням за замовчуванням:

- текстові рядки – ініціалізуються порожніми рядками;
- числа – значенням 0;
- змінні типу **Boolean** – **False**;
- дати – 30 грудень 1899.

Змінні, значення яких не змінюються в процесі виконання програми, називаються константами.

Const *Ім'яКонстанти* [**As Тип**] = Значення Константи

Константи (**constant**) – це дані, які не змінюються в ході виконання програми. При оголошенні констант обов'язково відразу вказується їх значення.

ПРИКЛАД.

Const *iMaxCount* = 5 000

У мові VBA розрізняють константи двох типів:

– літеральние (**literal**);

– символьні (**symbolic**).

Літеральні константи набираються безпосередньо в кодах програми і можуть являти собою числа і рядки (текстові значення). Строкові константи повинні братися в подвійні лапки.

ПРИКЛАД.

МійРядок = «чотири»

МоєЧисло = 4

Що стосується числових значень, VBA розпізнає їх експоненціальний запис і (з використанням префікса &H) та шістнадцятирічний запис.

1.2E5 – число 1,2 помножене на 10 в п’ятому ступені, тобто 120000

&HFE – еквівалентно десятковому числу 254

Шістнадцятирічна система обчислення заснована на використанні 16 цифр, тоді як звична нам десятирічна система побудована на застосуванні десяти цифр. У шістнадцятирічній системі цифри від 0 до 9 позначають перші десять цифр (як і в десятковій системі), а для позначення наступних шести цифр використовуються перші шість букв латинського алфавіту (від A до F).

Символьні константи мають власні імена. Якщо в коді програми виникає необхідність у використанні значення такої константи, слід вказати її ім’я. Визначаються символьні константи за допомогою ключового слова **Const**.

Const *Ім’яКонстанти = ЗначенняКонстанти*

Як ім’я константи, можна використовувати будь-яке ім’я, допустиме в мові VBA. Значення константи – це літеральна, числова або строкова константа, яка визначає значення створюваної символьної константи.

ПРИКЛАД.

Const МОЄІМ’Я = «*Петро*»

Const МІЙВІК = 30

Символьні константи мають дві переваги. Ім’я такої константи може і повинно бути описовим, що значно спрощує читання кодів програми. Однак більш важливим є наступне – якщо виникне необхідність змінити значення константи в кодах програми, то досить буде зробити це тільки в тому місці, де вона була оголошена. Згідно із загальноприйнятою угодою, назви констант набирають виключно прописними буквами, тоді як імена інших елементів, наприклад змінних, складаються з комбінацій великих і малих літер.

Наприклад, ім’я **ВІДСОТКОВАСТАВКА** відповідає константі, а ім’я **ВідсотковаСтавка** – змінній. Для програми VBA це правило не має ніякого значення, проте, керуючись ним, можна досить легко відрізнити в кодах програми імена констант від імен змінних.

Якщо константа може використовуватися у всіх модулях і процедурах, то спочатку потрібно написати зарезервоване слово **Public**.

Public Const *MyConst* **As Integer** = 59

Якщо константа може використовуватися тільки в даному модулі, то спочатку потрібно записати слово **Private**.

Private Const *Stavka* **As Single** = 0,007

Виділення значень, які не відповідають оголошеному типу змінної

Якщо оголосити змінну певного типу і спробувати привласнити їй дані іншого типу, то з’явиться помилка, і якщо не виправити її, можна отримати збій в роботі програми. На перший погляд, це може здатися гарною причиною, щоб не оголошувати змінні, але насправді, ніж раніше з’ясується, що одна з змінних отримала не ті дані, які повинна була отримати – тим краще. Інакше, якщо програма продовжить роботу, результати можуть виявитися неточними, а знайти причину помилок буде набагато важче. Можливо також, що макрос буде виконано. В результаті помилка залишиться непоміченою і робота продовжиться з невірними даними. У зв’язку з цим, невірний тип даних бажано виявляти і якомога раніше виправляти такі помилки в коді. З цих причин при написанні макросу VBA рекомендується оголошувати всі змінні.

Присвоєння

Змінна може отримати або змінити значення за допомогою оператора присвоювання.

[**Let**] *Ім'яЗмінної* = **Вираз**

ПРИКЛАД.

Визначити, які значення отримають змінні після виконання наступної програми.

Dim Example **As Single**, a **As Integer**, b **As Byte**, c **As Integer**
Private Sub Command1_Click ()
Example = 5,8
a = 5
b = 7
c = a + b
a = 100
End Sub

Після виконання програми значення змінних будуть такі a = 100, b = 7, c = 12, Example = 5,8.

Область дії змінних і констант

Кожна оголошена змінна або константа має свою обмежену область дії, тобто обмежену частину програми, в якій ця змінна існує. Область дії залежить від того, де було зроблено оголошення змінної або константи. Візьмемо, наприклад, змінну **sVAT_Rate**, яка використовується в функції **Total_Cost**. У наступному прикладі розглянуті два варіанти області дії змінної **sVAT_Rate**, оголошеної в двох різних позиціях в модулі.

ПРИКЛАД.
Option Explicit
Dim *sVAT_Rate* **As Single**
Function Total_Cost() **As Double**
...
End Function

Якщо змінна *sVAT_Rate* оголошена в самому початку модуля, то областю дії цієї змінної буде весь модуль (тобто змінна *sVAT_Rate* буде розпізнаватися усіма процедурами в цьому модулі). Отже, якщо в функції **Total_Cost** зміній *sVAT_Rate* буде присвоєно деяке значення, то наступна

функція, виконувана в межах цього ж модуля, буде використовувати змінну *sVAT_Rate* з цим же значенням. Однак, якщо буде викликана якась функція, розташована в іншому модулі, то для неї змінна *sVAT_Rate* буде не відома.

Option Explicit
Function Total_Cost() **As Double**
Dim *sVAT_Rate* **As Single**
...
End Function

Якщо змінна *sVAT_Rate* оголошена на початку функції **Total_Cost**, то її область дії буде обмежена тільки цією функцією (тобто в межах функції **Total_Cost**, можна буде використовувати змінну *sVAT_Rate*, а за її межами – ні). При спробі використовувати *sVAT_Rate* в іншій процедурі, компілятор VBA повідомить про помилку, так як ця змінна не була оголошена за межами функції **Total_Cost** (за умови, що використаний оператор **Option Explicit**).

У поданому вище прикладі змінна оголошена на рівні модуля за допомогою ключового слова **Dim**. Однак, буває необхідно, щоб оголошеними змінними можна було користуватися в інших модулях. У таких випадках для оголошення змінної замість ключового слова **Dim** потрібно використовувати ключове слово **Public**.

До речі, для того щоб оголосити змінну на рівні модуля, замість ключового слова **Dim** можна використовувати ключове слово **Private**, яке вкаже на те, що дана змінна призначена для використання тільки в поточному модулі. Для оголошення констант також можна використовувати ключові слова **Public** і **Private**, але не замість ключового слова **Const**, а разом з ним. У наступних прикладах показано використання ключових слів **Public** і **Private** в застосуванні до змінних і до констант.

ПРИКЛАД.
Option Explicit
Public *sVAT_Rate* **As Single**
Public Const *iMax_Count* = 5 000
...

У цьому прикладі ключове слово **Public** використано для

оголошення змінної *sVAT_Rate* і константи *iMax_Count*. Областю дії оголошених таким чином елементів буде весь поточний проєкт. Це означає, що *sVAT_Rate* і *iMax_Count* будуть доступні в будь-якому модулі проєкту.

Option Explicit
Private *sVAT_Rate* **As Single**
Private Const *iMax_Count* = 5 000
...

У цьому прикладі для оголошення змінної *sVAT_Rate* і константи *iMax_Count* використано ключове слово **Private**. Областю дії цих елементів є поточний модуль. Це означає, що *sVAT_Rate* і *iMax_Count* будуть доступні у всіх процедурах поточного модуля, але не будуть доступні для процедур, які перебувають в інших модулях.

Інструкція Option Explicit

Щоб активувати оголошення змінних, потрібно додати вираз **Option Explicit** в кожному модулі перед початком всіх процедур. Для активізації необхідності оголошення змінних можна скористатися меню **Tools / Options (Сервис / Параметры**). У діалоговому вікні на вкладці **Editor (Редактор**) необхідно встановити прапорець опції **Require Variable Declaration (Объявление переменных обязательно**).

При використанні інструкції **Option Explicit** необхідно явно описати всі змінні за допомогою інструкцій **Dim, Private, Public, ReDim** або **Static**. При спробі використовувати неописане ім'я змінної виникає помилка під час компіляції. Для автоматичної вставки в програму інструкції **Option Explicit** відкрийте редактор VBA (швидкі клавіші **Alt** + **F11**) увійдіть в меню **Сервис / Параметры** в діалоговому вікні **Параметры** та встановіть прапорець **Явное описание переменных**.

Таким чином виключається поява в програмі помилки в результаті не вірно записаного імені змінної.

Наприклад, використовуючи в коді змінну з ім'ям ***sVAT_Rate***, можна допустити друкарську помилку і, привласнюючи значення цій змінній, записати: «***VATRate = 0,175***». Очікується, що з цього моменту, змінна ***sVAT_Rate*** повинна містити значення 0,175, але, звичайно ж, цього не відбудеться. Якщо ж включений режим обов'язкового оголошення всіх

використовуваних змінних, то компілятор VBA відразу ж вкаже на помилку, тому що не знайде змінну ***VATRate*** серед оголошених.

Перетворення типів даних

Перетворення в VBA містить процедури, які перетворюють значення одного типу в значення іншого типу. Так, функція **Val** перетворює рядки в число, а **Str** перетворює числа в рядок. **Hex, Oct** перетворять числа в іншу систему числення. **CBool, CByte, CCur, CDbl, CDec, CInt, CLng, CSng, CStr, CVar, Fix, Int** – один тип даних в інший (табл. 3.3).

Таблиця 3.3 – Функції перетворення типів даних

Функція	Тип результату	Префікс типу	Опис типу
CBool (x)	Boolean	bln	Логічне значення
CByte (x)	Byte	byt	Однобайтне ціле число (від 0 до 255)
CInt (x)	Integer	int	Коротке ціле число (до ±32 тис.)
CCur (x)	Currency	cur	Число з фіксованою точкою (грошовий тип)
CDate (x)	Date	dtm	Дата та час
CDbl (x)	Double	dbl	Число с плаваючою точкою подвійної точності (цілі, дробові 10±348)
CLng (x)	Long	lng	Довге ціле (± 2 млрд.)
CSng (x)	Single	sng	Число с плаваючою точкою одинарної точності
CStr (x)	String	str	Текстовий рядок
CVar (x)	Variant	var	Будь-яке значення з поданих вище

Організація діалогу з користувачем

Функція **InputBox** використовується для отримання інформації від користувача. Синтаксис цієї функції такий.

InputBox (*Message, Title, Default, y, x*)

де *Message* – повідомлення-підказка;

Title – заголовок;

Default – значення за замовчуванням;

y, x – координати розташування вікна.

Функція **MsgBox** використовується для видачі інформації користувачеві. Синтаксис цієї функції такий.

MsgBox (*Msg, Style, Title, Help, Ctxt*)

де *Msg* – повідомлення для користувача;

Style – вид кнопки (наприклад, vbOKOnly – відображається тільки кнопка «OK»);

Title – заголовок;

Help – ім’я файлу довідки (необов’язковий аргумент).

Якщо цей аргумент вказано, необхідно вказати номер розділу довідки.

4 ПРОГРАМУВАННЯ ЗАВДАНЬ ІЗ РОЗГАЛУЖЕННЯМ

На практиці часто зустрічаються задачі, в яких в залежності від початкових умов або проміжних результатів необхідно виконати обчислення по одним або іншим формулами. Такі завдання можна описати за допомогою алгоритмів, що розгалужуються. У таких алгоритмах вибір напрямку продовження обчислення здійснюється за підсумками перевірки заданої умови. Процеси, що розгалужуються описуються оператором **IF** (умова).

Процес з розгалуженням – з декількох варіантів вибирають тільки один, причому вибір залежить від умови. Кожен окремий напрям обчислень в такому процесі називається гілкою обчислення. Вибір здійснюється перевіркою виконання логічного умови. У кожному конкретному випадку обробки даних обчислювальний процес виконується лише по одній гілці, а виконання інших – виключається (рис. 4.1).

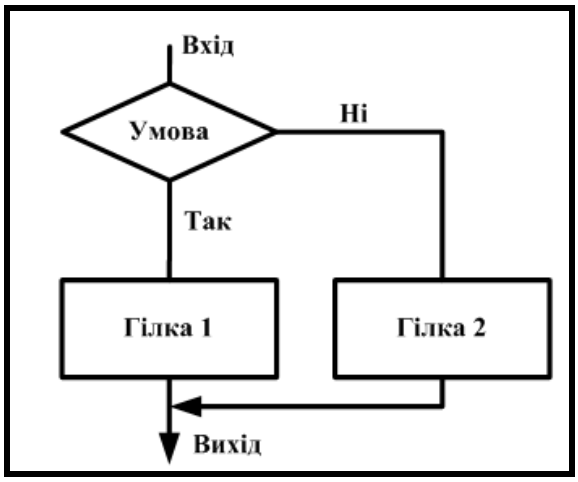


Рисунок 4.1 – Зображення алгоритму з розгалуженням

Розгалужений процес, що включає в себе дві гілки, називається **простим**, більш двох гілок – **складним**. Складний розгалужений процес можна здійснити за допомогою простих розгалужених процесів.

Напрями розгалуження вибираються логічної перевіркою, в результаті якої можливі дві відповіді:

- «так» – умова виконана;
- «ні» – умова не виконана.

Кожна гілка, за якою здійснюються обчислення, повинна сприяти завершенню обчислювального процесу.

Наприклад, якщо студент іногородній, то йому надається гуртожиток, якщо ні, то гуртожиток не надається.

Умовний оператор – це приклад складного оператора, тобто всередині можуть розташовуватися інші оператори. При застосуванні умовного оператора потрібно пам'ятати такі правила:

– умовний оператор визначається ключовим словом **IF**;

– після цього ключового слова указується умова – вираз, при обчисленні якого отримують логічне значення;

– за умовою йде ключове слово **THEN**.

Умова показує, яка група операторів виконується за ключовим словом **THEN**, якщо значення умови – істина, то виконується одна група операторів, якщо помилкова, то виконується інша група операторів.

Кінець розгалуження визначається спеціальним оператором, який складається з двох ключових слів – **END IF**.

Структури операторів дозволяють управляти послідовністю виконання програми. Без операторів управління усі оператори програми будуть виконуватися зліва направо та зверху вниз. Але іноді потрібно багато разів виконувати деякий набір інструкцій автоматично, або вирішувати задачу по-іншому залежно від значення змінних або параметрів, заданих користувачем під час виконання. Для цього служать конструкції управління та цикли.

VBA підтримує такі конструкції умовного оператора:

– **If . . . Then**

– **If . . . Then . . . Else**

– **Select Case**

Конструкція If . . . Then

Конструкція **If . . . Then** застосовується, коли необхідно виконати один або групу операторів залежно від деякої умови. Синтаксис цієї конструкції дозволяє задавати її в одному рядку або в декількох рядках програми:

If умова **Then** вираз
If умова **Then**
вираз
End If

Звичайно умова є простим порівнянням, але вона може бути будь-яким виразом зі значенням що обчислюється. Це значення інтерпретується як **False** (брехня), якщо воно нульове, а будь-яке ненульове розглядається як **True** (істина). Якщо умова істинна, то виконуються усі вирази, що стоять після ключового слова **Then**. Для умовного виконання одного оператора можна використовувати як синтаксис для одного рядка, так і синтаксис для декількох рядків (блокову конструкцію).

Наступні два оператори еквівалентні:

```
If anyDate < Now Then anyDate = Now
If anyDate < Now Then
  anyDate = Now
End If
```

Необхідно зазначити, що синтаксис оператора **If** . . . **Then** для одного рядка не використовує оператор **End If**. Щоб виконати послідовність операторів, якщо умова істинна, потрібно використовувати блокову конструкцію **If** . . . **Then** . . . **End If**.

```
If anyDate < Now Then
  anyDate = Now
  Timer.Enabled = False
End If
```

Якщо умова помилкова, то оператори після ключового слова **Then** не виконується, а управління передається на наступний рядок (або рядок після оператора **End If** у блоковій конструкції).

Конструкція If . . . Then . . . Else

Ця конструкція визначає декілька блоків операторів, один з яких буде виконуватися залежно від умови.

```
If умова 1 Then
   вираз 1
ElseIf умова 2 Then
   вираз 2
```

. . .
Else
вираз-n
End If

При виконанні спочатку перевіряється умова 1. Якщо вона помилкова, VBA перевіряє наступний вираз 2 і т. д., поки не знайде істинної умови. Знайшовши її, VBA виконує відповідний блок операторів і потім передає управління конструкції, наступної за оператором **End If**. В дану конструкцію можна включити блок оператора **Else**, який VBA виконує, якщо не виконана жодна з умов.

Конструкція **If** . . . **Then** . . . **ElseIf** насправді всього лише спеціальний випадок конструкції **If** . . . **Then** . . . **Else**. Зазаначимо, що в даній конструкції може бути будь-яке число блоків **ElseIf**, або навіть жодного. Блок **Else** можна включати незалежно від присутності або, навпаки, відсутність блоків **ElseIf**.

У якості умови можна використовувати логічний вираз, що повертає значення **True** (Істина) або **False** (Брехня), або будь-який арифметичний вираз. Якщо використовується арифметичний вираз, то нульове значення цього виразу еквівалентно логічному значенню **False**, а будь-який ненульовий вираз – значенню **True**. У тому випадку, коли умова повертає значення **False**, оператор або блок операторів, поміщених між ключовими словами **Then** і **End If** і складаючих тіло скороченого оператора розгалуження, не виконуватиметься.

Якщо умова істинна, виконується перший блок операторів, розташований між ключовими словами **Then** і **Else**; в іншому випадку виконується другий блок, розташований між ключовими словами **Else** і **End If**. Іноді доводиться робити вибір однієї дії з цілої групи альтернативних дій на основі перевірки декількох різних умов. Для цього можна використовувати ланцюжок операторів розгалуження **If**...**Then**...**ElseIf.**

ПРИКЛАД.

Відсоток відхилення (*Відхилення*) фактичного виконання (Факт) геодезичної зйомки заданої ділянки місцевості від плану (План) визначається за формулою:

$$Відхилення = \begin{cases} Відсоток - 100, якщо\ Відсоток > 100 \\ 100 - Відсоток, якщо\ Відсоток < 100 \end{cases}$$

де: *Відсоток* = (*Факт* / *План*) * 100.

Якщо *Факт* = *План* (тобто *Відсоток* = 100), то *Відхилення* не обчислюється.

Блок-схема цього прикладу наведена на рисунку 4.2.

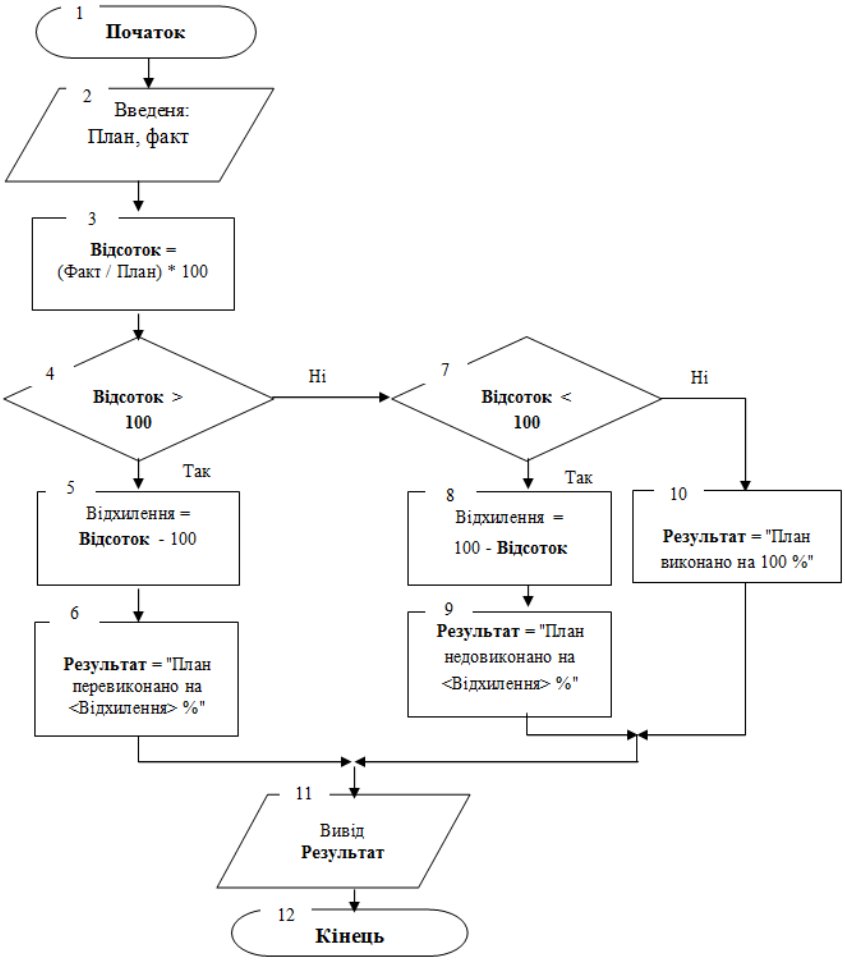


Рисунок 4.2 – Схема з декількома виходами. Гілки залежні

Конструкція Select Case

Ланцюжки операторів **If ... Then ... ElseIf** мають велику гнучкість і дозволяють вирішити всі проблеми, проте якщо вибір однієї з кількох можливостей весь час грунтується на різних значеннях одного і того ж виразу, набагато зручніше використовувати спеціально призначений для цього оператор вибору **Select Case**, що має такий синтаксис:

Select Case умова **Case** список значень1
оператори1> **Case** список значень2
оператори2 **Case** список значень3
оператори3
...
Case Else
оператори_**Else**
End Select

Вираз що перевіряється, обчислюється на початку роботи оператора **Select Case** і може повертати значення будь-якого типу, наприклад логічне, числове або строкове. Список виразів містить одне або кілька виразів, розділених комою. При виконанні оператора перевіряється, чи відповідає хоча б один з елементів цього списку виразу, що перевіряється.

У цьому випадку перевіряється, чи співпадає значення умови, що перевіряється з цією умовою.

умова1 **To**...умова2

У цьому випадку перевіряється, чи міститься значення, що перевіряється в зазначеному діапазоні значень.

Is… логічний оператор… умова

У цьому випадку умова, що перевіряється порівнюється з вказаним значенням за допомогою заданого логічного оператора (наприклад, умова **Is > = 10** вважається виконаною, якщо значення, що перевіряється не менше 10).

Якщо хоча б один з елементів списку відповідає умові, що перевіряється, то виконується відповідна група операторів і на цьому виконання оператора **Select Case** закінчується, а решта списків виразів не перевіряється, тобто в цьому випадку відшукується тільки перший відповідний елемент списку виразів. Якщо жоден з елементів всіх цих списків не відповідає значенню виразу, що перевіряється, то виконуються оператори групи **Else** (якщо така є).

Форми умовного оператора

Будь-яка зміна лінійної послідовності виконання дії називається переходом. Переходи бувають двох видів: умовні та безумовні. Умовний

перехід реалізується за допомогою умовного оператора **If**, який має блоковий і однорядковий формат запису (рис. 4.3), безумовний перехід реалізується за допомогою оператора **GoTo**.

Рисунок 4.3 – Формати запису умовного оператора

Оператор умовного переходу If

Блоковий оператор If

Залежно від конкретного завдання використовують різні його модифікації. Найпростіша – безальтернативна (усічена) форма запису. Вона застосовується тоді, коли в разі істинності умови необхідно виконати кілька програмних операторів, а при невиконанні умови ніяких дій немає.

(якщо) (тоді)
If *Умова* **Then**
Оператори при виконанні *Умови*
End If

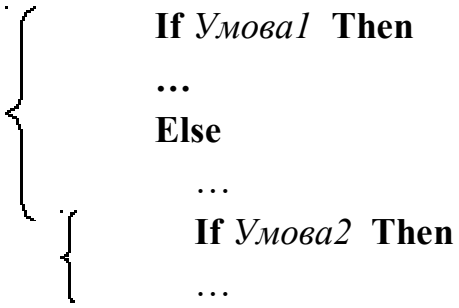
ПРИКЛАД.
If a*b <>0 Then
y=1/(a*b)
MsgBox “y=” & y
End If

Альтернативна форма оператора **If** застосовується в тих випадках, коли при істинності умови необхідно виконати один набір операторів, а при невиконанні умови – інший. У цьому випадку оператор **If** записується в наступному вигляді.

If *Умова* **Then**
Оператори при виконанні *Умови*

Else ‘**і*накше***
Оператори при не виконанні *Умови*
End If

Необхідно підкреслити що кількість умовних блоків (операторів **If**) завжди на одиницю менше числа гілок, на які розходиться обчислювальний процес. Як видно з прикладу вище, оператори **If** можуть бути вкладеними. При цьому кожен внутрішній **If** повинен повністю входити в зовнішній **If**.



If *Умова1* **Then**
…
Else
…
If *Умова2* **Then**
…
Else
…
End If
…
End If

Якщо між зовнішнім **Else** і внутрішнім **If** немає інших операторів, то їх зазвичай з’єднують в одне слово, а команду **End If**, відповідну внутрішньому **If**, виключають.

Однорядковий (лінійний) оператор If

Лінійний формат умовного оператора **If** записується в такий спосіб:

If *Умова* **Then** Оператори при виконанні *Умови* [**Else** Оператори при не виконанні *Умови*]

Лінійною форма оператора називається тому, що умовний оператор записується тільки в одному рядку, без перенесення його на інший рядок (**End If** не потрібен). Лінійний **If** застосовується рідко, зазвичай в тих випадках, якщо необхідно перевіряти не більше двох умов, і оператори

після ключових слів досить короткі.

Оператор безумовного переходу GoTo

Він використовується для передачі управління в певне місце програми без перевірки будь-яких умов і має наступний вигляд.

GoTo *Мітка*

Мітка позначає місце в програмі, куди здійснюється перехід. Ім'я мітки вибирається за звичайними правилами для імен змінних і закінчується двокрапкою. Мітка може знаходитися як до, так і після оператора **GoTo**.

ПРИКЛАД.

```
GoTo m1 ‘Перехід на рядок з міткою m1
…….
m1: y = cos(x)
```

5 ПРОГРАМУВАННЯ ЗАВДАНЬ З ЦИКЛІЧНИМИ ОБЧИСЛЮВАЛЬНИМИ ПРОЦЕСАМИ

Способи організації циклів. Структура циклічних алгоритмів

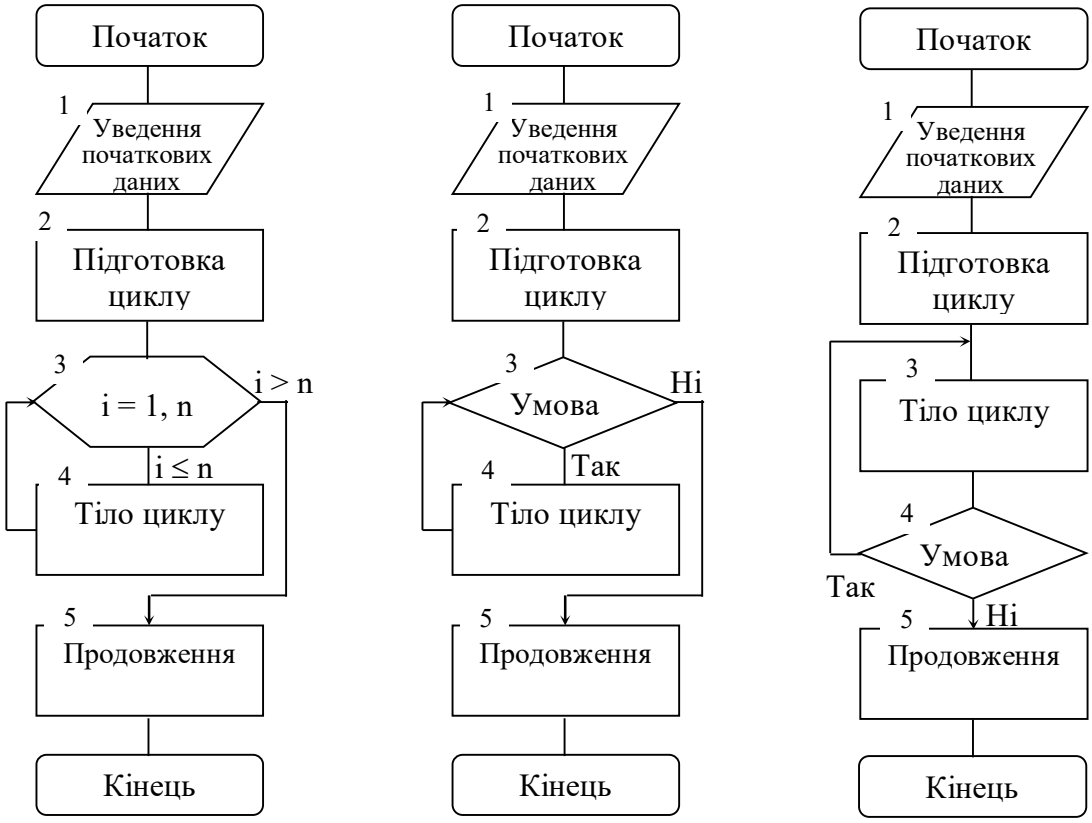
Основою розв'язання на комп'ютері прикладних задач є принцип повторення: обчислення виконується багато разів за одними формулами, але при різних початкових даних. Такі процеси називаються циклічними, а частини які повторюються – циклами.

Основу циклічних алгоритмів складають базові алгоритмічні конструкції **цикл**, а основу циклічних програм – оператори циклу.

У циклічних алгоритмах і програмах можна організувати цикл трьома способами:

- цикл з лічильником циклів (з параметром) (рис. 5.1 а);
- цикл з передумовою (рис. 5.1 б);
- цикл з післяумовою (рис. 5.1 в).

Кожному способу відповідає своя структура алгоритму. Типові структури циклічних алгоритмів наведені на рисунку 5.1.



а) б) в)

Рисунок 5.1 – Типові структури циклічних алгоритмів

У циклічних алгоритмах можна виокремити дві обов'язкові частини: підготовка циклу і цикл. Підготовка циклу (символ 2) включає привласнення початкових значень перемінним, які використовуються для організації циклу та обчислень в циклі. Змінна, яка використовується для управління циклом, називається параметром циклу.

Цикл включає такі дії: перевірку умови виконання циклу (символ 3) і тіло циклу (символ 4). Тіло циклу – це послідовність дій, які повторюються.

Алгоритми відрізняються тільки способом організації циклу. В алгоритмі з лічильником циклів (рис. 5.1а) як параметр циклу використовується змінна цілого типу. Лічильник циклів позначається символом «Модифікація» (символ 3). Наприклад, відомо, що цикл виконується **n** разів. В якості лічильника циклів використовується перемінна цілого типу, наприклад, **i**. Під час виконання циклів значення **i** змінюється від **1** до **n** з кроком **1**. При **i > n** цикл завершується. Значення **n** визначається під час підготовки циклу.

Лічильник циклів застосовується для організації циклів, якщо кількість циклів задано або можна обчислити під час підготовки циклу.

В циклах з предумовою перевірка умови повторення циклу здійснюється перед початком циклу (рис. 5.1б). В умові порівнюються поточні значення параметру циклу із заданим значенням, наприклад, **x < b**. Як параметр циклу може бути змінна будь-якого типу. Цикл виконується доти, доки справедлива умова (**Так**), інакше (**Ні**) цикл завершується. При підготовці циклу необхідно параметру циклу привласнити початкове значення, наприклад, **x = a**. В тілі циклу поточні значення параметру циклу потрібно змінювати, наприклад:

$$x = x + h,$$

де **h** – крок зміни параметру циклу.

Інакше умова буде справедлива завжди, і цикл буде повторюватися нескінченно.

У циклах із післяумовою перевірка умови повторення циклу здійснюється після циклу (рис. 5.1 в). В алгоритмах з лічильником циклів і в циклах з передумовою цикл може не виконуватися жодного разу, а в циклах із післяумовою цикл виконується хоча б один раз.

У VBA існує багатий вибір засобів організації циклів, які можна розділити на дві основні групи: цикли з умовою **Do ... Loop** і цикли з перерахуванням **For ... Next**.

Цикли типу **Do ... Loop** використовуються в тих випадках, коли заздалегідь не відомо, скільки разів має повторитися виконання блоку операторів, що становить тіло циклу. Такий цикл продовжує свою роботу до тих пір, поки не буде виконано певну умову. Існують чотири види циклів **Do ... Loop**, які розрізняються типом умови, що перевіряється і часом виконання цієї перевірки. Синтаксиси цих чотирьох конструкцій наведені в таблиці 5.1.

Таблиця 5.1 – Синтаксиси управляючих конструкцій

Синтаксис конструкції	Опис
Do While *Умова* *Оператори тіла циклу* **Loop**	Умова перевіряється до виконання групи операторів, що утворюють тіло циклу. Цикл продовжує свою роботу, поки ця умова виконується (має значення **True**), іншими словами, в цій конструкції вказується умова продовження роботи циклу
Do *Оператори тіла циклу* **Loop While** *Умова*	Умова перевіряється після виконання хоча б один раз операторів, що складають тіло циклу. Цикл продовжує свою роботу, поки ця умова залишається істиною, іншими словами, в цій конструкції вказується умова продовження роботи циклу
Do Until *Умова* *Оператори тіла циклу* **Loop**	Умова перевіряється до виконання групи операторів, що утворюють тіло циклу. Цикл продовжує свою роботу, якщо ця умова ще не виконана, і припиняє роботу, коли вона стане виконаною, іншими словами, в цій конструкції вказується умова припинення роботи циклу
Do *Оператори тіла циклу* **Loop Until** *Умова*	Умова перевіряється після виконання хоча б один раз операторів, що складають тіло циклу. Цикл продовжує свою роботу, якщо ця умова ще не виконана, а коли вона буде виконана, цикл припиняє роботу, іншими словами, в цій конструкції вказується умова припинення роботи циклу

У VBA також є два види операторів циклу з перерахуванням **For ... Next**. Дуже часто при обробці масивів, а також в тих випадках, коли потрібно повторити виконання певної групи операторів задану кількість разів, використовується цикл **For ... Next з лічильником**. На відміну від

циклів **Do ... Loop** даний цикл використовує спеціальну змінну, яка називається лічильником, значення якої збільшується або зменшується при кожному виконанні тіла циклу на задану величину. Коли значення цієї змінної досягає заданого значення, виконання циклу закінчується.

Синтаксис такого циклу має наступний вигляд (у квадратних дужках надані необов'язкові елементи синтаксичної конструкції).

For лічильник = початкове значення **To** кінцеве значення
[**Step** приріст]
Оператори тіла циклу
Next лічильник

Розглянемо ще один вид циклу **For ... Next**, часто використовуваний в VBA при обробці об'єктів, що становлять масив або сімейство однорідних об'єктів. У циклі **For Each ... Next** лічильник відсутній, а тіло циклу виконується для кожного елемента масиву або сімейства об'єктів.

Синтаксис такого циклу має наступний вигляд (у квадратних дужках надані необов'язкові елементи синтаксичної конструкції).

For Each елемент **In** совокупність
Оператори тіла циклу **Next** [елемент]

Тут **елемент** – це змінна, яка використовується для посилання на елементи родини об'єктів; **сукупність** – ім'я масиву або сімейства.

ПРИКЛАД.

Видача на друк списку всіх полів для всіх таблиць поточної відкритої бази даних. Програмний код вирішення даного завдання наведено нижче.

```
Public Sub EnumerateAllFields( )
Dim MyBase As Database
Dim t As TableDef, f As Field
    Set MyBase=DBEngine.Workspaces(0).Databases(0) For Each t In
MyBase.TableDefs
        Debug.Print "Таблиця: " & t.Name For Each f In t.Fields
        Debug.Print "Поле: " & f.Name Next f Next t
```

```
    Set  MyBase = Nothing
End  Sub
```

В операторах **Dim** змінна **MyBase** оголошена як об'єкт «База даних», а змінні **t** і **f** – як визначення таблиці і поле таблиці відповідно. Оператор **Set** призначає змінній **MyBase** поточну відкриту базу даних. Далі для кожного визначення виконується вивід на друк назви таблиці, а потім вкладений цикл такого ж типу, друкує назви всіх її полів.

Циклічні процеси можна класифікувати залежно від кількості й складності циклів на **прості** (один цикл) і **складні** (цикл у циклі або розгалуження в циклі). У свою чергу прості цикли поділяються залежно від способу виходу із циклу на **арифметичні** й **ітераційні**. Розглянемо кожен з таких циклів.

Арифметичний цикл

Циклічний процес, у якому заздалегідь відомо кількість повторень циклу, називається арифметичним циклом.

Для підрахунку кількості повторень кожного арифметичного циклу вводять спеціальні змінні, які називають лічильниками циклів. Лічильник являє собою цілу змінну величину. На підготовчому етапі обчислень у схемі алгоритмів необхідно задати початкове значення для лічильника. Якщо лічильник працює на збільшення, тобто початкове значення його становить 0 чи 1, а після кожного виконання циклу лічильник збільшується на одиницю, у такому разі лічильник називається прямим. Якщо початкове значення відповідає максимальному числу повторень циклу, а після кожного виконання циклу воно зменшується на одиницю, то такий лічильник називається оберненим.

Ітераційний цикл

Не у всіх задачах заздалегідь відома кількість повторень циклу. Прикладами таких задач можна вважати обчислення квадратного кореня із 51 цілих позитивних чисел, обчислення тригонометричних функцій за допомогою рядів, інтегралів наближеними методами, уведення набору даних змінної довжини. Також ітераційними методами можна розв'язувати багато задач, пов'язаних з плануванням діяльності в економіці та проектувальних задач. В ітераційних циклах великого значення набуває визначення точності вишукуваного результату, що дозволяє встановити

момент виходу із циклу, або врахуванням якоїсь іншої умови.

Складний цикл

Як вже зазначалось, крім простих циклів, в обчислювальних алгоритмах застосовуються складні. Котрі, як правило , включають два й більше циклів, вкладених один в одного або розгалужених і таких, що не перетинаються.

6 МОДУЛІ, ОБ'ЄКТИ ТА КЛАСИ У ПРОГРАМУВАННІ

Для того, щоб користуватися розширеними можливостями Visual Basic, необхідно створювати власні процедури і модулі. Найважливіші «будівельні блоки» **Visual Basic** – форми та елементи. Третім видом таких «блоків» є програмні модулі.

Модулі являють собою текстові ASCII-файли з програмним кодом. У них зручно групувати взаємопов'язані процедури, які можуть використовуватися в програмі. Код проекту може складатися з безлічі програмних модулів. Класи представляють собою основні будівельні блоки об'єктно-орієнтованого програмування (ООП) – моделі, при якій програма описується в вигляді сукупності об'єктів. Клас не тільки дозволяє виділити частину функціональних засобів програми в окремий об'єкт, але і розширює можливості базових модулів – з'являється можливість захистити одні фрагменти програми, а інші – надати в розпорядження програми. Цей процес називається **інкапсуляція**.

Програмні об'єкти, що конструюються в ООП, імітують поведінку об'єктів реального світу. Добре спроектований клас являє собою цілком самостійний фрагмент програми. Це означає, що ви можете перенести клас з одного проекту в інший, і він буде нормально працювати без будь-яких виправлень. Оскільки модулі містяться в окремих файлах, їх можна включати відразу в кілька проектів. Таким чином, з'являється можливість повторно використовувати написаний код.

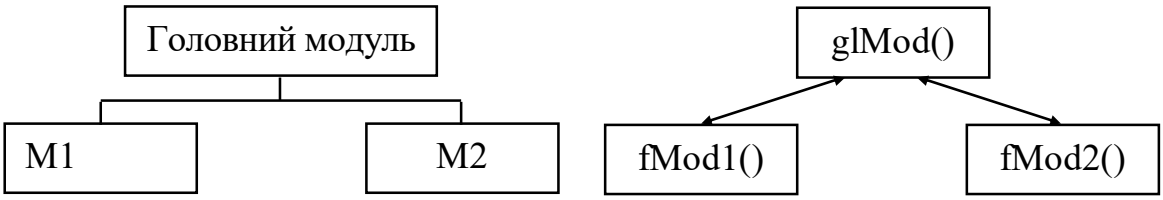
Наприклад, існує кілька модулів, що містять взаємопов'язані процедури. В одному модулі зберігаються функції для роботи з готовими вікнами. В іншому модулі об'єднані функції, що спрощують роботу з мультимедіа-пристроями. Групуючи однорідні функції в межах одного модуля, можна створити **програмну бібліотеку**. Якщо помістити модулі в один каталог або логічно зв'язаний набір підкаталогів, їх можна буде використовувати в інших проектах.

Використання програмних бібліотек економить час, оскільки вам не доведеться заново створювати вже написаний код. Можна сказати, що програмна бібліотека нагадує вантажівку для перевезення інструментів і припасів. В ній міститься багато ящиків, що містять різні інструменти; в одному ящику лежать молотки і викрутки, в іншому – приладдя для

фарбування. Різні типи даних знаходяться в одній бібліотеці, але при цьому не змішуються один з одним. Інша перевага програмних бібліотек полягає в тому, що після налагодження їх вміст можна сміливо використовувати в інших проектах. Ви отримуєте в своє розпорядження набір інструментів з довічною гарантією: не буде потреби витрачати час на створення фрагментів коду, яку вже були створені у кожному новому завданні.

Принципи модульного програмування

В основу **модульного програмування** покладено принцип функціональної декомпозиції. Складне завдання розбивається на підзадачі, а програма на модулі (рис. 6.1 а). На верхніх рівнях знаходяться керуючі модулі. Вони організовують виклики модулів нижніх рівнів. На нижніх рівнях знаходяться виконавчі модулі (М1 і М2). Кожен з них вирішує певну підзадачу. Відповідно до ієрархічної схеми програми розробляються основний і допоміжний алгоритми модулів. Основний алгоритм реалізує функції головного модуля, а допоміжні алгоритми – модулі нижніх рівнів.



а б

Рисунок 6.1 – Модульна структура алгоритму (а) і програми (б)

Між модулями організовується зв’язок з управління та інформації. Зв’язки з управління здійснюються по вертикалі зверху вниз. Модулі верхніх рівнів викликають модулі **нижніх** рівнів. Передача управління модулю, що викликається, здійснюється через його початок, а вихід відбувається через його закінчення. **Модуль, що викликається**, повертає управління модулю, який його викликав. Модулі нижніх рівнів не можуть викликати модулі верхніх рівнів. Модулі одного рівня не можуть викликати один одного.

Зв’язки за інформацією здійснюється або через **глобальні** змінні, які доступні у всіх модулях, або шляхом передачі параметрів. Оголошені в модулі змінні є **локальними** змінними. Вони не доступні іншим модулям.

Модулі VBA реалізуються **процедурами** і **функціями**. Винахід функцій – це великий крок в еволюції мов програмування. Функції є будівельними блоками, з яких будують складні програми. В основу появи функцій покладено принцип позначення складних комплексних сутностей простими іменами, який притаманний людському мисленню. В процесі пізнання люди систематизують свої знання, об'єднують їх в логічні конструкції і дають їм назви. З цих конструкцій збираються ще більш складні конструкції, яким теж дають назви. Аналогічно відбувається і з функціями.

Процедури, модулі і класи

Процедура – це сукупність операторів, що виконують певні дії. Процедури мають стандартне оформлення:

Public {[Private],[Static]} Sub *Name* (Список параметрів)
тіло процедури
End Sub

Де **Public** – глобальна процедура, доступна для усіх інших процедур у всіх модулях проекту;

Private – процедура модуля, доступна для усіх інших процедур в даному модулі;

Static – службове слово, яке говорить про те, що локальні змінні процедури зберігаються в проміжках часу між викликами цієї процедури;

Name – ім'я процедури, що задовольняє стандартним правилам написання імен у VBA. Ім'я процедури обробки події складається з імені об'єкту та імені події;

Список аргументів – список формальних параметрів (аргументів) процедури **Sub** (імен змінних, які повинні бути передані в процедуру під час звернення до неї).

Синтаксис елемента структури **Список аргументів** такий:

[Optional] [ByVal, ByRef] [ParamArray] Ім'я змінної **As** Тип

де **Optional** – ключове слово, яке вказує, що аргумент не є обов'язковим. Усі аргументи, описані як **Optional**, повинні належати до

типу **Variant**;

ByVal – вказує на те, що цей аргумент передається за значеннями;

ByRef – вказує на те, що цей аргумент передається за посиланням на його адресу в пам'яті. Опис **ByRef** використовується за замовчуванням;

ParamArray – використовується тільки як останній елемент у списку аргументів для вказівки на те, що кінцевим аргументом у списку параметрів процедури є масив значень, описаний як тип **Variant**, тобто це слово дозволяє задавати довільну кількість аргументів у процедурі.

Функції багато в чому схожі на процедури. Існує лише одна принципова відмінність – під час виклику вони повертають значення. Функція одержує один або декілька об'єктів даних, званих аргументами, і виконує з ними деякі дії. Їхній результат повертається функцією.

Функції мають такий вигляд:

Function Ім'я_Функції [(Список аргументів)] **As** тип
тіло функції
End Function

Модулі становлять текстові ASCII-файли з програмним кодом. У них зручно групувати взаємопов'язані процедури, які можуть використовуватися в програмі.

Код проєкту може складатися з безлічі програмних модулів. **Класи** представляють собою основні будівельні блоки об'єктно-орієнтованого програмування – моделі, в якій програма описується у вигляді сукупності об'єктів. Клас не тільки дозволяє виділити частину функціональних засобів програми в окремий об'єкт, але й розширює можливості базових модулів. Ви можете захистити одні із фрагментів програми, а інші надати в розпорядження програми. Цей процес називається *інкапсуляція*.

Оскільки модулі розміщуються в окремих файлах, їх можна включати відразу в декілька проєктів. Таким чином, з'являється можливість повторно використовувати написаний код.

Модуль класу – це спеціальний тип модулю, який використовується під час створення класів, що розробляються програмістом для вирішення своїх задач.

У VBA передбачена можливість створення призначених для користувача об'єктів, які є екземплярами класів. Усі об'єкти одного й того

самого класу використовують однакові методи у відповідь на однакові повідомлення. Класи конструюються в модулях класів, які створюються командою **Insert – Class Module** і записуються у вікні, що з'явилося. Натиснувши клавішу **F4**, можна вибрати ім'я класу.

ПРИКЛАД.

Процедура створення покажчика миші у формі пісочного годинника.

```
Public Sub ShowHourdlass()
  Screen.MousePointer = 11
End Sub
```

Повернення до звичайного покажчика миші

```
Public Sub ShowMousePointer()
  Screen.MousePointer = 0
End Sub
```

Створення модулів

Модуль – це файл з текстом програми, вставлений в документ. Програма на мові VBA складається з одного або декількох модулів. Модулі можуть бути наступних типів:

– модулі, пов'язані з об'єктами додатку, що реагують на події (модулі-обробники подій);

– програмні (стандартні) модулі класів, створювані програмістом;

– модулі макросів, створювані Macrorecorder;

– модулі-обробники подій завжди пов'язані з об'єктами, які реагують на події.

У модулі варто розміщувати тільки ті процедури і функції, які безпосередньо обробляють події. Програмні (стандартні) модулі – основний вид модулів, який створюється програмістом. Більшу частину всіх процедур і функцій слід розміщувати в стандартні модулі.

Практика доводить, що доцільніше створювати не один великий модуль, а кілька маленьких. В один модуль слід поміщати набір функцій, які спільно викликають один одного, об'єднаних спільною темою і загальної функціональної спрямованістю. Завжди виділяється головний модуль, у якому зосереджено опис глобальних змінних.

Модуль класу – спеціальний тип модуля, який використовується

при створенні класів, що розробляються програмістом для вирішення своїх завдань.

У VBA передбачена можливість створення користувацьких об'єктів, які є екземплярами класів. Всі об'єкти одного і того ж класу використовують однакові методи у відповідь на однакові повідомлення. Класи конструюються в модулях класів, які створюються командою **Insert / Class Module** і записуються у вікні. Натиснувши клавішу **F4**, можна вибрати ім'я класу.

Ім'я модуля класу є ім'ям класу об'єктів. У розділі опису модуля оголошуються змінні рівня модуля, які використовуються як «значення властивостей».

Клас ініціалізується за допомогою процедури *Class_Initialaize*. Імена властивостей, значеннями яких є числові дані, оголошуються за допомогою процедури **Property Let**. Процедурою **Property Set** оголошуються імена властивостей, значеннями яких є об'єкти. За допомогою процедури **Property Get** встановлюється можливість зчитування значень властивостей. Методи створюються за допомогою звичайних процедур і функцій.

Функції

Функції багато в чому схожі на процедури. Існує лише одна принципова відмінність – при виклику вони повертають значення. Функція отримує один або кілька об'єктів даних, званих аргументами, і виконує з ними деякі дії. Їх результат повертається функцією. Функції мають такий вигляд:

```
Function Ім'я_Функції [(Список аргументів)] As Тип
    тіло функції
End Function
```

Краще розглянути роботу функцій на конкретному прикладі. Наприклад необхідно створити функцію обчислення кубічного кореня. **ПРИКЛАД**.

```
Public Function CubeRoot (x As Double) As Double
  If x = 0 Then
```

```
    CubeRoot = 0
Exit Function End If
    CubeRoot = 10 ^ ((Log (Abs (x)) / Log (10)) / 3)
    If x < 0 Then
      CubeRoot = - CubeRoot
    End If
End Function
```

Виклик функції

```
A = 16
Z = CubeRoot (A)
```

Змінна Z отримує значення 2.

Розглянемо наведений вище код. Функція отримує аргумент **X** і обчислює його кубічний корінь. Тип значення, що повертається функцією (в прикладі **Double** – числовий тип, який може працювати з дуже великими і дуже малими, а також дробовими числами), вказується в заголовку функції після ключового слова **As**.

Текст функції закінчується командою **End Function**, а не **End Sub**. Зверніть увагу на три рядки, в яких присвоюється значення змінній **CubeRoot**. Значення функції повертається у вигляді змінної, ім'я якої збігається з ім'ям функції (в прикладі – **CubeRoot**). Якби функція називалася, припустимо, **TimesTwo** (), то і значення поверталося б в змінній **TimesTwo** (TimesTwo = x × 2).

Перед тим як приступати до обробки аргументу, бажано перевірити отримані дані. Спочатку необхідно перевірити, чи дорівнює аргумент 0 – як відомо, кубічний корінь з 0 дорівнює 0, тому можна відразу присвоїти змінній **CubeRoot** значення, що повертається і вийти з функції, не виконуючи подальших обчислень. Для виходу з функції застосовується команда **Exit Function**.

Команда **CubeRoot** = 10 ^ ((Log (Abs (x)) / Log (10)) / 3) повертає кубічний корінь аргументу **X**.

Тепер необхідно перевірити, чи не є аргумент функції негативним числом. Наведена вище формула завжди повертає позитивний результат, тому для негативного аргументу необхідно змінити знак значення, що повертається на протилежний. У Visual Basic це робиться командою:

CubeRoot = - CubeRoot

Then ... дії вашої програми ... **End If**

Об’єктно-орієнтована технологія розробки програм

В об’єктно-орієнтованому програмуванні базовими одиницями програм і даних є об’єкти.

Об’єкт – це сутність, яка чітко проявляє свою поведінку. Об’єкт складається з наступних трьох частин:

- ім’я об’єкта;
- стан (змінні стану);
- методи (операції).

Інтерфейс об’єкта з його оточенням визначено повністю його методами, так як до його стану немає іншого доступу ззовні, як через методи. Об’єкт зберігає свій стан від звернення до звернення. Зміна станів проводиться тільки через виклик методів цього об’єкта. Цим істотно обмежується можливість введення об’єкта в неприпустиме стан чи несанкціоноване руйнування об’єкта. Можливість управляти станами об’єкта через виклик методів в кінцевому підсумку буде визначати поведінку об’єкта.

Реалізація методів (тобто, операцій, виконуваних об’єктом), може бути задана різними способами. Однак це «внутрішня справа» об’єкта. Об’єкт може посилати повідомлення іншим об’єктам і приймати повідомлення від них.

Повідомлення є сукупністю даних певного типу, переданих об’єктом відправником об’єкту одержувачу, ім’я якого вказується в повідомленні. Одержувач реагує або ніяк не реагує (захист) на повідомлення виконанням деякої операції (методу), ім’я якого також може бути зазначено в повідомленні. За своїм змістом об’єкт є представником певної реальної суті (реального об’єкта, процесу або ситуації), яка:

- зберігається та обробляється;
- здатна впливати на інші об’єкти і обчислювальне середовище, посилаючи повідомлення і реагувати на прийняті повідомлення.

Сукупність об’єктів у системі ООП утворює середу, в якій виконуються обчислення шляхом обміну повідомленнями між об’єктами. Стан обчислювального середовища в ООП стає розділеним на стани

об’єктів, це принципово відрізняє об’єктно-орієнтовані обчислення від обчислень, заданих на процедурно-орієнтованих мовах. Процедури виконуються у загальній пам’яті, в той час як об’єкти виконують свої операції з урахуванням даних одного повідомлення і свого власного стану.

Об’єкти з однаковими властивостями, тобто з однаковими наборами змінних стану і методів, утворюють клас. Кожен клас задається своїм описом на мові ООП. Опис включає інформацію, необхідну для створення об’єктів даного класу і для їх існування. Це інформація про змінних стану і операціях об’єкта. Інакше кажучи, об’єктно-орієнтований підхід до програмування деякого завдання полягає в тому, щоб створити певний інструментарій, властивий задачі, що розв’язується, а потім вже програмувати в термінах цього завдання.

Атрибути об’єкта

Об’єкт має атрибути, що визначають його як самостійну сутність в світі програмування. У об’єктів, як і у будь-яких компонентів Visual Basic, є властивості, події і методи. Вони відрізняються від процедур тим, що підтримують два принципи: успадкування та інкапсуляцію.

Властивості об’єкта

Властивості описують об’єкт. Кожен об’єкт, який використовується або створюється в програмі, має деяке ім’ям класу, за яким його можна відрізнити від інших об’єктів.

Методи об’єкта

Методами називаються дії, що виконуються об’єктом. Наприклад, об’єкт-вікно може відображати або приховувати себе на екрані і навіть змінювати свої розміри.

Наслідування

Наслідуванням називається здатність об’єкта зберігати атрибути класу-батька. Наприклад, створений об’єкт форми успадковує властивості і методи свого класу-батька. Він володіє такими властивостями, як **Name**, **WindowState** і **BorderStyle**, а також методами **Load**, **UnIoad** і **Hide**. Щоб створити екземпляр об’єкта в Visual Basic, слід вказати на його приналежність до певного класу і потім скористатися ключовим словом **New**. Новий об’єкт успадковує властивості і методи батька.

Інкапсуляція

Інкапсуляцією називається механізм, завдяки якому дані і методи об'єкта ховаються від зовнішнього світу. Програміст ізолюється від складнощів внутрішньої реалізації об'єкта. Інкапсуляція становить одну з найсильніших сторін ООП.

Проєктуючи об'єкт, намагайтеся обмежуватися мінімумом відкритих властивостей. Закривайте все, з чим не повинні працювати зовнішні програми. Це запобіжить випадковому внесенню помилок іншими програмістами.

Відношення класів. Типи класів

У Arclnfo розробники додатків можуть використовувати нову COM-технологію розробки ArcObjects. Вона надає можливість використання добре документованої моделі даних з компонентами, що впроваджуються. ArcObjects дозволяє вирішувати широке коло завдань: від налаштування і розширення Arclnfo до побудови нових ГІС-додатків. Оскільки Visual Basic for Applications (VBA) – середовище розробки, що поставляється з Arclnfo, то він використовується найбільш часто.

Технологія COM

Технологія COM (Component Object Model) – компонентна модель об'єктів, надає можливість одній програмі (клієнту) працювати з об'єктом іншої програми (сервера). COM – це модель об'єкта, яка передбачає повну сумісність у взаємодії між компонентами, написаними різними компаніями і на різних мовах. З точки зору COM додаток містить кілька об'єктів (в окремому випадку може бути один об'єкт). Кожен об'єкт має один або кілька інтерфейсів. В інтерфейсі описані методи об'єкта, до яких можуть отримати доступ зовнішні програми. Якщо інтерфейсів кілька, кожен з них експонує деяку підмножину методів, що виконують однорідні функції. Об'єкт є частиною сервера COM. Сервером може бути виконуваний файл або бібліотека DLL. При установці сервера в Windows до реєстру заноситься інформація про всі його об'єкти. Ця інформація включає в себе ідентифікатор класу CLSID (Class Identifier), який однозначно визначає клас об'єкта. Заноситься інформація про тип сервера:

– внутрішній (in-process – всередині процесу) – DLL, підключається до клієнта,

– локальний (local) – працює окремим процесом на комп'ютері клієнта,

– віддалений (remote) – працює на віддаленому комп'ютері.

Для внутрішніх і локальних серверів до реєстру заноситься повне ім'я файлу, а для віддалених – повна мережева адреса. Таким чином, в системі зберігається вся інформація про сервер COM, необхідна для виклику його в потрібний момент. Внутрішнім сервером є DLL (динамічна бібліотека, що приєднується), яка експортує автоматні об'єкти. Оскільки автоматні об'єкти поставляються з DLL, а не з інших додатків, вони є частиною програми клієнта. Це позбавляє від великих накладних витрат, які супроводжують кожний виклик автоматного сервера.

Локальний або віддалений сервер – це автономний виконуваний файл, що експортує автоматні об'єкти. Прикладом цього міг би бути ArcMap. ArcMap має об'єкти, які він експонує в якості автоматних.

Зовнішні додатки, які звертаються до об'єкта COM, є клієнтами COM. Клієнт отримує покажчик на цікавий для його інтерфейс об'єкта і через цей покажчик може викликати методи об'єкту. Специфікація COM забороняє змінювати один раз оголошений інтерфейс. Це забезпечує нормальну роботу клієнта при будь-яких модифікаціях сервера. Таким чином, клієнту достатньо знати інтерфейси об'єкта і надані ними методи. Про інше подбає система. У потрібний момент вона запустить сервер COM, якщо він ще не був запущений, сервер створить об'єкт, об'єкт завантажить всі необхідні йому дані і клієнтові повернуться покажчики на об'єкт і його інтерфейси, з якими він може працювати. Система подбає також про те, щоб забезпечити роботу об'єкта відразу з декількома клієнтами. Для цього вона веде облік числа посилань на об'єкт. При видачі клієнту покажчика на інтерфейс число посилань збільшується на 1. А при закінченні роботи клієнта з об'єктом число посилань на 1 зменшується. Якщо число посилань стало дорівнювати нулю, система знищує об'єкт, з яким в даний час не працює жоден клієнт. Все добре, але звідки розробник клієнтського додатка може отримати інформацію про об'єкти COM, їх інтерфейси, властивості, методи, параметрах методів? Вся ця інформація міститься в бібліотеці типів, яка створюється розробником об'єкта COM і поширюється разом з об'єктом. Бібліотека створюється за допомогою мови опису інтерфейсу IDL (Interface Definition Language).

Інтерфейси

Тепер розглянемо докладніше інтерфейси. Кожен інтерфейс має ім'я, що починається з символу «I», і GUID – глобальний унікальний ідентифікатор (Globally Unique Identifier). Подібні GUID створюються і використовуються не тільки для інтерфейсів. Для інтерфейсів GUID називається IID.

Кожен об'єкт COM має інтерфейс IUnknown. Цей інтерфейс має всього три методи:

- QueryInterface – отримання покажчика на інтерфейс,
- AddRef і Release – збільшення і зменшення на 1 числа посилань на об'єкт.

Метод **QueryInterface** повертає покажчик на інтерфейс з заданим IID.

Метод **Release** повинен викликатися після закінчення роботи з інтерфейсом, щоб повідомити об'єкт, що даний клієнт в ньому більше не працює.

Ці два методи використовуються завжди при роботі з об'єктом COM. Метод **AddRef** використовується тільки в тих випадках, коли один клієнт передав іншому посилання на інтерфейс. Оскільки при цьому метод AddRef, який автоматично збільшує число посилань, не викликається, клієнт, якому передане посилання, повинен викликати AddRef, щоб доповісти об'єкту, що він теж з ним працює.

Всі інші інтерфейси об'єктів є спадкоємцями IUnknown і успадковують ті ж три методи, додаючи, свої власні. Це можна застосувати до будь-яких інтерфейсів. Серед цих інтерфейсів-спадкоємців необхідно відзначити **IDispatch**. Це інтерфейс, використовуваний клієнтами для доступу до методів і властивостей об'єкта.

Бібліотека COM

Технологія COM реалізується спеціальними бібліотеками. Вони містять стандартні інтерфейси і функції, які забезпечують створення об'єктів COM і управління ними.

Імена всіх функцій бібліотеки починаються з «**Co**». Створення об'єкта здійснюється викликом функції **CoCreateInstance** цієї бібліотеки і передачею в неї **CLSID** потрібного класу, **IID** інтерфейсу і необхідного типу сервера. Бібліотека COM звертається до системного реєстру, в якому

зберігається інформація про сервер, запитує його і повертає необхідний покажчик. Так як технологія COM не залежить від мови, в ній використовуються типи, відмінні від інших мов. Перш за все, це відноситься до рядків, які в різних мовах описуються по-різному.

У COM використовується свій строковий тип – **BSTR** (Basic String). Він описує рядок, на початку якого вказана його довжина. Оскільки довжина рядка відома, завершального нульового символу немає.

Реалізація технології COM у ArcObjects

ArcObjects – це платформа розробки для таких модулів ArcGIS, як ArcMap, ArcCatalog і ArcScene. Програмні компоненти ArcObjects охоплюють повний діапазон функціональних можливостей, доступних в Arclnfo і Arc View для розробників програм.

ArcObjects – це інфраструктура, яка дозволяє створювати специфічні для даної предметної області компоненти з інших компонент. Компоненти ArcObjects взаємодіють, щоб обслужити кожну з функцій управління даними і функції уявлення карти, загальні для більшості ГІС додатків. ArcObjects включає більше 1200 об’єктів, які можуть бути використані для налаштування, розширення і побудови ГІС додатків.

Структура ArcObjects

ArcObjects побудований з використанням технології **Моделі компонентного об’єкту фірми Microsoft (COM – Component Object Model**).

Модель компонентного об’єкта (МКО) – це методологія розробки програм. МКО визначає протокол, який з’єднує один програмний компонент або модуль з іншим. Використовуючи цей протокол, можна будувати програмні компоненти багаторазового використання, які можуть бути динамічно замінні в розподіленій системі. МКО також визначає модель програмування, відому як інтерфейсно-засноване програмування. Компоненти полегшують багаторазове використання програмних компонентів, тому що вони – окремі стандартні блоки, які можна легко зібрати у великі системи.

7 ВИКОРИСТАННЯ КЛАСІВ ТА ОБ'ЄКТІВ У ПРОГРАМУВАННІ

Типи і класи

Клас є узагальненням поняття типу даних і задає властивості і поведінку об'єктів класі – екземплярів класу. Кожен об'єкт належить деякому класу. Відношення між об'єктом і його класом таке ж, як між змінною та її типом. **Клас** – це об'єднання даних і процедур, що їх обробляють та функцій.

Дані називаються також змінними класу, а процедури і функції – методами класу. Змінні визначають властивості об'єкта, а сукупність їх значень – стан об'єкта.

Окрім властивостей і методів із класом поєднується ще одне поняття – події. Кожен клас має певний набір подій, які можуть виникати при роботі з об'єктами класу, найчастіше при певних діях користувача, іноді, як результат дії системи. При виникненні події, пов'язаної з тим чи іншим об'єктом, система посилає повідомлення об'єкту, яке може бути оброблено методом – обробником події, спеціально створеним при конструюванні об'єкта. Події забезпечують більшу гнучкість при роботі з об'єктами. Методи класу виконуються однаково для всіх об'єктів класу, а на події кожен об'єкт реагує індивідуально, оскільки має власний обробник події.

Програмування в класах є основним способом роботи сучасного програмування.

ПРИКЛАД.

Таким чином побудована операційна система Windows – вікно є її основним об'єктом. Office повністю побудований на класах і роботі з об'єктами цих класів – тут все, починаючи від програми і закінчуючи окремим символом, розглядається як об'єкт деякого класу.

Професійний прикладний програміст, що працює в деякій проблемній області та вирішує різноманітні завдання цієї області, зазвичай починає зі створення класів, що описують специфіку даної проблемної області. Потім вже рішення тих чи інших спеціальних завдань він описує в термінах роботи з об'єктами даної проблемної області. VBA дозволяє програмісту створювати власні класи. Синтаксично клас представляє окремий модуль спеціального виду – модуль класу.

Нове в класах у додатку MS Office

Тема класів знаходиться на магістральному шляху розвитку офісного середовища. У порівнянні з попередніми версіями, у останніх зроблені три важливих нововведення:

– класи можуть тепер мати власні події. Раніше, об'єкти класів мали тільки обмежений набір стандартних подій. Можливість створення власних подій серйозне досягнення у формуванні повноцінного класу;

– можливість реалізації інтерфейсів, визначених іншими класами;

– введено нову властивість **Instancing**, що регулює способи взаємодії з класами в системі документів інших проєктів.

Створення класу «Особистість»

Припустимо, був створений призначений для користувача тип **Person** (змінні ***Петров*** і ***Козлов***). Але йому не вистачає власних операцій: якби змінні Петров і Козлов мали тип, наприклад, **Integer**, можна було б їх порівнювати, складати і множити, присвоювати значення і багато іншого. Зрозуміло, що перемноження даних типу **Person** навряд чи розумно, але зате для нього можна визначити свої операції. Ось які операції можна визначити для цього типу:

– **InitPerson** – ініціалізація полів запису;

– **PrintPerson** – друк полів запису;

– **CopyPerson (Source As Person**) – копіювання джерела (запису **Source**) – аналог оператора присвоювання;

– **WhoIs** – більш специфічна операція, з певною вірогідністю визначає стать, аналізуючи ім'я і прізвище.

Завжди при визначенні призначеного для користувача типу, так чи інакше, але слід визначити операції над даними цього типу. Природно зібрати визначення типу і операції над ним в одному місці. Таке об'єднання представляє вже майже повне, з точністю до подій визначення типу. Для типів, визначених подібним чином, введено новий термін – **клас**.

Синтаксично класи в VBA оформляються у вигляді модуля класу. Тому починати створення класу в редакторі Visual Basic потрібно з вибору в меню **Insert** пункту **Class Module**. Цей модуль має таку ж структуру, як і стандартний модуль. Модуль складається з двох розділів – оголошень і методів. У першому з них природним чином описуються властивості

класу, а в другому – його методи. І тут діють специфікатори області дії **Public** і **Private**:

- **Public** – властивості;
- **Private** – методи,що складають інтерфейс класу.

Об'єкти та змінні

Спробуємо узагальнити відомості про об'єкти і змінних. Почнемо з того, що у них спільного і що їх відрізняє. Якщо є описи:

Dim X **As** T**,** Y **As** T1

то без контексту зрозуміти не можна, що є *X* і *Y* – «звичайні» змінні або об'єкти.

Наприклад, якщо *T* – тип, заданий користувачем, а *T1* – визначений ним клас, то *X* – це змінна, а **Y** – об'єкт.

Клас і **тип** – поняття, якщо не еквівалентні, то близькі за змістом. Також близькі поняття об'єкта і змінної. **Клас – це спеціальна форма визначення типу**. Якщо є тип *T* і клас *T1*, то можна оголосити довільне число екземплярів типу *T* і класу *T1*.

Екземпляри типу T називаються **змінними**, класу T1 – **об'єктами**. Клас задає властивості, методи і події своїх об'єктів. Тип завжди задає властивості, неявно – методи, але ніколи – події.

В оголошенні: **Dim** Y **As** T1, де T1 – клас, об'єкт Y можна назвати змінною, стверджуючи, що вона має тип **Object**. Змінні типу **Object** розглядаються як посилання, що задають адресу пам'яті, де зберігається об'єкт. У них фіксована довжина 4 байти. При оголошенні такої змінної пам'ять для самого об'єкта може і не відводитися на відміну від звичайних змінних, тому не визначено і значення посилання. Задати посилання – зв'язати змінну з об'єктом можна двома шляхами:

- створити новий об'єкт, виділивши йому пам'ять;
- послатися на вже наявний об'єкт.

Є два способи зв'язування: **раннє** і **пізнє**. При пізньому зв'язуванні змінна оголошується так:

Dim V **As** Object

Це оголошення свідчить про те, що змінна є об'єктом (посиланням), але нічого про клас цього об'єкта – він може бути довільним, і з'ясується це тільки динамічно при виконанні програми, коли змінна *V* буде

зв’язуватися із щойно створеним або існуючим об’єктом того чи іншого класу. Тому таке зв’язування називається пізнім, або динамічним. Природно, що неоголошені змінні або ті, яким тип при оголошенні не заданий (за замовчуванням тип **Variant**), допускають тільки пізнє зв’язування. При ранньому зв’язуванні в момент оголошення вказується клас об’єкта, наприклад:

Dim Петров **As** Особистість, Козлов **As** Особистість

Це дозволяє ще на етапі трансляції перевірити, чи припустимі ті чи інші операції щодо об’єктів *Козлов* і *Петров*. За винятком вельми сумнівної переваги пізнього зв’язування – можливістю зв’язатися з об’єктом будь-якого типу, раннє зв’язування краще в усіх відношеннях.

Яким же чином створюються нові об’єкти, і як відбувається зв’язування з об’єктами, вже існуючими? Для зручності розділимо всі об’єкти на три групи:

– об’єкти, чий клас визначений користувачем в одному з модулів класу; наприклад, об’єкти класу *Особистість*;

– об’єкти додатку (Excel, Word, PowerPoint), якому належить проект і які доступні за замовчуванням;

– **ActiveX** і **Com AddIns** – об’єкти інших додатків Office при їх підключенні до вихідного додатка.

ПРИКЛАД. В Word можна підключити об’єкти Excel, а документи Word включити в робочі листи Excel.

Об’єкти, клас яких визначено користувачем

Зрозуміло, що при роботі з такими об’єктами хоча б частину з них необхідно спочатку створити, і робиться це при їх оголошенні:

{Dim | Private | Public | Static} <ім’я змінної> As New <ім’я типу>

Специфікатор **New** вказує, що в момент оголошення потрібно створити новий об’єкт, тобто виділити йому пам’ять. В цей же момент посилання на об’єкт отримує конкретне значення. Виділення пам’яті ще не означає ініціалізації значень властивостей об’єкта. Ініціалізацію можна задати при визначенні події **Initialize** або в спеціально побудованому методі **Init**, який слід запускати на початку роботи з об’єктом.

Ось приклад створення об’єкта з подальшою його ініціалізацією:

Public Sub MyCreateObject()

‘Не можна створювати об’єкти власного класу, використовуючи CreateObject

Dim Nemo As Особистість

‘Set Nemo = **CreateObject** ("Особистість ")

‘Можна їх створювати, використовуючи **New**

Set Nemo = **New** Особистість

Nemo.InitPerson FN:="Prince", LN:="Dakar", DoB:=#1/23/1838#

Nemo.PrintPerson

End Sub

Тільки метод **New** дозволяє створити новий об’єкт для класів, визначених програмістом. Для цієї мети можна, наприклад, використовувати метод **CreateObject**. Але, зауважте, специфікатор **New** не є обов’язковим, його можна опускати при оголошенні таких змінних класу, які в подальшому отримають значення, завдяки посиланням на інші, раніше створені об’єкти. Задати посилання на існуючий об’єкт не можна звичайним оператором присвоєння – для цього в VBA є спеціальний оператор **Set**:

Set <им’я змінної – об’єкта> = {< об’єктний вираз>| Nothing }

Set – спеціально виділений, окремий випадок оператора присвоєння, в його лівій частині стоїть ім’я змінної, визначеної як об’єкт при оголошенні, а в правій – деякий вираз, значення якого – посилання на об’єкт. В результаті змінна лівої частини отримує значення виразу у правій частині. Ось приклад деякого оголошення об’єкта **Q**.

Dim Q **As** Особистість

Set Q = Nemo

Q.PrintPerson

У результаті буде надруковано такий текст:

Prince Dakar народився 23.01.1838

Об'єкти «батьківського» додатка

Кожен VBA-проект занурений в «батьківський» додаток Office (ArcMap), об'єкти якого доступні в проекті. Для створення таких об'єктів не використовується ні специфікатор **New**, ні метод **CreateObject**. Нові об'єкти, якщо і створюються, то спеціальними методами свого класу. Наприклад, метод **Add** дозволяє додавати елементи в колекції. При роботі з об'єктами цієї групи оголошуються змінні відповідного класу, потім вони зв'язуються з уже існуючими об'єктами. Зв'язування виконує оператор **Set**.

ActiveX-об'єкти

Технологія **ActiveX** забезпечує взаємодію додатків – автоматизацію (Automation), при якій один додаток управляє роботою іншого. На цій технології побудовано взаємодію додатків Office. Щоб почати роботу з ActiveX-об'єктом, потрібно оголосити змінну з описом, створити сам об'єкт і зв'язати змінну з об'єктом. Один із способів виконання цієї роботи – використовувати специфікатор **New** в операторі оголошення змінних, який в цій ситуації має вигляд:

Dim <ім'я змінної> **As New** < ім'я додаткуя. ім'я класу>

Порявняно зі звичайною конструкцією оператора оголошення, тут складніше задається тип (клас) створюваного об'єкта.

Перша частина – **імя_додатку** вказує додаток, що створює об'єкт, і водночас це ім'я бібліотеки типів цього додатка – **TypeLib**. Додаток є сервером, який викликається для забезпечення роботи з об'єктом.

Друга частина – **ім'я_класу** задає той конкретний клас, визначений у сервері, екземпляр ActiveX-об'єкт якого буде створений. Бібліотека типів додатка, що підключається повинна бути видимою, для чого треба включити посилання на неї в меню **References**.

ПРИКЛАД. Підключення Excel до додатка Word:

```
Public Sub WorkWithActiveX()
'Створення об'єкта Excel.Application при роботі з документом Word
```

```
    Dim xlApp As Excel.Application
    'Set xlApp = CreateObject("Excel.Application.9")
    Set xlApp = New Excel.Application
    xlApp.Application.Visible = True
    'Додати нову книгу
    xlApp.Workbooks.Add
    'Тепер можна працювати з клітинками цієї книги
    xlApp.Workbooks(1).Activate
    xlApp.Range("A1:A2") = 2
    xlApp.Range("B1") = "=A1+A2"
    xlApp.Range("B2") = "=A1*A2"
    'Закриваємо відкриту книгу без збереження змін
    xlApp.Workbooks(1).Close SaveChanges:=False
    'Закриваємо додаток
    xlApp.Quit
End Sub
```

Створити Excel-додатки в цьому випадку можна двома способами – шляхом застосування конструкції **New** або функції **CreateObject**. Однак, використання функції **New** для створення ActiveX-об'єктів не завжди можливо. Не всі програми допускають такий спосіб підключення, і не всі об'єкти доступні – наприклад, недоступні внутрішні ActiveX-об'єкти, які одночасно можуть зберігатися в одному і тому ж файлі. Більш універсальний спосіб створення ActiveX-об'єктів – виклик спеціальних функцій **CreateObject** і **GetObject**. Як правило, виклик цих функцій поміщається в праву частину оператора **Set**. В результаті його виконання об'єктний вираз повертає посилання на створений об'єкт, що й стає значенням змінної.

Приховування властивостей

Виникає питання, чому властивості оголошені за допомогою

оператора **Private**, а не **Public**? Зверніть увагу, таким же чином було описано властивості і в класі «*Особистість*». Така загальноприйнята практика об'єктно-орієнтованого програмування. Якби властивості мали атрибут **Public** – вони були б відкритими, тоді при роботі з об'єктом, можна було б мати до них прямий доступ, як при читанні, так і запису. Але така свобода, зазвичай неприпустима.

Ось можливі стратегії при роботі з властивостями:

- читання, запис (Read, Write);
- читання, запис при першому зверненні (Read, Write-once);
- тільки читання (Read-only);
- тільки запис (Write-only);
- ні читання, ні запис (Not Read, Not Write).

Конструктори та деструктори

Наступним кроком у створенні класу є розробка його конструкторів і деструкторів. Нагадаємо, що новий об'єкт – екземпляр класу створюється конструкцією **New**, наприклад:

Dim ***MyRationalNumber*** **As New** ***Rational***

У цей момент:

- створюється типізований покажчик **MyRationalNumber**;
- створюється новий об'єкт – екземпляр класу **Rational**;
- при створенні об'єкта викликається конструктор за замовчуванням, ініціалізуючий цей об'єкт, який визначає значення його властивостей;
- покажчик зв'язується з об'єктом.

У VBA, на відміну від багатьох інших мов програмування, є розумна стратегія початкової ініціалізації змінних. Тому тут є тільки **конструктор за замовчуванням** – конструктор без параметрів, та й той часто не визначається, покладаючись на стандартну ініціалізацію. Звичайно ж, форматувати об'єкт «справжніми» значеннями все одно доведеться в якийсь момент. Тому завжди для класу створюються свої конструктори, які синтаксично є методами– їх може бути кілька. У класі «*Особистість*» таким конструктором з параметрами є метод **InitPerson**. Зверніть увагу, можна було б визначити ще один конструктор, який на відміну від

першого, проводив би повну ініціалізацію всіх властивостей, включаючи не тільки **Ім'я**, але й **По батькові** класу «*Особистість*».

Деструкція викликається автоматично при знищенні об'єкта. У VBA немає динамічного знищення об'єкта в момент, визначений програмістом, об'єкти знищуються також як і змінні, при виході з області їх дії. Тому деструктор, як правило, не пишеться.

Стандартні події *Initialize* і *Terminate*

Роль конструктора за замовчуванням в класах VBA грає обробник події **Initialize**. Це загальна для багатьох об'єктів подія, зустрічається, коли об'єкт завантажується, для об'єктів-екземплярів класів вона виникає при створенні об'єкта. У обробника цієї події немає параметрів, тому він грає роль конструктора за замовчуванням, що не має параметрів.

Роль деструктора грає обробник події **Terminate**. Він викликається, коли всі раніше встановлені посилання на екземпляр об'єкта отримують значення **Nothing** або всі покажчики перестають існувати, вийшовши з області свого визначення. Треба зауважити, що при ненормальному завершенні програми ця подія не виникає. Оброблювач цієї події (деструктор) пишеться значно рідше, оскільки в момент його виклику об'єкт і так коректно буде знищений.

Програмісти Microsoft створили всі необхідні класи для роботи з об'єктами VBA, такі як форми і елементи управління. А програмісти ESRI, у свою чергу написали класи для ГІС-об'єктів, таких як карти і шари. Проте для вирішення специфічних завдань потрібно створювати свої класи.

Уявіть, що ви працюєте у муніципальній міській компанії, яка створює додаток для податкового управління з інвентаризації. Він буде корисний для моделі міських кварталів в ArcMap. Але VBA та ArcMap не мають об'єкта **Квартал** (**Parcel**). Можна створити свій клас кварталів з класу **Parcel Feature** в базі геоданих і запитувати його властивості. Але не існує програмного об'єкта, який зветься квартал. Тому не можна написати код для запиту значення кварталу (його номер або ранг):

MsgBox *myParcel*.Value

Не можна також написати код для отримання кодування зони кварталу:

MsgBox *myParcel*.Zoning

Тому при частому використанні доцільно створити власний клас. Перед тим, як програмувати потрібно спланувати, які об’єкти та методи будуть входити до цього класу. Насамперед треба викласти свої думки на папері. Але рано чи пізно доведеться звернутися до мови моделювання UML.

UML – це технологія створення схематичних діаграм. Програмісти використовують діаграми, щоб малювати класи, властивості і методи стандартними символами. Наприклад, класи (подібно до кварталу **Parcel**) зображені як прямокутники, властивості (подібно до **Value** і **Zoning**), поряд із назвою мають спеціальні мітки і методи (подібно **CalculateTax**) поряд із назвами стрілок. Ці UML діаграми мають назву – діаграмами моделі об’єкта (рис. 7.1).

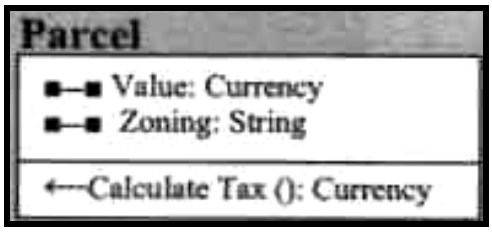


Рисунок 7.1 – UML діаграма моделі об’єкта

На діаграмі типи даних відображені зліва від властивостей і методів. Наприклад, властивість **Zoning** має рядковий тип.

У мові VBA, щоб створити новий клас, формують новий модуль класу. Тобто скориставшись командою: **Insert / Class Module**, модуль можна зберегти в будь-якому з трьох проектів: поточному документі карти, новому або основному шаблоні. При цьому будуть розрізнятися рівні доступу до модулю, як і під час роботи з макросами та процедурами.

Щоб створити властивість для класу, необхідно оголосити перемінну з потрібним типом даних. Нехай клас **Parcel** буде мати дві властивості, які можна створити, оголошуючи змінні в модулі класу (рис. 7.2).

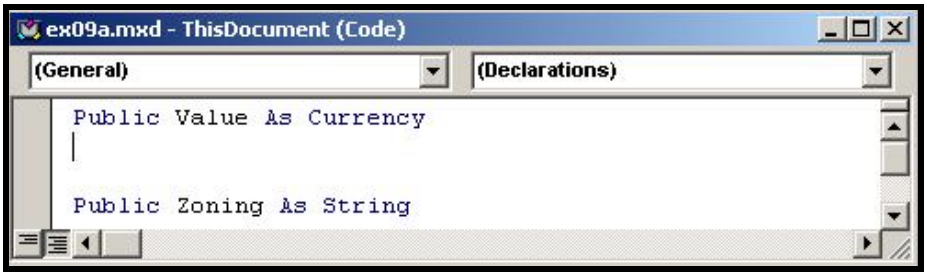


Рисунок 7.2 – Оголошення перемінних в модулі класу

Ці змінні оголошуються поза процедурою з ключовим словом **Public**, замість ключового слова **Dim**. Змінні, оголошені з оператором **Dim**, можуть використовуватися тільки в процедурі, в якій вони оголошені. Зверніть увагу, що змінні **Value** та **Zoning** не належать певній процедурі.

Оголошення змінних поза процедурою за допомогою оператора **Public** дозволяє використовувати їх в будь-якій процедурі та в будь-якому модулі коду (поки процедура знаходиться в поточному документі карти, оскільки клас зберігається в поточному документі карти).

Щоб створити метод для класу, необхідно закодувати підпрограму або функцію залежно від того, чи повертає цей метод значення. Метод **CalculateTax** повертає розмір податку, тому даний метод був закодований як функція. На діаграмі класу відзначено, що даний метод має тип **Currency** (грошовий). Очевидно, що він повертає значення вказаного типу. Аргументів він не має, тому дужки залишаються порожніми. Таким чином, оголошення цього методу буде виглядати наступним чином (рис. 7.3).

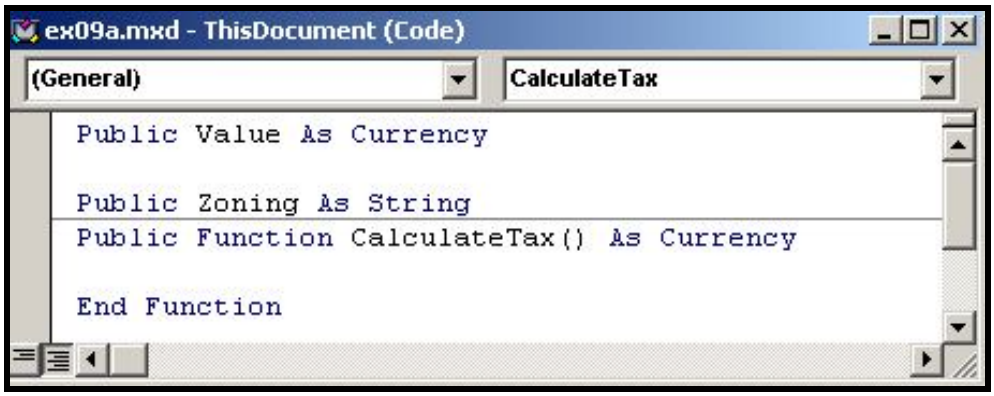


Рисунок 7.3 – Оголошення методу

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Основні

1. Поморцева Е. Е. Программирование геоинформационных задач: Учеб. пособие / Е. Е. Поморцева. – Харьков : ХНУГХ им. А. Н. Бекетова, 2017. – 121 с.
2. Гурвиц Г. А. Microsoft Access 2010. Разработка приложений на реальном примере / Г. А. Гурвиц. – Изд. BHV, 2010. – 496 с.
3. ArcTooIBox Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2020 – 105 с.
4. ArcMap Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2020. – 220 с.
5. ArcCatalog Руководство пользователя. – Киев : ЗАО «ЕКОММ», 2020. –180 с.

Додаткові

6. Попов И. В. Эффективное использование ArcObjects. Методическое руководство / И. В. Попов, М. А. Чикинев. – Новосибирск : СО РАН, 2003. – 160 с.

Ресурси мережі Internet

7. Сайт додатків Office корпорації Microsoft – Режим доступу: http://office.microsoft.com/ru-ru/access-help/.
8. Сайт корпорації «ArcGis Recource Center» по роботі з програмним продуктом. – Режим доступу: http://help.arcgis.com/
9. Сайт «Высокие технологии». – Режим доступа: http://www.citymap.odessa.ua.
10. Цифровий репозиторій ХНУМГ ім. О. М. Бекетова. – Режим доступу : http://eprints.kname.edu.ua.

Навчальне видання

ПОМОРЦЕВА Олена Євгенівна

ПРОГРАМУВАННЯ ГЕОІНФОРМАЦІЙНИХ ЗАДАЧ

КОНСПЕКТ ЛЕКЦІЙ

(для студентів першого (бакалаврського) рівня вищої освіти спеціальності 193 – Геодезія та землеустрій)

Відповідальний за випуск *О. Є. Поморцева*

За авторською редакцією

Комп'ютерний набір *О. Є. Поморцева*

Комп'ютерне верстання *О. Є. Поморцева*

Підп. до друку 24.09.2021. Формат 60 х 84/16.
Електронне видання. Ум. друк. арк. 5,0.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Маршала Бажанова, 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб'єкта видавничої справи:
ДК № 5328 від 11.04.2017.