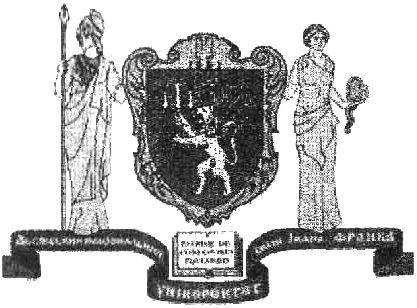
Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

І. М. Дудзяний

Програмування мовою Visual Basic/VBA

Навчальний посібник



Львів
Видавничий центр ЛНУ імені Івана Франка
2004

Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

І. М. Дудзяний

Програмування мовою Visual Basic/VBA

Рекомендовано
Міністерством освіти і науки України
як навчальний посібник
для студентів вищих навчальних закладів

Львів
Видавничий центр ЛНУ імені Івана Франка
2004

УДК 004.438 Visual Basic (075.8)
Д–81
ББК 3973.2-018я73 Visual Basic

Рецензенти:

Ю.М. Рашкевич, д-р техн. наук, проф.
(Національний університет "Львівська політехніка")
І.І. Лазурчак, канд. фіз.–мат. наук, доц.
(Дрогобицький державний педагогічний університет)

Рекомендовано до друку Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів (*лист Першого заступника Міністра – керівника апарату за №14/18.2-1781 від 26.07.2004 р.*)

Дудзяний І.М.

Д–81 Програмування мовою Visual Basic/VBA. Навчальний посібник. - Львів: Видавничий центр ЛНУ імені Івана Франка, 2004. - 240 с.

ISBN

У посібнику систематизовано основні прийоми програмування мовою Visual Basic: опис типів даних, оголошення змінних і констант, організацію галужень і циклів, опис і використання структурованих типів даних, підпрограм і модулів. Розглянуто реалізацію у Visual Basic об'єктно-орієнтованої парадигми програмування (класи, об'єкти, конструктори і деструктори, методи, властивості та події). На достатньо детальному для навчальних цілей рівні описано методи та властивості основних елементів керування, технологію розроблення проектів у середовищі Visual Basic. Особливу увагу приділено програмуванню офісних застосувань мовою Visual Basic for Application (VBA).

Для студентів вищих навчальних закладів, які вивчають сучасні інформаційні технології.

ББК 3973.2-018я73 Visual Basic

ISBN

ПЕРЕДМОВА

Першу версію мови Visual Basic створено компанією Microsoft навесні 1991 року. Вона зразу ж стала дуже популярною серед розробників комерційних застосувань, оскільки давала змогу швидко створювати робочі прототипи Windows-застосувань. Хоча Visual Basic 1.0 була абсолютно новим інструментом створення Windows-застосувань, у ній не було вбудовано підтримки баз даних. Компанія Microsoft розуміла, що ця обставина накладає певні обмеження на роботу зі серверами, а тому версія Visual Basic 3.0 уже забезпечувала доступ до даних за допомогою технології DAO.

Компанія Microsoft постійно працює над удосконаленням мови Visual Basic та її інтегрованим середовищем. Значної уваги розвитку Visual Basic приділяє сам Білл Гейтс. Після виходу версії 6.0, у якій підтримується доступ до даних за допомогою технології ADO, мова Visual Basic стала потужною і надійною мовою програмування корпоративних застосувань. Порівняно незначна кількість базових понять, простий синтаксис досить швидко зробили її популярною в усьому світі.

Інтегроване середовище Visual Basic поєднує візуальні засоби проектування програмних продуктів, підтримку сучасних інформаційних технологій, роботу з базами даних тощо. Побудований на архітектурі об'єктно-орієнтованих елементів керування, Visual Basic дає змогу багатократно використовувати уже створений код, зменшуючи тим самим час і зусилля, які витрачаються на розроблення програмних продуктів.

Зауважимо, що Visual Basic використовують як вбудовану мову багатьох популярних пакетів, зокрема MS Office, MS Visio тощо (мова Visual Basic for Application, коротко VBA).

У посібнику систематизовано головні прийоми програмування мовою Visual Basic: опис типів даних, оголошення змінних і констант, організацію галужень і циклів, опис і використання структурованих типів даних, підпрограм і модулів. Зазначимо, що систематичне викладення головних прийомів програмування мовою Visual Basic у книзі базується на кодуванні типових алгоритмів роботи з масивами, рядками, записами, файлами і динамічними структурами даних.

Розглянуто реалізацію у Visual Basic об'єктно-орієнтованої парадигми програмування. На достатньо детальному рівні описано методи та властивості базових елементів керування, технологію розроблення проектів у середовищі Visual Basic. Значну увагу приділено програмуванню офісних застосувань мовою Visual Basic for Application. Навчальні програми та проекти, описані у посібнику, протестовано автором в інтегрованому середовищі Visual Basic 6 та MS Office 2000/XP. З метою опису виконання послідовності операцій у середовищі VB використовуватимемо систему позначень, запозичену у [13 – 15]:

Позначення	Виконання операції
➢**Назва команди**	**Вибір** у поточному меню команди з певною назвою (встановити курсор миші на команду і натиснути на *ліву кнопку миші* – **ЛКМ**)
❑**Назва кнопки**	**Натискання** кнопки з зазначеною назвою у активному діалоговому вікні (встановити курсор миші на кнопці та натиснути на **ЛКМ**)
⇩**Назва списку**	**Розкриття** списку (встановити курсор миші на кнопку розкриття списку, натиснути на **ЛКМ**)
⇗**Елемент списку**	**Вибір** елемента списку (встановити курсор миші на зазначений елемент, натиснути на **ЛКМ**)
🗀**Назва закладки**	**Вибір** закладки з зазначеною назвою (встановити курсор миші на ярлику закладки та натиснути на **ЛКМ**)
Назва поля:=*значення*	**Уведення** *значення* з клавіатури у текстове поле введення, список або лічильник. Значення лічильника можна змінювати і за допомогою кнопок-регуляторів
⊙**Назва перемикача** (радіокнопки)	**Вибір** перемикача з зазначеною назвою в активному діалоговому вікні (встановити курсор миші на перемикач і натиснути на **ЛКМ**)
☑**Назва індикатора** (прапорця, опції)	**Позначення** індикатора з зазначеною назвою в активному діалоговому вікні (встановити курсор миші на індикаторі та натиснути на **ЛКМ**)

1. Вступ у Visual Basic/VBA

План викладу матеріалу:

1. Загальна характеристика Visual Basic.
2. Робоче середовище Visual Basic 6 (VB 6).
3. Особливості програмування мовою Visual Basic.
4. Керування проектом Visual Basic.
5. Деякі засоби введення-виведення даних.
6. Редактор VBA.

Ключові терміни розділу

- *Visual Basic як інтерпретатор*
- *Visual Basic як компілятор*
- *Головне вікно середовища VB 6*
- *Панель інструментів*
- *Панель елементів*
- *Вікно форми*
- *Вікно властивостей*
- *Вікно проекту*
- *Вікно коду*
- *Вікно розкладки*
- *Рядки програми*
- *Коментарі*
- *Явне/неявне оголошення змінних*
- *Тип* `Variant`
- *Оператор присвоєння*
- *Процедури опрацювання подій*
- *Загальні процедури*
- *Модуль форми*
- *Програмний модуль*
- *Проект Visual Basic*
- *Функції* `MsgBox` і `InputBox`
- *Вікно негайного виконання*
- *Об'єкт* `Debug`
- *Метод* `Print`
- *Редактор VBA*
- *Проект VBA*

1.1. Загальна характеристика Visual Basic

Назва **Visual Basic** говорить сама за себе. Слово *Visual* означає, що у Visual Basic реалізовано *візуальний* стиль програмування: передусім створюють робоче середовище, а згодом розпочинають набирати перший рядок коду. Слово *Basic* у назві засвідчує, що синтаксис програм і оператори спираються на мову високого рівня *Basic* (Beginners Allpurpose Symbolic Instruction Code). Однак якщо ви знаєте звичайний Basic, то дуже швидко переконаєтеся, що Visual Basic помітно від нього відрізняється.

Visual Basic не можна однозначно зачислити ні до компіляторів, ні до інтерпретаторів (вважають, що він є "і тим, і іншим").

Головною ознакою інтерпретатора є те, що створені програми виконуються тільки у середовищі розробки. Програму можна запустити безпосередньо з середовища і якщо в ній є помилки, вони відразу ж розпізнаються. Усе це притаманне і Visual Basic, можна запустити програму безпосередньо у середовищі розробки. При цьому Visual Basic використовує технологію *Threaded-p-Code*, за якої кожен уведений рядок коду перетворюється у проміжний код Threaded-p-Code. Це ще не зовсім машинний код, проте такий код виконується швидше, ніж при роботі зі звичайним інтерпретатором. Visual Basic відразу ж перевіряє синтаксис програми і видає повідомлення щодо виявленої помилки.

Однак Visual Basic не просто інтерпретатор, оскільки це означало б, що програми виконуються тільки у його власному середовищі. Visual Basic дає змогу створювати і *exe*-файли, тому його можна зачислити і до компіляторів. Visual Basic не можна назвати чистим компілятором, оскільки на відміну, наприклад, від Visual C++, Visual Basic не створює файл, що виконується відразу ж при запуску з середовища розробки. Створення такого файла необхідно зробити явно (команда ➢`File` ➢`Make *.exe`). Починаючи з п'ятої версії, Visual Basic має *Native Compiler*, тобто компілятор, який може створювати машинний код. Отже, Visual Basic об'єднує у собі можливості як інтерпретатора, так і компілятора. І це має більше переваг, ніж недоліків.

1.2. Робоче середовище Visual Basic 6

Запустити Visual Basic можна за допомогою команди меню або подвійним клацанням миші на піктограмі програми. Після запуску Visual Basic на екрані з'являється діалогове вікно, в якому можна вибрати тип застосування, що створюється. З цього ж вікна можна завантажити вже наявний проект. За деякими піктограмами діалогового вікна переховуються майстри (*Wizards*), які супроводжують розробника при створенні застосувань і беруть на себе частину його роботи (наприклад, підключення бази даних або створення форми).

Один з головних майстрів – майстер застосування Visual Basic, за допомогою якого можна створити базовий "каркас" для звичайних Windows-додатків. Такий додаток створюється за допомогою елемента **`Standard EXE`**.

Робоче середовище VB 6 є багатодокументним (MDI – *Multiple Document Interface*) і містить декілька вікон (рис. 1.1). Усі вікна підпорядковані головному вікну (містить рядок меню) Visual Basic і можуть "прикріплятися" до одного з його країв.

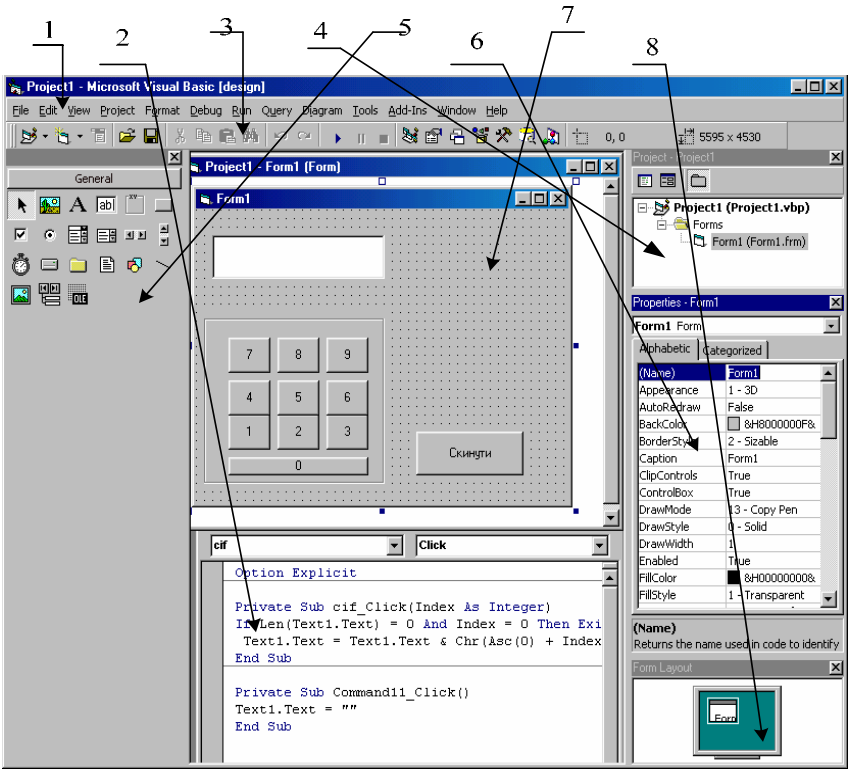


Рис. 1.1. Робоче середовище Visual Basic 6:

1 – головне вікно; 2 – вікно коду; 3 – панель інструментів; 4 – вікно проекту; 5 – панель елементів; 6 – вікно властивостей; 7 – вікно форми; 8 – вікно розкладки

У верхній частині екрана розташована *панель інструментів* (`Toolbar`). Її можна настроювати, як це зазвичай здійснюють у застосуваннях Microsoft. Кнопки, поля введення та інші *елементи керування*, необхідні для створення у застосуваннях, розташовані на *панелі елементів* (`Toolbox`).

Вікно форми (або ж *форма*) є головним елементом застосування. Форма – це контейнер для елементів керування. Точки сітки на формі допомагають розміщенню елементів і при роботі застосування не помітні.

Під час запуску Visual Basic форма, що відкривається на екрані, не містить елементів керування. Після натиснення на піктограмі необхідного елемента керування курсор миші набуває форми хрестика. Тепер необхідно зазначити на формі початковий кут елемента керування, натиснути ліву кнопку миші і, не відпускаючи її, встановити розмір елемента. Після досягнення потрібного розміру кнопка відпускається і в формі з'являється обраний елемент керування.

У *вікні властивостей* задаються властивості обраного елемента керування. У рядку заголовка вікна властивостей поруч з текстом `Properties` зазначають назву обраного елемента керування. Поле зі списком під рядком заголовка дає змогу обрати необхідний елемент керування. У списку, розташованому нижче, перелічено властивості цього елемента (в алфавітному порядку або за категоріями). Набір властивостей залежить від типу елемента керування. Список властивостей складається з двох стовпців: у правому перелічені назви властивостей, у лівому – їхні значення. Редагування властивості здійснюється або вручну (наприклад, введення назви елемента), або вибором відповідного поля зі списку, або за допомогою діалогового вікна настроювання властивості. Короткий опис обраної властивості відображається у нижній частині вікна.

У *вікні проекту* відображаються форми, модулі, класи та усі інші елементи додатку, згруповані за категоріями. У Visual Basic усі розроблені застосування називають *проектами*. Проект містить декілька груп компонентів (форми, модулі тощо). Усі застосування Visual Basic побудовано за модульним принципом, тому і об'єкт-

ний код складається не з одного великого файла, а з декількох частин. Декілька застосувань також можуть об'єднуватися у групи.

Щоб зберегти наявний елемент (форму, модуль тощо), треба виокремити його у списку вікна проекту та обрати команду ➢File ➢Save або ➢File ➢Save As. Для запису проекту загалом (у тім числі усіх компонентів) обирають команду ➢File ➢Save Project або ➢FiIe ➢Save Project As (для збереження проекту – під іншою назвою).

Щоб додати у проект новий елемент, необхідно виконати команду ➢Project ➢Add. Для видалення елемента треба виокремити його у вікні проекту і потім виконати команду меню ➢Project ➢Remove. Усі ці елементи зберігаються як окремі і незалежні файли. Тому їх можна у будь-який час завантажувати і зберігати. Це дає змогу використати в проекті форми і коди, створені для інших проектів, що економить робочий час.

Вміст вікна проекту зберігається у спеціальному файлі, що має розширення *.vbp* і містить список елементів, які треба завантажити у робоче середовище. Якщо декілька проектів об'єднано в групу, їхні назви зберігаються у файлі з розширенням *.vbg*.

Відразу після запуску Visual Basic *вікно коду* не відображається. Це вікно найважливіше у Visual Basic, оскільки саме у ньому вводиться програмний код. Код у Visual Basic розділяється на процедури і функції. Здебільшого процедури безпосередньо зв'язані з певними елементами керування. Це дає змогу відкрити вікно коду подвійним натисненням на елементі управління в формі.

Декілька слів про редактор коду Visual Basic. Прийоми роботи у ньому такі ж, як і при редагуванні текстів у інших застосуваннях Windows. Набрані символи вставляються на місці курсора уведення. Після натиснення клавіші *Insert* текстовий курсор приймає форму бруска, що засвідчує активізацію *режиму заміни*. Повторне натиснення клавіші *Insert* переводить редактор зворотно у *режим вставки*. Виокремлений текст при введенні замінюється новим.

Комбінація клавіш *Ctrl+X* видаляє виокремлений текст і вміщує його у буфер обміну Windows. Клавіші *Ctrl+C* слугують для копіювання тексту в буфер обміну, а *Ctrl+V* – для вставки з буфера

обміну. Окрім цього, комбінація клавіш *Ctrl+Y* вміщує у буфер обміну рядок, в якому перебуває текстовий курсор. Комбінація *Ctrl+N* вставляє перед поточним рядком порожній рядок. Окремі процедури можна переглядати за допомогою комбінацій клавіш *Ctrl+T*. Клавішею *Tab* створюється відступ у рядку або у всіх виокремлених рядках. Завдяки *Shift+Tab* текст зсувається праворуч.

Над вертикальною смугою прокрутки розташоване маленьке поле, яке можна перетягувати за допомогою миші вниз для розділення вікна на дві частини (*split window*). Це дає змогу редагувати в одному вікні дві різні процедури. Розділення відміняється, якщо роздільну лінію перемістити до самого краю вікна або виконати подвійне натиснення кнопки миші на роздільній лінії.

Вигляд і розмір шрифту можна змінювати на вкладці `Editor Format` діалогового вікна `Options` (команда меню ➢`Tools` ➢`Options`). За допомогою поля `Tab Width` вкладки `Editor` встановлюється число символів для відступу клавішею *Tab*. При встановленому прапорці `Auto Indent` натиснення клавіші *Enter* вміщує курсор введення у колонку, з якої розпочинався попередній рядок. Вибір `Default to Full Module View` дає змогу переглядати у вікні коду декілька процедур форми. Встановити і відмінити цей режим можна за допомогою кнопок, розташованих зліва від горизонтальної смуги прокрутки вікна коду.

Звичайною проблемою розробки прикладних застосувань у Visual Basic є різна роздільна здатність екранів монітора розробника і користувачів. Зазвичай застосування розробляють у середовищі, де монітор має високу роздільну здатність екрана. Це може спричинити незручності для користувача, що має монітор з низькою роздільною здатністю. У *вікні розкладки* можна встановлювати розмір і позицію форми на екрані, відображаючи допоміжні лінії для різних роздільних здатностей екрана.

1.3. Особливості програмування мовою Visual Basic

У кожному рядку коду програми на Visual Basic міститься один оператор. Visual Basic застосовує спеціальну технологію для перекладу коду – Threaded-p-Code – і тому відразу ж після натискання клавіші *Enter* може перевірити правильність написання коду.

Якщо код некоректний, то Visual Basic повідомляє щодо помилки. Крім того, у Visual Basic для багатьох процедур і функцій відображається підказка по синтаксису (*Tooltip*).

Visual Basic дає змогу розділяти логічний рядок, а, отже, й оператор, на кілька фізичних рядків. *Роздільником рядків* слугує пропуск, за яким записано символ підкреслення (_). Це допомагає форматувати довгі, важкодоступні для огляду рядки так, щоб вони цілком містилися на сторінці екрана. Наприклад, у цьому фрагменті один оператор вручну розбитий на два рядка:

```
Presenations.Open FileName:="D:\M\Prezent.ppt", _
                  ReadOnly:=msoFalse
```

Рядок програми у Visual Basic може містити максимум 1023 символи і не більше десяти роздільників – цього зазвичай достатньо. В одному рядку можна також записувати декілька операторів, розділяючи їх двокрапкою. Однак такий запис має сенс лише для дуже простих операторів, інакше програмний код буде нечитабельним. Кількість рядків коду (форми, модуля і т.п.) обмежено – 65534. Це обмеження неістотне, оскільки здебільшого кількість рядків у програмах є меншою.

У Visual Basic, як і у більшості мов програмування, можна використовувати *коментарі*. Вони призначені для пояснення окремих фрагментів програми й ігноруються Visual Basic при виконанні програми. Для виокремлення початку коментаря можна використовувати апостроф (**'**), чи оператор **Rem** – їхня дія однакова. Оператор **Rem** повинен перебувати в окремому рядку. Апостроф можна ставити у будь-якому місці рядка, при цьому текст коментаря розташовують праворуч:

```
Rem Це коментар
Print "Hello world" 'Це теж коментар
```

Visual Basic, на відміну від інших мов програмування, не вимагає *явного оголошення* змінних. Оператор **Dim** *явно* задає назву і тип змінної. Проте змінна може оголошуватися автоматично, коли вона просто з'являється у коді без будь-якого попереднього оголошення. Це – *неявне оголошення* змінної. За неявного оголошення поруч з назвою змінної можна вказувати знак відповідного типу (див. § 2.1). Якщо тип даних не ідентифікований знаком, то Visual

Basic застосовує тип **Variant**. Зауважимо, що змінні цього типу можна вводити і оператором **Dim**. *Оператор присвоєння* у Visual Basic має вигляд:

```
змінна=вираз
```

Тип **Variant** конкретизується залежно від значення змінної. Якщо змінна типу **Variant** має значення 5 (ціле число), то вона набуває тип **integer**; а якщо 1.2 — тип **Double**; якщо ж текст, то **String** (рядок). Змінна типу **Variant** змінює свій тип під час виконання програми. Ось простий приклад:

```
Dim V As Variant
V = "25"  'V містить рядок "25" (тип String)
V = V +5  'V містить 30 (число)
V = V & "штук" 'V містить рядок "30 штук"
```

Як бачимо, змінна V змінює тип даних залежно від значень. У другому рядку фрагмента програми змінній V присвоюється рядок "25" (*рядки у Visual Basic обмежують лапками*), тому V одержує тип String. У наступному рядку потрібно виконати додавання, що неможливо для змінних типу **String**. Тому Visual Basic намагається перетворити попереднє значення у числове; а це можливо для послідовності символів 2 і 5. Нарешті, до числа 25 можна додати 5. Внутрішній тип даних у цьому місці – **Integer**. В останньому рядку проводиться *об'єднання рядків* (позначення операції об'єднання – символ "&" з обох боків виокремлено пропуском). Унаслідок цього значення знову перетвориться у рядок символів.

У Visual Basic, як і в багатьох інших мовах програмування, увесь програмний код перебуває всередині підпрограм (процедур і функцій). *Процедура* – це підпрограма, яка починається оператором **Sub** і закінчується оператором **End**, між якими і розташовано код. Назва процедури опрацювання події складається з назв елемента керування і події, розділених символом підкреслення:

```
Private Sub Commandl_Click()
  ' Код опрацювання події натискання (Click)
  ' кнопки з назвою Commandl
End Sub
```

Можна створювати і власні (*загальні*) процедури. З цією метою переходять до секції *General-Declaration*. У вікні коду вводять **Sub**, потім назву, наприклад `MyProc`, і натискають клавішу *Enter*, унаслідок чого з'являється нова процедура:

```
Sub MyProc()
  ' Тут треба вводити код
End Sub
```

Ця процедура належить до секції *General*. Заголовок процедури закінчується порожніми дужками, однак там можна розміщувати параметри.

Приклад 1.1. Створити віконне застосування, яке виводить просте повідомлення за допомогою процедури `MsgBox` (див. наступний параграф). Для формування повідомлення використати загальну процедуру.

На формі розташуємо кнопку з написом "Повідомлення" (задаємо за допомогою властивості `Caption`). Властивість `Caption` для самої форми задамо як "Перший проект". Після цього форма виглядатиме приблизно так, як на рис. 1.2.

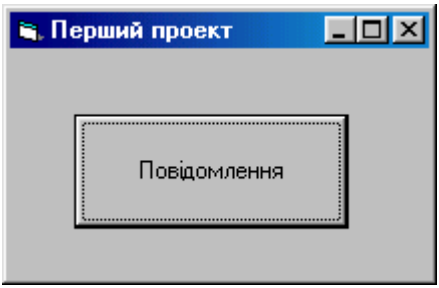


Рис. 1.2. Форма першого проекту

Розташування елементів керування на формі та модифікація самої форми належить до етапу *візуального програмування* (до етапу *проектування інтерфейсу користувача*). На етапі *кодування* запрограмуємо такі дві процедури:

```
Sub MyProc()
  MsgBox "Ви почали програмувати на VB. Вітаємо!"
End Sub
Private Sub Command1_Click()
  MyProc ' Виклик процедури MyProc
End Sub
```

Після цього збережемо наш перший проект за допомогою команди ➢File ➢Save Project або ➢FiIe ➢Save Project As. У діалозі послідовно збережуться файли: *форми* (Form1.frm) та *проекту* (Project1.vbp). Назви Form1 і Project1 у цьому випадку доцільно змінити на більш мнемонічні.

Отже, з кожною формою зв'язується *модуль форми*, що містить процедури опрацювання подій для елементів керування цієї форми та загальні (допоміжні) процедури. Модуль форми зберігається у файлі форми (Form1.frm). Файл форми зберігає також інформацію про елементи керування форми та саму форму.

З метою запуску програми виконаємо команди меню ➢Run ➢Start або натиснемо відповідну кнопку панелі інструментів. Після декількох секунд, які витрачаються на інтерпретацію та компонування програми, отримаємо результат (рис. 1.3).

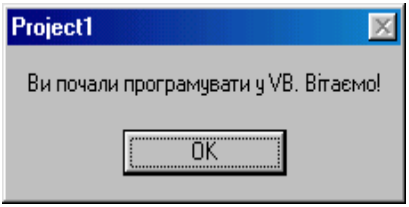


Рис. 1.3. Результат роботи програми

Підпрограми (процедури і функції) можна зберігати і у *програмному модулі*, який не зв'язується з жодною формою. Програмний модуль вводять за допомогою команд: ❑ПКМ {у вікні Project} ➢Add ➢Module. Файл, що зберігає програмний модуль, має стандартну назву Module1.bas (можна змінити при зберіганні). У програмному модулі зазвичай записують підпрограми, які мають загальне значення для модулів форм і проекту загалом.

Найпростіше застосування може містити тільки один програмний модуль (*псевдоконсольне застосування*). Серед підпрограм такого модуля має бути головна процедура main. Для видалення з проекту форми необхідно виконати команди: ⇗Form1{у вікні Project} ❑ПКМ ➢Remove Form1.

Приклад 1.2. Створити *псевдоконсольне* застосування, яке виводить просте повідомлення (див. рис. 1.3).

Вміст **програмного модуля**:

```
Sub Main()
  MsgBox "Ви почали програмувати на VB. Вітаємо!"
End Sub
```

1.4. Керування проектом Visual Basic

Усі застосування Visual Basic, навіть найпростіші, створюють як проекти. **Проект** – це контейнер, який налічує форми застосування та інші візуальні компоненти разом з програмним кодом. Отже, проект – це засіб інтеграції візуальних і програмних компонентів застосування. Проект Visual Basic треба зберігати в окремій папці. Усі компоненти зберігаються в пам'яті окремо і незалежно один від одного. Проект може налічувати такі компоненти:

- Файл форми (`*.frm`). Кожна форма має свій окремий файл.
- Файл форми з елементами керування, що містять бінарну інформацію (`*.FRX`). Такі файли створюються автоматично для форм, які містять елементи керування з властивостями `Picture` і `Icon`.
- Файл проекту, що містить посилання на свої компоненти (`*.VBP`). Після компіляції Visual Basic створює службовий файл доповнення до файла проекту (`*.VBW`).
- Файл кожного програмного модуля (`*.BAS`).
- Файл кожного модуля класів (`*.CLS`).
- Файли додаткових елементів керування (`*.OCX`).
- Максимум один файл ресурсів (`*.RES`). Файл ресурсів слугує для зберігання інформації (тексту, позначок, растрових зображень), модифікація якої не вимагає редагування коду.
- Файли, що залежать від вигляду проекту (`*.CTL` та інші).

Отже, компоненти проекту зберігаються окремо. У цьому можна переконатися, зберігаючи програму. З цією метою активізують у меню *File* команду *Save Project* чи клацають на панелі інструментів на піктограмі з зображенням дискети, унаслідок чого для кожної складової проекту з'являється діалогове вікно *Save As...* .

Сам компонент, наприклад форма, нічого не "знає" про інші компоненти проекту. Тому їх можна зачислювати і в інші проекти. З виконанням команди ➢File ➢Open Project компоненти завантажуються з урахуванням взаємозв'язків між ними. Кожен компонент проекту можна зберігати чи вилучати окремо. З цією метою необхідно виокремити його у вікні проекту та виконати команду ➢File ➢Save чи ➢Project ➢Remove. У проект можна додавати наявні компоненти за допомогою команди ➢Project ➢Add.

Щоб програма Visual Basic функціонувала не тільки у середовищі Visual Basic, її необхідно скомпілювати і скомпонувати. Для цього призначена команда меню ➢File ➢Make *.exe.

Однією з причин успіху Visual Basic є можливість використання так званих *Custom Controls* – елементів керування, розроблених сторонніми виробниками. У назві OCX перша літера позначає OLE (*Object Linking and Embedding*). Для зачислення елемента керування у проект необхідно викликати діалогове вікно ➢Project ➢Components. На вкладках *Controls*, *Designers* і *Insertable Objects* вікна обирати елементи керування, які необхідно додати до цього проекту.

У деяких випадках після завантаження проекту Visual Basic не відображає деякі вікна середовища розробника. З метою активізації "зниклого" вікна необхідно скористатися однією з команд меню View.

1.5. Деякі засоби введення-виведення даних

При організації інтерфейсу користувача досить часто використовують вбудовані діалогові вікна, знайомі нам із застосувань MS Office (відкриття та збереження файлів, пошуку, пошуку та заміни тощо). Такі вікна розглядатимемо згодом. Однак для організації діалогу з користувачем здебільшого достатньо обмежитися використанням вбудованих функцій MsgBox і InputBox.

1.5.1. Виведення повідомлень (функція MsgBox). Діалогове вікно, створене функцією MsgBox, слугує для виведення на екран повідомлення та отримання від користувача простої реакції на це повідомлення у вигляді клацання на одній з кнопок вікна. У найпростішому випадку цю функцію викликають як процедуру вигляду:

```
MsgBox(prompt[, buttons][, title][, helpfile, context])
```

- Параметр `prompt` (*повідомлення*) – вираз типу рядка символів, результатом якого може бути рядок довжиною до 1024 символів. Розбиття на окремі рядки у цьому виразі задають за допомогою вбудованої константи `vbCrLf`.
- Числовий параметр `buttons` *(кнопки)* задає вигляд командних кнопок, кнопку за домовленістю, вигляд піктограми вікна і модальність вікна. Його можна отримати як суму кодів кнопок та інших властивостей вікна. За домовленістю його значення дорівнює 0 (нулю). Коди кнопок та інших властивостей вікна функції `MsgBox` наведено у таблиці 1.1.
- Параметр `title` *(заголовок вікна)* – це вираз типу рядка символів, що задає заголовок діалогового вікна. За домовленістю його значенням є назва застосування.
- Параметри `helpfile` *(файл довідки)* і `context` *(контекст)* необхідно застосовувати тільки разом. Перший з них – назва файла, що містить довідку, яка виводиться шляхом натиснення клавіші *F1*, а другий – числовий вираз, що задає номер теми у цьому файлі.

Таблиця 1.1. Коди кнопок та інших властивостей вікна функції `MsgBox`

Код	Константа	Опис
1	2	3
Коди командних кнопок		
0	`vbOkOnly`	Кнопка `OK`
1	`vbOkCancel`	Кнопки `OK` + `Cancel`
2	`vbAbortRetryIgnore`	Кнопки `Abort` + `Retry` + `Ignore`
3	`vbYesNoCancel`	Кнопки `Yes` + `No` + `Cancel`
4	`vbYesNo`	Кнопки `Yes` + `No`
5	`vbRetryCancel`	Кнопки `Retry` + `Cancel`
Коди активності кнопок за домовленістю		
0	`vbDefaultButton1`	Активна перша кнопка
256	`vbDefaultButton2`	Активна друга кнопка
512	`vbDefaultButton3`	Активна третя кнопка

Закінчення табл. 1.1

2	2	3
Код піктограми вікна		
16	vbCritical	Важливе повідомлення
32	vbQuestion	Запит
48	vbExclamation	Попередження
64	vbInformation	Інформаційне повідомлення
Коди модальності вікна		
0	vbApplicationModal	Програмне модальне вікно
4096	vbSystemModal	Системне модальне вікно

Увага! У локалізованих версіях MS Office та VB назви кнопок російські: *Отменить*, *Да*, *Нет* і т.д.

Існує два типи модальних вікон – програмне модальне і системне модальне. Якщо вікно програми є програмним модальним, то користувач має закрити це вікно, якщо він хоче продовжити працювати з програмою; йому не обов'язково закривати його, якщо він хоче працювати з іншою програмою. Якщо вікно є системним модальним, то користувачеві заборонено працювати з будь-якими програмами доти, доки не закрите це вікно.

Щоб викликати у програмі функцію MsgBox, треба вирішити, з яких кодів складатиметься другий параметр.

Приклад 1.3. Виклик MsgBox з використанням стандартних зумовлених констант:

```
Public Sub main()
 Dim Code As String
 Code = vbYesNo + vbDefaultButton2 + vbExclamation
 MsgBox prompt:="Ви дійсно хочете завершити роботу?", _
   buttons:= Code, title:= "Просте діалогове вікно"
End Sub
```

Приклад 1.4. Виклик MsgBox з використанням числових кодів:

```
Public Sub main()
 MsgBox prompt:="Ви дійсно хочете завершити роботу?", _
```

```
   buttons:= 308,title:="Просте діалогове вікно"
End Sub
```

В обох випадках буде виведено одне і те ж саме діалогове вікно (рис. 1.4). Очевидно, що текст процедури `main` у прикладі 1.3. є значно читабельнішим і зрозумілішим.

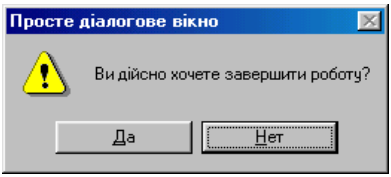


Рис. 1.4. Діалогове вікно

Щоб використовувати такі діалогові вікна у програмі, необхідно знати, яку кнопку натиснув користувач. З цією метою функція `MsgBox` повертає результат – числовий код, за яким можна визначити реакцію користувача. Значення кодів, які повертає функція `MsgBox`, подано у таблиці 1.2.

Таблиця 1.2. Коди значень, які повертає функція `MsgBox`

Код	Константа	Натиснута кнопка
1	vbOk	OK
2	vbCancel	Cancel
3	vbAbort	Abort
4	vbRetry	Retry
5	vbIgnore	Ignore
6	vbYes	Yes
7	vbNo	No

Приклад 1.5. Опрацювання реакції користувача:

```
Public Sub main()
 Dim Code As String, Prm As String, Ttl As String
 Dim Reply As Integer
 Prm = "Ви дійсно хочете завершити роботу?"
 Code = vbYesNo + vbDefaultButton2 + vbExclamation
 Ttl = "Просте діалогове вікно"
 Reply=MsgBox(prompt:=Prm,buttons:=Code, Title:=Ttl)
```

```
 Select Case Reply
  Case vbYes
            MsgBox "Так"
  Case vbNo
            MsgBox "Ні"
 End Select
End Sub
```

1.5.2. **Вікно введення даних** (функція `InputBox`). Вікно, яке створюється функцією `InputBox`, призначене для введення рядка символів. Це вікно містить поле редагування тексту, командну кнопку `OK` і командну кнопку `Отмена` (`Cancel`). Будь-який текст, набраний користувачем у полі редагування, буде повернено з функції, якщо користувач натиснув кнопку `OK`. Якщо користувач натисне кнопку `Cancel`, то буде повернено порожній рядок. Синтаксис виклику функції `InputBox`:

```
InputBox (prompt[,title][, default] [,xpos] [,ypos] _
          [, helpfile, context])
```

Параметри `prompt`, `title`, `helpfile` і `context` мають такий же сенс, що й у функції `MsgBox`. Параметр `default` містить деякий текст за домовленістю, який на початку відображатиметься у полі редагування тексту. Якщо користувач не введе у це поле новий текст, то вважатиметься, що він погоджується з текстом за домовленістю і цей текст повернеться у викликаючу програму. Числові вирази `xpos` і `ypos` задають відстань по горизонталі/вертикалі лівого верхнього кута вікна від лівого верхнього кута екрана. Якщо ці параметри не задано, то вікно виникає у центрі екрана.

Зауважимо, що результат функції `InputBox` є рядком символів. Якщо за логікою програми можна припустити, що користувач через цю функцію повертає число, то необхідно використовувати функцію `Val` для перетворення результату-рядка у число.

1.5.3. Виведення даних у вікно негайного виконання. Середовище розробки Visual Basic надає розробнику три вікна налагодження програми:

- вікно *контрольного значення* (Watch Window) відображає список виразів, які контролюють, та їхні поточні значення;
- вікно *негайного виконання* (Immediate Window) дає змогу виконувати однорядкові оператори у режимі налагодження;
- вікно локальних змінних (Locals Window) відображає усі оголошені змінні та їхні значення для поточної процедури.

Для створення псевдоконсольних застосувань дуже зручно використовувати вікно *негайного виконання*. Використовуючи об'єкт `Debug` і його метод `Print`, можна посилати повідомлення вікну негайного виконання саме з програмного коду. Виведені значення можна переглянути як під час роботи програми, так і після її зупинки. Об'єкт `Debug` є системним об'єктом, і тому ключове слово `Debug` не доцільно використовувати для задання назв інших об'єктів. Вікно негайного виконання можна також використати для циклічного виведення значень. Однак у цьому вікні зберігаються тільки останні 200 рядків. Вікно негайного виконання активізується послідовністю команд: ➢`View` ➢`Immediate Window`. Синтаксис методу `Print`:

```
Debug.Print список_виразів
```

Два сусідні елементи `списку_виразів` можна розділити *комою* або *крапкою з комою*. У першому випадку відповідні значення у вікні розділяють табуляцією, у другому – розташовують поряд. Якщо виклик методу `Print` завершується розділовим знаком (комою чи крапкою з комою), то з наступним викликом методу `Print` нові значення виводитимуться у тому ж рядку, що був при попередньому виклику методу `Print`. Для переходу на новий рядок під час виведення у вікно негайного виконання наприкінці оператора виклику методу `Print` не ставлять ніяких розділових знаків.

Приклад 1.6. Виведення даних у вікно негайного виконання:

```
Public Sub main()
  Dim st As String, Prm As String
  Dim Ttl As String, Def As String
  Dim x As Integer, y As Integer
  Prm = "Введіть ціле число!"
  Ttl = "Введення цілого числа"
  Def = "1"
  st = InputBox(Prm, Ttl, Def) : x = Val(st)
  st = InputBox(Prm, Ttl, Def) : y = Val(st)
  Debug.Print "x="; x, "y="; y,
  Debug.Print "x+y="; x + y
  Debug.Print "x-y="; x - y
End Sub
```

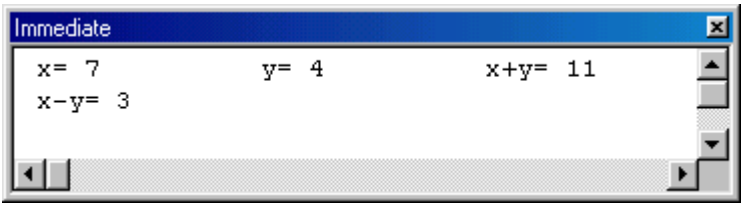


Рис. 1.5. Виведення результатів у вікно негайного виконання

1.6. Редактор VBA

Мова **VBA** (Visual Basic for Applications) – це мова Visual Basic, яка є внутрішньою мовою програмування у застосуваннях MS Office та інших програмних продуктах. Проекти VBA *зберігаються* безпосередньо у документах MS Word, робочих книгах MS Excel, презентаціях MS Power Point тощо.

Розробляючи застосування засобами VBA, використовують Редактор Visual Basic, який активізують командами ➢Сервис (Tools) ➢Макрос (Macro) ➢Редактор Visual Basic (Visual Basic Editor). Унаслідок запуску Редактора VBA в головновному вікні зазвичай відкриваються три вікна: вікно програми, вікно проекту і вікно властивостей.

У вікні програми (Code Window) вводиться і редагується код VBA. Якщо це вікно відсутнє, то його можна відкрити за допомогою меню Вид (View). Вікно програми має два режими перегляду: Представление полного модуля (Full Module View) і Представление процедуры (Procedure View), перемикання між якими здійснюється за допомогою кнопок у лівому куті вікна програми.

У режимі повного модуля у вікні програми одна за одною перелічені усі підпрограми і функції. Щоб додати нову підпрограму/функцію, необхідно розташувати текстовий курсор за останньою підпрограмою/функцією, задати ключове слово **Sub** або **Function**, а тоді ввести назву підпрограми/функції і натиснути клавішу <Enter>. Ключові слова **End Sub** або **End Function** Редактор Visual Basic додає автоматично.

У режимі подання процедури у будь-який момент часу відображається тільки одна процедура. Щоб перейти до іншої процедури, необхідно вибрати її у списках, розташованих у верхній частині вікна програми. У лівому списку перелічено об'єкти, які містять процедури, у правому – процедури відповідного об'єкта. Ці списки можна використовувати в обох режимах відображення коду у вікні програми.

Під час введення програми Редактор Visual Basic може відображати синтаксис властивостей, методів, операторів і функцій. У деяких випадках Редактор Visual Basic пропонує на вибір список значень, з якого можна обрати необхідне значення за допомогою клавіш <Enter> або <Space>.

Щоб переконатися, що використано всі можливі підказки Редактора Visual Basic, треба виконати команди ➢Сервис (Tools) ➢Параметры... (Options) і на вкладці Редактор встановити прапорці для списку компонентів (Auto Quick Info) і підказок значень даних (Auto Data Tips).

Частиною архітектури документа є *програмний проект* –сукупність модулів. Модулі, які формують програмний проект, можуть бути таких типів: модулі, що містять *процедури опрацювання подій* (кожна форма має власний модуль); *програмні* модулі; модулі *класів*; модулі *макросів*. Отож проект VBA нагадує проект VB.

Як тільки у проект додають нову форму, зразу ж утворюється зв'язаний з нею модуль. Процедури опрацювання подій задає програміст, однак каркас таких процедур автоматично формує Редактор VBA. Щоб отримати каркас процедури опрацювання події, необхідно визначити її у списках, розташованих у верхній частині вікна програми. У лівому списку перелічено об'єкти, а у правому – події відповідного об'єкта.

? Запитання для самоперевірки

1. Що таке панель інструментів?
2. Що таке панель елементів?
3. З якою метою використовують вікно форми?
4. З якою метою використовують вікно властивостей?
5. Що відображає вікно проекту?
6. Що відображає вікно коду?
7. З якою метою використовують вікно розкладки?
8. Як розділити логічний рядок програми на два фізичні?
9. Як задати явне оголошення змінних?
10. Що таке неявне оголошення змінних?
11. Опишіть тип `Variant`.
12. Як формується назва процедури опрацювання події?
13. Що таке загальна процедура?
14. Що таке модуль форми?
15. Що таке програмний модуль?
16. Опишіть функцію `MsgBox`.
17. Опишіть функцію `InputBox`.
18. Як активізувати вікно негайного виконання?
19. Що таке об'єкт `Debug`?
20. Опишіть метод `Print`.
21. Як активізувати редактор VBA?
22. Де зберігається проект VBA?

💻 Завдання для програмування

Завдання 1.1. Скласти *псевдоконсольну* програму, яка анкетує користувача (4-5 запитань) і робить деякі тривіальні висновки. Наприкінці вивести дані про автора програми. У програмі використати функції `MsgBox`/`InputBox` та вікно негайного виконання.

2. Типи даних і вирази

План викладу матеріалу:

1. Змінні. Прості типи даних.
2. Явне оголошення змінних.
3. Область визначення та час існування змінних.
4. Константи.
5. Присвоєння значень змінним. Вирази.
6. Вбудовані функції.

Ключові терміни розділу

- *Змінна, назва змінної*
- *Логічний тип*
- *Раціональні типи*
- *Тип дати і часу*
- *Спеціальні символи типу*
- *Явне оголошення змінних*
- *Область визначення змінних*
- *Константи*
- *Види і пріоритет операцій*
- *Математичні функції*
- *Функції перевірки типів*
- *Поняття типу даного*
- *Цілі типи*
- *Грошовий тип*
- *Тип рядка символів*
- *Неявне оголошення змінних*
- *Початкова ініціалізація змінних*
- *Час існування змінних*
- *Літерали*
- *Правила обчислення виразів*
- *Функції дати і часу*
- *Функції перетворення типів даних*

2.1. Змінні. Прості типи даних

Змінна представляє зарезервоване місце в оперативній пам'яті для тимчасового зберігання даних. Кожна змінна має власну назву, яку обирають довільно, дотримуючись таких правил:

- назва розпочинається з літери;
- максимальна довжина назви – 255 символів;
- назва може містити літери, цифри і символ підкреслення (_); великі та малі літери у назві не розрізняються;
- назва не може бути зарезервованим висловом Visual Basic.

Сьогодні при розробці великих проектів у найменуваннях змінних рекомендовано використовувати префікси, що відображають тип змінної та область її дії. У цьому випадку підвищується читабельність програми і зменшується кількість помилок.

Для явного оголошення змінної найчастіше використовують оператор **Dim**, що має такий синтаксис:

```
Dim назва_змінної [As тип_даних]
```

Якщо `тип_даних` не зазначено, то за домовленістю приймають тип **Variant**.

Увага. У назвах змінних та інших елементів мови Visual Basic можна використовувати кириличні символи. З огляду на перенесення програм цим не варто зловживати. Тому надалі кириличні символи використовуватимемо у коментарях і рядках символів.

Приклад 2.1. Оголошення змінних:

```
Dim FName As String
Dim Pr As Currency
Dim Coun As Integer
```

Можна задати і коротший опис змінних:

```
Dim FName As String, Pr As Currency, Coun As Integer
```

Тип даних вивчає:

- область можливих значень типу;
- структуру організації даних;
- операції, які можна використовувати над даними цього типу.

Типи поділяють на *прості* (скалярні) і *складні*. Складні типи вирізняються за способом структуризації даних простих типів. Окрім того, типи даних поділяють на *вбудовані* та *визначені користувачем*. Вбудовані типи початково належать мові програмування та формують її базу. За вбудованими типами користувач може будувати свої типи, однак правила побудови цих типів також закладено у мову. Мова Visual Basic підтримує такі типи даних.

- Дані типу **Boolean** можуть містити значення **True** чи **False**. Значенню **True** відповідає –1, а **False** – значення 0. Якщо змінній цього типу присвоєно значення 0, то вона містить **False**. Усі інші значення дають **True**.

  Приклад 2.2. Присвоєння значення логічній змінній:

  ```
  Dim nBool As Boolean
  nBool = 5  'Результат: True
  ```

- Дані типу **Byte**, **Integer**, **Long** містять лише цілі числові значення з різних діапазонів. Якщо змінній такого типу присвоєно 1.4, то повертається 1, якщо 1.5— повертається 2.

  Приклад 2.3. Присвоєння значення цілій змінній:

```
Dim nVar As Integer
nVar = 1.7  ' Результат: 2
```

- Дані типу **Single** і **Double** містять *раціональні* числа (або числа з плаваючою комою) з різних діапазонів значень. Дані типу **Currency** (грошовий тип) також слугують для представлення раціональних чисел, однак число розрядів після коми обмежено чотирма. Цього досить при виконанні грошових розрахунків. Роздільником цілої та дробової частин числа є крапка.
- Дані типу **Date** спеціально призначені для обробки інформації про дату і час. Значення дати і/чи часу можна помістити між двома знаками (#). Тоді при введенні необхідно користуватися американським форматом дати і часу. Якщо ж при введенні даних цього типу використовувати лапки ("), то застосовується формат дати і часу, встановлений у системі.

  Приклад 2.4. Присвоєння значень змінній типу **Date**:

```
Dim dtVar As Date
dtVar = #10/6/99#    ' Результат: 6.10.99
dtVar=#1:25:00 PM#   ' Результат: 13:25:00
dtVar = "6.10.99"    ' Результат: 6.10.99
dtVar = "13:25"      ' Результат: 13:25:00
```

- Дані типу **String** слугують для збереження рядків символів. Кожен символ займає один байт пам'яті. Операційні системи різних платформ підтримують різну максимальну довжину рядка. Рядок беруть у лапки.

  Приклад 2.5. Присвоєння значень змінній типу рядка символів:

```
Dim s As String
s = "Hello world" ' Результат: Hello world
```

  Довжину змінної типу **String** зазвичай обмежено лише операційною системою. Проте, за необхідності, її можна зазначити

явно (зафіксувати). З цією метою після слова **String** додають зірочку і максимально припустиме число символів:

```
Dim назва_змінної As String[* число_знаків]
```

Приклад 2.6. Оголошення рядка фіксованої довжини:

```
Dim FirstName As String* 30
```

Тип даних при оголошенні можна встановлювати додаванням спеціального символу типу до назви змінної (табл. 2.1).

Таблиця 2.1. Спеціальні символи типу

Тип змінної	Спеціальний символ типу	Приклад
Integer	%	Coun%
Long	&	NrS&
Single	!	Result!
Double	#	Number#
Currency	@	Summa@
String	$	FName$

Приклад 2.7. Оголошення типу змінних за допомогою спеціальних символів типу (див. для порівняння приклад 2.1):

```
Dim FName$, Pr@, Coun%
```

Не усі типи даних мають у своєму розпорядженні власні спеціальні символи. Microsoft більше не рекомендує використовувати ці символи для позначення типу. Вони у Visual Basic слугують тільки для сумісності з попередніми версіями. З метою усунення можливої двозначності при встановленні типу змінної не рекомендують використовувати ці спеціальні символи у назвах змінних.

Visual Basic, на відміну від інших мов програмування, не вимагає явного оголошення змінних. Оператор **Dim** явно задає назву і тип змінної. Проте змінна може з'являтися у коді без попереднього оголошення, що називають *неявним оголошенням* змінної. Виходячи з цього, випливає, що такі коди еквівалентні:

```
Dim Price As Currency      і      Price@ = 523
Price= 523
```

При неявному оголошенні поруч з назвою змінної можна зазначати символ відповідного типу. Якщо тип даних не зазначено, то незалежно від способу оголошення VB застосує тип **Variant**.

Змінні типу **Variant** мають велике практичне значення, однак при їхньому застосуванні виникають проблеми. По-перше, при читанні коду не видно, який внутрішній тип має змінна у цей момент, що може вкрай утруднити виявлення логічних помилок програмування. По-друге, дані цього типу унаслідок безперервних внутрішніх перетворень займають більше пам'яті, ніж аналогічні дані, оголошені з зазначенням явного типу.

Наведемо таблицю 2.2, у якій представлено усі вбудовані прості типи даних VB та діапазони відповідних значень цих типів.

Таблиця 2.2. Вбудовані прості типи даних Visual Basic

Тип даних	Розмір (байти)	Діапазон
Boolean	2	Логічні значення `True` або `False`
Integer	2	Цілі числа в межах від –32 768 до 32 767
Byte	1	Цілі числа в межах від 0 до 255
Long	4	Цілі числа в межах від –2 147 483 648 до 2 147 483 647
Single	4	Числа з плаваючою крапкою, що набуває значення (приблизно від $-3{,}4\cdot10^{38}$ до $-1{,}4\cdot10^{-45}$ для від'ємних значень і від $1{,}4\cdot10^{-45}$ до $3{,}4\cdot10^{38}$ для додатних значень)
Double	8	Числа з плаваючою крапкою, що набувають значень (приблизно від $-1{,}79\cdot10^{308}$ до $-4{,}9\cdot10^{-324}$ для від'ємних значень і від $4{,}9\cdot10^{324}$ до $1{,}79\cdot10^{308}$ – для додатних значень)
Currency	8	Числа з фіксованою десятковою крапкою і чотирма цифрами у дробовій частині в межах від –922 337 203 685 477,5808 до 922 337 203 685 477,5807
Date	8	Значення дати/часу в межах від 01.01.100 до 31.12.9999
String	Змінний	Будь-який рядок від 0 до 65535 символів
Object	4	Покажчик на будь-який об'єкт
Variant	Змінний	Універсальний тип, значенням якого слугують будь-які дані (тип за домовленістю)

2.2. Явне оголошення змінних

Важливо пам'ятати, що змінні можна оголошувати на двох рівнях – на *рівні методу* (процедури/функції) і на *рівні модуля*. Для оголошення змінних використовуються оператори **Dim**, **Private**, **Public**, **Static**. Оператор **Dim** можна використовувати на обох рівнях, **Private** і **Public** – на рівні модуля, а **Static** – тільки на рівні методу. Оголошення змінної задається оператором:

```
{Dim|Private|Public|Static} назва_змінної [As тип_даних] _
            [, назва_змінної [As тип_даних]]...
```

Приклад 2.8. Оголошення декількох змінних в одному операторі:

```
Dim v, j As Integer ' v типу Variant
Private Cost As Currency, i As Long
```

Явне оголошення змінних має великі переваги перед неявним: воно наочніше, поліпшує читабельність програми та виявлення помилок тощо. Щоб змінні завжди оголошувалися явно, необхідно задати опцію **Explicit**. У цьому випадку Visual Basic вимагатиме явного оголошення змінних, що усуває можливі помилки при написанні програми.

З метою активізації цієї опції необхідно відкрити діалогове вікно Options (команда меню ➢Tools ➢Options...) і на вкладці Editor установити опцію Require Variable Declaration. Відповідно в усі створювані компоненти (форми, модулі, класи) автоматично вставляється рядок:

```
Option Explicit
```

Після установлення цієї опції Visual Basic вимагає явного оголошення усіх змінних і з виникненням неоголошеної змінної видає повідомлення про помилку. Проте це не стосується раніше створених компонентів: форм, модулів чи класів. Для вирішення цієї проблеми необхідно вручну додати рядок **Option Explicit** у секцію (*General-Declarations*) цих компонентів.

У будь-який момент, коли Visual Basic створює нову змінну, ця змінна отримує початкове значення (*ініціалізується*):

- числові змінні отримують значення 0 (нуль);
- рядки отримують нульову довжину (порожні рядки);

- логічні змінні отримують значення **False**;
- змінні типу **Object** отримують значення **Nothing** (нічого);
- змінні типу **Date** отримують значення 30 грудня 1899;
- змінні типу **Variant** до першого присвоєння мають значення **Empty** (порожньо). Це спеціальне значення, відмінне від нуля і рядка нульової довжини. За допомогою функції `IsEmpty` можна виявити, чи було присвоєння значення змінній типу **Variant** (повертає значення **True** або **False**). У деяких випадках можна очистити вміст змінної типу **Variant**, присвоївши їй значення **Empty**.

Інше специфічне значення, яке набувають змінні типу **Variant**, – це **Null**. Значення **Null** використовують у базах даних для зазначення відсутності даних, або даних невідомого типу. Для перевірки змінних типу **Variant** можна використовувати функцію `IsNull`. Присвоєння значення **Null** змінним іншого типу, ніж тип **Variant**, зумовить виникнення помилки під час виконання програми.

2.3. Область визначення та час існування змінних

Visual Basic налічує три види областей визначення змінних: *локальну*, *контейнерну* та *глобальну*.

Локальна змінна доступна тільки у поточній підпрограмі.

Контейнерна змінна доступна тільки у поточних формі, модулі чи класі. Змінні контейнера визначаються у секції (*General-Declarations*) відповідного контейнера (форма, модуль чи клас).

Глобальна змінна доступна в усьому проекті. Глобальні змінні визначаються у секції (*General-Declarations*) модуля. У цьому випадку замість слова **Dim** використовують зарезервоване слово **Public**. Глобальні змінні доступні в усіх модулях і процедурах проекту. Термін *глобальна* змінна походить зі старих версій Visual Basic, де замість слова **Public** використовували слово **Global**. Однак вже у п'ятій версії Visual Basic зарезервоване слово **Global** у цьому контексті більше не використовують.

Локально оголошені змінні при виході з процедури видаляються з пам'яті, а при новому виклику процедури ініціалізуються

заново. Їхні значення у цьому випадку не зберігається, що не завжди доречно. Visual Basic дає змогу оголошувати змінні як *статичні*. Під час виходу з процедури значення статичної змінної зберігається. У випадку повторного виклику цієї процедури статична змінна набуває значення, яке вона мала за останнього виходу з цієї процедури. Для оголошення статичної змінної необхідно замість слова **Dim** використовувати слово **Static**.

Приклад 2.9. Використання статичної змінної:

```
Private Sub Commandl_Click()
Static A As Integer ' Початкове значення 0
Dim Y As Integer    ' Початкове значення 0
A = A + 1 'Збільшення попереднього значення на 1
Y = Y + 1 'Значення Y рівне 1
…
End Sub
```

Щоб оголосити статичними всі локальні змінні підпрограми, необхідно записати слово **Static** у заголовку підпрограми.

Приклад 2.10. Оголошення статичними усіх локальних змінних:

```
Static Sub Test ()
Dim A, Y As Integer 'A типу Variant! A і Y –статичні
…
End Sub
```

2.4. Константи

Головна відмінність *константи* від змінної полягає у тому, що її значення не можна змінювати у процесі виконання програми. Вона завжди зберігає своє початкове значення. Розрізняють константи (як і змінні) *контейнера*, *локальні* й *глобальні*. При оголошенні констант використовують ключове слово **Const**. Глобальна константа оголошується як **Public**; при цьому глобальні константи можна оголошувати тільки у модулі. Синтаксис оголошення константи:

[**Public** | **Private**] **Const** [**As** тип] назва_константи = значення

Як значення припустиме використання тільки постійних значень (літералів) та їхніх комбінацій, у тім числі арифметичні і/або логічні оператори, проте не функції.

Приклад 2.11. Оголошення констант:

```
Const Pi As Double = 3.14159265
Const Size = 255
Const EndDate = #31.12.98#
Const MaxSize = Size*10
Public Const ArrayLimit = 100 ' Глобальна конст.
```

Мова VB/VBA володіє багатьма вбудованими константами (вони мають префікс "vb"). Константи з бібліотеки MS Access мають префікс "ac", а константи з бібліотеки MS Excel – префікс "xl". Вони не тільки поліпшують розуміння тексту програм, а й забезпечують сумісність додатків з новими версіями Visual Basic, оскільки, зазвичай, змінюється фактичне значення константи, а не її назва.

Інформацію щодо наявних констант, їхніх значень і застосування можна отримати, звертаючись до відповідних розділів довідки або користуючись каталогом об'єктів (Object Browser).

2.5. Присвоєння значень змінним. Вирази

Присвоєння значень змінним здійснюється за допомогою оператора присвоєння, у якому зліва стоїть змінна, а справа – значення, що їй присвоюється, або вираз, а саме:

```
[Let] Змінна= Вираз
```

Ключове слово Let зазвичай опускають. Під змінною розуміють назву простої змінної, елемент масиву, поле запису або властивість об'єкта. Тип виразу повинен відповідати типові змінної. Змінним типу **Variant** можна присвоювати значення різних типів. Змінним типу **String** можна присвоювати будь-які значення типу **Variant**, окрім **Null**. Числовій змінній значення типу **Variant** можна присвоювати тільки у випадку, якщо його можна перетворити до числа.

Операції та *вирази* задають певну послідовність дій, однак не є закінченими реченнями мови. *Прості вирази* містять знак операції та операнди, наприклад 3.14+x. Операції можуть виконуватися з одним або двома операндами. Відповідно, розрізняють *унарні* та *бінарні* операції. Операндами слугують *літерали*, *константи*, *змінні*, *виклики функцій* і *вирази*.

Про *константи* та *змінні* ми вже говорили. *Літерал* – це теж константа, проте без назви; її визначають з контексту програми. Розрізняють *числові літерали* (або просто *числа*), *літерали дати/часу* (наприклад: #1/15/03#), *рядки символів* (або просто *рядки*; наприклад: `"Це рядок"`) і *логічні* літерали (**`True`** чи **`False`**). Числа класифікують як *цілі* (наприклад: `22`, `–524`) та *раціональні* (наприклад: `3.1415`, `-3e8`).

Виклик функції – це вказівка назви функції, за якою у круглих дужках записують список аргументів (може бути порожнім): `F(x+c, y)`.

Вираз – це послідовність знаків операцій, операндів і круглих дужок, який задає обчислювальний процес отримання результату певного типу. *Найпростішими* виразами є *літерали, константи*, *змінні* та *виклики функцій*.

Символи “+”, “–”, “*”, “^”, “/” і “\” – знаки *бінарних* арифметичних операцій додавання, віднімання, множення, піднесення до степеня, ділення і цілочисельного ділення, відповідно. Для цілих чисел існує операція **`mod`** – *ділення за модулем* (обчислення залишку від ділення). Наприклад: `14 mod 3=2`.

Прикладом *унарної* арифметичної операції є операція *зміни знака числа*, яку позначають символом “мінус”, що стоїть перед одним операндом: `–x+y`. Як бачимо, зміст операції залежить від контексту. Зміст операції залежить також від типу її операндів: “+” у виразі `a+b` означає додавання раціональних чисел, якщо операнди мають тип **`Double`**, проте це буде додавання цілих чисел, якщо вони мають тип **`Integer`**.

Порядок обчислення виразу визначають за розташуванням круглих дужок, знаків операцій і *пріоритетності* операцій. Операції з найвищим пріоритетом виконують першими. Якщо у виразі є декілька операцій одного і того ж пріоритету, то їх виконують зліва направо. У сумнівних випадках варто ставити круглі дужки. Перелік і пріоритетність операцій наведено у таблиці 2.3.

Таблиця 2.3. Перелік і пріоритетність операцій

Пріоритет	Знак операції	Назва операції
1	`()`	Виклик функції і дужки
2	`^`	Піднесення до степеня
3	`-`	Унарний мінус
4	`*` `/`	Множення Ділення
5	`\`	Цілочисельне ділення
6	**`mod`**	Залишок від цілочисельного ділення
7	`+` `-`	Додавання Віднімання

8	`<`
	`>`
	`<=`
	`>=`
	`=`
	`<>`
		Менше
		Більше
		Менше або рівне
		Більше або рівне
		Рівне
		Не рівне

9	**`Not`**	Логічне заперечення
10	**`And`**	Логічне І
11	**`Or`**	Логічне АБО
12	**`Xor`**	Додавання за модулем 2
13	**`Equ`**	Логічна еквівалентність
14	**`Imp`**	Логічна імплікація

Результатом будь-якої операції порівняння (восьмий пріоритет) є логічні значенняі **`True`** або **`False`**. Якщо обидва операнди мають однаковий тип, то виконують просте порівняння значень для цього типу. Якщо один чи обидва операнди мають тип **`Variant`**, то попередньо здійснюється спроба перетворити цей тип у деякий сумісний тип.

Порівняння рядків символів є набагато складнішим, ніж порівняння чисел. Два рядки символів є рівними тільки тоді, коли містять однакові символи у такому ж порядку і мають однакову довжину. VB враховує початкові й/або кінцеві пропуски у рядку і

порівнює кожний рядок зліва направо *посимвольно*. Результат порівняння визначається негайно, як тільки відповідні символи не збігаються. Порівняння символів відповідає звичному алфавітному порівнянню, за яким буква "а" вважається меншою за букву "б" і так далі.

VB дає змогу використовувати два різні способи порівняння однакових символів різних регістрів: *двійковий* та *текстовий*. За домовленістю, під час порівняння рядків, відповідні символи порівнюються за їхніми двійковими значеннями (кодами). У цьому випадку рядок `"CCC"` буде меншим за рядок `"ccc"`.

Оператор **`Option Compare Text`** на початку модуля задає алфавітне порівняння символів у рядках, що не залежить від регістру (у цьому випадку рядки `"CCC"` і `"ccc"` будуть рівними).

Операція **`Like`** дає змогу виконати порівняння рядка з *шаблоном*. Операцію **`Is`** використовують тільки для порівняння змінних типу **`Object`**. Усі ці операції детальніше опишемо у наступних розділах.

Як операцію об'єднання рядків у Visual Basic можна використовувати знак "підсумовування" (+) та знак "комерційне і" (&). Однак для кращої читабельності коду варто застосовувати тільки &, тому що знак "плюс" використовують зазвичай при підсумовуванні числових значень. Однак варто пам'ятати, що знак & функціонує і як ідентифікатор типу для змінних **`Long`**, якщо він знаходиться наприкінці назви. Отже, якщо для задання операції об'єднання рядків використано знак &, то його з обох боків необхідно виокремити пропусками.

2.6. Вбудовані функції

У Visual Basic є великий набір вбудованих функцій і процедур, використання яких суттєво спрощує програмування. Їх можна розділити на такі категорії:

- математичні функції;
- функції дати і часу;
- функції перевірки типів;
- функції перетворення типів даних;
- функції опрацювання рядків;

- функції опрацювання масивів;
- функції вибору.

У цьому розділі розглянемо перші чотири категорії функцій.

2.6.1. Математичні функції. Набір математичних функцій VBA досить стандартний. Перелічимо їх (`число` тут позначає *числовий вираз*):

`Abs(число)`	Абсолютне значення числа
`Atn(число)`	Арктангенс числа (у радіанах)
`Cos(число)`	Косинус числа. Аргумент – число у радіанах
`Exp(число)`	Експонента числа
`Log(число)`	Натуральний логарифм числа
`Rnd[(число)]`	Рівномірно розподілене випадкове число на проміжку `[0; 1]`
`Sgn(число)`	Знак числа (для додатного числа результат становить 1; для від'ємного – -1; для нуля – 0)
`Sin(число)`	Синус числа. Аргумент – число у радіанах
`Sqr(число)`	Квадратний корінь з числа
`Tan(число)`	Тангенс числа. Аргумент – число у радіанах

Приклад 2.12. Створити псевдоконсольне застосування для обчислення функції $y = \sqrt{\ln^2(\sin x + 2) + x^2}$ у заданій точці:

```
Option Explicit
Public Sub main()
   Dim st  As String, Prm As String
   Dim Ttl As String, Def As String
   Dim x As Double, y As Double
   Prm = "Введіть  число!"
   Ttl = "Введення  числа"
   Def = "1"
   st = InputBox(Prm, Ttl, Def): x = Val(st)
   y = Sqr(Log(Sin(x) + 2) ^ 2 + x * x)
   Debug.Print "x="; x, "y="; y
End Sub
```

2.6.2. Функції роботи з датами і часом

Date	Повертає поточну системну дату
DateSerial(рік, місяць, день)	Повертає дату у внутрішньому форматі, для зазначеного року, місяця і дня
DateValue(рядок символів)	Перетворює рядок символів, що є зображенням дати, до вигляду дати у внутрішньому форматі
Day(дата)	Повертає день місяця з дати (ціле число від 1 до 31)
Hour(час)	Повертає ціле число від 0 до 23, яке показує годину
Minute(час)	Повертає число хвилин
Month(дата)	Повертає число місяця з зазначеної дати (ціле число від 1 до 12)
Now	Повертає поточну дату і час
Second(час)	Повертає число секунд
Time	Повертає поточний час
Timer	Повертає число секунд, яке пройшло з півночі
Year(дата)	Повертає рік із зазначеної дати

Зрозуміло, що під датою, часом, годинами, хвилинами тощо розуміють вирази відповідних типів.

2.6.3. Функції перевірки типів даних

IsArray(змінна)	Повертає значення **True**, якщо змінна є назвою масиву, і **False** – інакше
IsDate(вираз)	Повертає значення **True**, якщо значення виразу можна перетворити у дату, і **False** – інакше
IsEmpty(змінна)	Повертає значення **True**, якщо змінна ініціалізована, і **False** – інакше
IsError(вираз)	Повертає значення **True**, якщо значення виразу є номером помилки, і **False** – інакше
IMissing(аргумент)	Повертає значення **True**, якщо аргумент відсутній, і **False** – інакше
IsNull(вираз)	Повертає значення **True**, якщо вираз не містить даних, і **False** – інакше
IsNumeric(вираз)	Повертає значення **True**, якщо значення виразу є числом, і **False** у протилежному випадку

`IsObject(змінна)`	Повертає значення **`True`**, якщо `змінна` є об'єктною змінною, і **`False`** – інакше
`TypeName(змінна)`	Повертає назву типу `змінної`

2.6.4. Функції перетворення типів. VBA містить функції, що дають змогу перетворювати значення одного типу у значення іншого типу. Наприклад, функція `Val` перетворює рядок, який містить цифри, у число, а функція `Str`, навпаки, перетворює число у рядок цифр. Окрім того є функції, які перетворюють значення виразів у значення конкретних типів:

`CBool(вираз)`	Перетворює `вираз` у значення типу **`Boolean`**. Якщо вираз дорівнює нулю, то функція повертає значення **`False`**, у протилежному випадку повертає **`True`**. Якщо вираз не можна інтерпретувати як числове значення, то виникає помилка під час виконання програми
`CCur(вираз)`	Перетворює значення `виразу` у тип **`Currency`**. Якщо значення виразу знаходиться поза допустимим діапазоном типу даних **`Currency`**, то виникне помилка під час виконання програми
`CDate(вираз)`	Перетворює значення `виразу` у тип **`Date`**
`CDbl(вираз)`	Перетворює значення `виразу` у тип **`Double`**
`CInt(вираз)`	Перетворює значення `виразу` у тип **`Integer`**. Функція завжди округлює вираз до найближчого парного цілого. Якщо вираз знаходиться поза допустимим діапазоном типу `Integer`, то виникне помилка під час виконання програми
`CLng(вираз)`	Перетворює значення `виразу` у тип **`Long`**
`CSng(вираз)`	Перетворює значення `виразу` у тип **`Single`**
`CStr(вираз)`	Перетворює значення `виразу` у тип **`String`**
`CVar(вираз)`	Перетворює значення `виразу` у тип **`Variant`**

? Запитання для самоперевірки

1. Що таке ідентифікатор?
2. Що розуміють під типом даних?

3. Що повідомляється компіляторові при визначенні змінних/констант?
4. Що таке область визначення змінних?
5. Що таке час існування змінних?
6. Зазначте і охарактеризуйте цілі типи даних.
7. Зазначте і охарактеризуйте раціональні типи даних.
8. Що таке логічний тип?
9. Як представляються числа?
10. Як представляються рядки символів?
11. Що таке константи?
12. Що таке вираз? Зазначте правила обчислення виразу.
13. Наведіть приклади операцій з однаковим пріоритетом.
14. Наведіть приклади операцій порівняння.
15. Що таке логічне І/АБО?
16. Наведіть приклади унарних операцій.
17. Наведіть приклади функцій перетворення типів даних.
18. Наведіть приклади функцій перевірки типів.
19. Наведіть приклади математичних функцій.

💻 Завдання для програмування

Завдання 2.1. Скласти програму обчислення функції (функцію обирати за номером студента у списку студентів підгрупи) у заданій точці:

1. $y = \cos^3(2x) - \log_2(x^2 + 2).$
2. $y = \sin^4(3x) - 2e^{x-2}.$
3. $y = \cos^2(2x) - \sin x.$
4. $y = \sin^3(3x) - \cos x.$
5. $y = \sqrt{x^2 + 3} - 2\log_5(x^2 + 1).$
6. $y = \sqrt{x^3 + 3x} + 2\sin x.$
7. $y = (x^2 - 2x)^3 - 4\log_3(x^4 + 1).$
8. $y = \sqrt[3]{x^4 + 3x^3} - 2\sin x.$
9. $y = 2^{x^3 + 2x} - 4\cos x.$
10. $y = 3^{x^3 - 2x} + 4\log_4(x^2 + 3).$
11. $y = \sin^{\frac{2}{5}}(3x) + \cos x.$
12. $y = \sin^{\frac{4}{7}}(3x) + 2\cos x.$
13. $y = \sqrt[5]{x^5 + 3x^3} + 2\sin x.$
14. $y = \sqrt[3]{x^5 + 5x^4} - 4x^{\frac{2}{3}}.$
15. $y = \sqrt{x^7 + 5x} + 10\sin x.$

3. Галуження і цикли

План викладу матеріалу:

1. Оператори і визначення.
2. Оператори галуження. Безумовна передача керування.
3. Оператор вибору.
4. Оператори циклу.
5. Програми з простим повторенням.
6. Програмування задач цілочислової арифметики.
7. Сумування рядів.

Ключові терміни розділу

- *Оператори керування*
- *Оператор* **Goto**
- *Оператор циклу* **For–Next**
- *Оператори* **Exit** {**Do** | **For**}
- *Оператор циклу* **While–Wend**
- *Оператори галуження*
- *Оператор вибору*
- *Оператор* **For Each–Next**
- *Оператор циклу* **Do–Loop**
- *Вкладені цикли*

3.1. Оператори і визначення

У Visual Basic є великий набір операцій, за допомогою яких у виразах утворюються нові значення, які присвоюються змінним за допомогою *оператора присвоєння*. Потік керування у програмі задається *операторами керування* (умовного і безумовного переходу, вибору, циклів), а *визначення* (**Dim**, **Const**, **Public**/**Private**, **Static**) використовують для введення у програму назв змінних, констант, масивів тощо. Зауважимо, що *визначення є операторами*, а тому вони можуть розташовуватися у довільному місці програми.

Оператори керування функціонально еквівалентні аналогічним операторам інших мов програмування і дають користувачеві можливість створювати складні й потужні процедури. Оператор, який є тільки в Visual Basic – це оператор **For Each-Next**. Подаємо перелік найуживаніших операторів керування Visual Basic:

If-Then-Else	Перевіряє умову і змінює хід виконання програми залежно від результатів перевірки
Select-Case	Обирає один із можливих варіантів виконання програми залежно від значення змінної
For-Next	Виконує деякі дії певну кількість разів

While-Wend	Виконує деякі дії доти, доки справедлива умова
Do-Loop	Виконує деякі дії доти, доки справедлива умова, або доти, доки умову не буде виконано
For-Each-Next	Виконує деякі дії для кожного об'єкта у колекції або для кожного елемента масиву

3.2. Оператори галуження. Безумовна передача керування

До операторів галуження належать однорядковий і багаторядковий (блочний) оператори умовного переходу **If–Then–Else**. Оператор **If**–**Then**–**Else** виконує ту чи іншу послідовність операторів залежно від значення умовного виразу, тобто виразу, результатом якого буде **True** чи **False**. Якщо умова має значення **Null**, то їй присвоюється значення **False**. Синтаксис оператора умовного переходу:

```
If умова Then оператори [Else оператори-else]
```

або:

```
If умова Then
      [оператори]
[ElseIf умова-N Then
      [оператори-elseif]]
      …
[Else
      [оператори-else]]
End If
```

- `оператори`: один або кілька операторів, які виконуються, якщо `умова` має значення **True**;
- `умова-N`: те ж саме, що й `умова` – логічний вираз;
- `оператори-elseif`: один або кілька операторів, які виконуються, якщо відповідна `умова-N` має значення **True**;
- `оператори-else`: один або кілька операторів, які виконуються, якщо жодна з попередніх виразів `умова` чи `умова-N` не має значення **True**.

Однорядкову форму доцільно використовувати для коротких, простих перевірок. Блочна форма забезпечує дещо структурованіший підхід і є простішою для читання, оброблення та налагодження. У випадку, якщо `умова` має значення **True**, виконуються `опе-`

ратори після **Then**. Якщо умова має значення **False**, то почергово обчислюються умови **ElseIf** (якщо такі є). Якщо виникне умова-N зі значенням **True**, то будуть виконані оператори-elseif, розташовані безпосередньо за відповідним **Then**. Водночас кожна наступна перевірка здійснюватиметься тільки у тому випадку, якщо результат попередньої дорівнюватиме **False**. Якщо жодна з умов-N не мала значення **True** (або ElseIf не задано), то виконаються оператори-else.

Увага! Припустиме вкладання блоків, тобто блоки **If** можуть перебувати в інших блоках **If**. Якщо після **Then** на тому ж рядку розташовано будь-який оператор, за винятком коментаря, то оператор **If** вважають однорядковим.

Приклад 3.1. Однорядковий умовний оператор:

```
If A>3 Then A=A+1: B=B+A: C=C+B Else B=A: C=2*A
```

Приклад 3.2. Обчислити $z = \max(x, y)$:

```
If x>y Then z=x Else z=y
```

Приклад 3.3. Обчислити $z = \max(x, y, t)$:

```
If x>y And x>t Then
    z=x
ElseIf y>t Then
    z=y
Else
    z=t
End If
```

Приклад 3.4. Скласти псевдоконсольне застосування обчислення величини $y = \begin{cases} -1, \text{ якщо } x < -2; \\ x+1, \text{ якщо } -2 \le x < 0; \\ x^2+1, \text{ якщо } 0 \le x \le 1; \\ 3x-1, \text{ якщо } x > 1. \end{cases}$

```
Option Explicit
Public Sub main()
   Dim st  As String, Prm As String
   Dim Ttl As String, Def As String
   Dim x As Double, y As Double
   Ttl = "Введення": Prm = "Введіть  число!"
   Def = "4"
```

```
    st = InputBox(Prm, Ttl, Def): x = Val(st)
    If x < -2 Then y = -1
    If x < 0  And x >= -2 Then y = x + 1
    If x <= 1 And x >= 0  Then y = x * x + 1
    If x > 1 Then y = 3 * x - 1
    Debug.Print "y="; y
End Sub
```

Під час роботи цієї програми завжди виконуються один за одним чотири умовних оператори. Істинною вважають тільки одну умову і, відповідно, присвоєння значення змінній *y* відбудеться тільки один раз. Запишемо умовні оператори так, щоб зменшити кількість порівнянь (другий варіант):

```
Option Explicit
Public Sub main()
   Dim st As String, Prm As String
   …
   st = InputBox(Prm, Ttl, Def): x = Val(st)
   If x < -2     Then
                      y = -1
   ElseIf x < 0  Then
                      y = x + 1
   ElseIf x <= 1 Then
                      y = x * x + 1
   Else
                      y = 3 * x - 1
   End If
   Debug.Print "y="; y
End Sub
```

Перевірка належності аргумента *x* наступному проміжку виконується тільки у тому випадку, якщо *x* не входить у попередній проміжок. Програма стала компактнішою та ефективнішою, проте менш наочною. Який варіант є кращим? У сучасній ієрархії критеріїв якості програм на першому місці стоїть надійність, простота підтримки та модифікації, а ефективність і компактність відходять на другий план. Якщо немає спеціальних вимог щодо швидкодії, рекомендують обирати найбільш наочний варіант (у нашому випадку – перший).

В умові галуження може стояти вираз:

```
TypeOf Назва_об'єкта Is Тип_об'єкта
```

тут назва_об'єкта – це будь-який вказівник на об'єкт. Вираз отримує значення **True**, якщо об'єкт має вказаний тип_об'єкта.

Приклад 3.5. Використовуючи оператор **If TypeOf**..., визначаємо тип елемента керування, що передається процедурі:

```
Sub ControlProcessor(MyControl As Control)
    If TypeOf MyControl Is CommandButton Then
          Debug.Print "Передано командну кнопку"
    ElseIf TypeOf MyControl Is CheckBox Then
          Debug.Print "Передано прапорець"
    ElseIf TypeOf MyControl Is TextBox Then
          Debug.Print "Передано поле"
    End If
End Sub
```

Оператор **GoTo** використовують для виконання безумовного переходу до іншого рядка у процедурі/функції. Синтаксис:

```
GoTo {Позначка | Номер_рядка}
```

Будь–який рядок з оператором можна позначити за допомогою позначки або номера_рядка. Позначка – деяка символьна назва, що розпочинається з букви і закінчується двокрапкою. Позначку ставлять перед відповідним оператором на початку рядка. Кожен номер_рядка має налічувати тільки цифри і відрізнятися від інших рядків у процедурі; перед ним у рядку можуть стояти тільки пропуски. Оператор GoTo дає змогу передати керування безпосередньо на потрібний оператор програми. Конкретною позначкою у програмі можна позначати тільки один оператор. Операторів переходу з однією і тією ж позначкою можна задати будь-яку кількість.

Використання позначок утруднює налагодження і тестування програми і є ознакою поганого стилю програмування. Тому позначки та оператори **GoTo** бажано не використовувати.

3.3. Оператор вибору

Умовний оператор дає змогу легко реалізувати вибір між двома варіантами. Однак інколи виникає необхідність обрати один варіант з декількох (більше двох) варіантів. Цього досягають, використовуючи вкладеність умовних операторів, унаслідок чого програма стає складнішою і менш зрозумілішою. Зручним засобом вибору одного варіанта з множини варіантів є оператор вибору (оператор **Select Case**) , який має вигляд:

```
Select Case Перевірка
  [Case Список_порівнянь-N
    [Оператори-N]]
    ...
  [Case Else
    [Оператори-else]]
End Select
```

- Перевірка: деякий текстовий або числовий вираз;
- Список_порівнянь-N має вигляд:

  елемент_порівняння [, елемент_порівняння]…

  де елемент_порівняння є {Вираз | Вираз1 **To** Вираз2
  | **Is** операція_порівняння Вираз}
- Оператори-N: один або кілька операторів, які будуть виконані у тому випадку, якщо вираз перевірка збігається з частиною виразів зі списку_порівнянь-N;
- Оператори-else: один або кілька операторів, які виконаються у випадку, якщо вираз перевірка не збігається з жодним з речень **Case**.

Увага! Вираз **TypeOf**… в операторі **Select Case** використовувати **не можна**.

Якщо значення виразу перевірка збігається зі значеннями виразів списку_порівнянь у декількох реченнях **Case**, то буде виконано тільки перший за порядком набір операторів. Доречно включати речення **Case Else** у блок **Select Case** для опрацювання непередбачених значень виразу перевірка. Припустимі вкладені оператори **Select Case**. Кожне речення **Select Case** разом з тим повинно мати відповідне речення **End Select**. У кож-

ному реченні Case можна використовувати кілька виразів чи діапазонів, наприклад:

```
Case 1 To 4, 7 To 9, 13, Is > MaxNumber
```

Приклад 3.6. Сформувати повідомлення про діапазон величини x:

```
Option Explicit
Public Sub main()
   Dim st As String, Prm As String
   Dim Ttl As String, Def As String
   Dim x As Integer
   Ttl = "Введення": Prm = "Введіть  число!"
   Def = "4"
   st = InputBox(Prm, Ttl, Def): x = Val(st)
   Select Case x
    Case 0 To 3
        st = " 0<= x <=3 "
    Case 4, 5, 6, 7, 8, 9, 10
        st = " 4<= x <=10 "
    Case Else
        st = " x<0 або x>10 "
   End Select
   Debug.Print st
End Sub
```

3.4. Оператори циклу

3.4.1. Оператор циклу з параметром (оператор **For-Next**). Оператор циклу з параметром (оператор For–Next) призначений для виконання деякої послідовності операторів задану кількість разів. Синтаксис:

```
For Лічильник=Початкове_значення To Кінцеве_значення _
                [Step Крок]
    [Оператори_тіла_циклу]
    [Exit For]
    [Оператори_тіла_циклу]
Next [лічильник]
```

Лічильник повинен бути числовою змінною, що не є елементом масиву або запису, якій спочатку присвоюється початкове_значення. Якщо крок має додатне значення, то тіло циклу виконуватиметься доти, доки значення лічильника не перевищить кінцевого_значення. Якщо фразу **Step** не задано, то крок вважають рівним одиниці. Щоб вийти з циклу раніше, ніж змінна циклу досягне заданого значення, необхідно використати оператор **Exit For**.

У тілі циклу **For–Next** можна змінити значення лічильника, однак це ускладнить перевірку та налагодження такого циклу. Зміна кінцевого_значення у тілі циклу не впливає на виконання циклу. Цикли **For–Next** можуть бути вкладеними.

Приклад 3.7. Скласти псевдоконсольне застосування обчислення *n*!

```
Option Explicit
Public Sub main()
   Dim st  As String, Prm As String
   Dim Ttl As String, Def As String
   Dim n As Byte, i As Byte, fact As Long
   Ttl = "Введення": Prm = "Введіть  число!"
   Def = "4"
   st = InputBox(Prm, Ttl, Def): n = Val(st)
   fact = 1
   For i = 2 To n
       fact = fact * i
   Next i
   Debug.Print n; "!="; fact
End Sub
```

Приклад 3.8. Обчислити $S = \sum_{i=1}^{n} (-1)^i \cdot \frac{x}{i} = x \cdot (-1 + \frac{1}{2} - \frac{1}{3} + ... + (-1)^n \cdot \frac{1}{n})$.

Зауважимо, що знак доданків у заданому ряді почергово змінюється, а тому вводимо спеціальну змінну znak, яка міститиме знак чергового доданка. Величини x і n отримаємо з діалогу.

```
Public Sub main()
  Dim n As Byte, i As Byte, znak As Integer
  Dim x As Double, S As Double
  n = InputBox("Введіть число n:", "Введення n")
```

```
  x = InputBox("Введіть число x:", "Введення x")
  S = 0 : znak = 1
  For i = 1 To n
        znak = -znak
        S = S + znak / i
  Next i
  S = S * x
  MsgBox "S=" & S, vbOkOnly, "Результат"
End Sub
```

Увага! Visual Basic автоматично реалізує перетворення значення правої частини оператора присвоєння до значення типу лівої частини (див. у прикладі 3.8 отримання величин `x` і `n` з діалогу `InputBox`).

Оператор **`For Each-Next`** виконує `оператори_тіла_циклу` для кожного елемента у колекції об'єктів чи масиву. Синтаксис:

```
For Each Елемент In Група
    [Оператори_тіла_циклу]
[Exit For]
    [Оператори_тіла_циклу]
Next [Елемент]
```

`Елемент` – це змінна, що представляє об'єкти колекції чи елементи масиву, а `група` – це назва колекції чи масиву. Цикли **`For Each -Next`** можуть бути вкладеними, проте назви змінних `елементів` у цьому випадку не можуть збігатися.

Приклад 3.9. Закриття усіх відкритих робочих книг:

```
Dim Book As Object
For Each Book In Workbooks
       Book.Close
Next Book
…
```

3.4.2. Оператори циклу з умовою. Оператор циклу **`Do-Loop`** повторює `оператори_тіла_циклу` доти, доки задана `умова` має значення **`True`** (варіант **`While`**) або доки вона не набуде цього значення (варіант **`Until`**).

Синтаксис:

```
Do [ { While | Until } Умова]
    [Оператори_тіла_циклу]
    [Exit Do]
    [Оператори_тіла_циклу]
Loop
```

або

```
Do
    [Оператори_тіла_циклу]
    [Exit Do]
    [Оператори_тіла_циклу]
Loop [ {While | Until } Умова]
```

Під час використання **While** виконання тіла циклу продовжується доти, доки задана умова має значення **True**. При використанні **Until** виконання тіла циклу припиняється, як тільки умова набуде значення **True**.

Якщо **While** чи **Until** знаходяться у реченні **Do**, то умову перевіряють *до виконання* тіла циклу. Якщо ж **While** чи **Until** фігурують у реченні **Loop**, то умова перевіряється *після виконання* тіла циклу. З циклу можна вийти у будь-якому місці за допомогою оператора **Exit Do**, не чекаючи перевірки умови на початку чи наприкінці циклу.

Оператор циклу **While-Wend** повторює оператори_тіла_циклу доти, доки задана умова має значення **True**. Синтаксис:

```
While Умова
   [Оператори_тіла_циклу]
Wend
```

Приклад 3.10. Послідовно вводять дійсні числа. Ознакою завершення уведення даних є введення нуля. Обчислити різницю між найбільшим і найменшим числами:

```
Public Sub main()
    Dim i As Byte, x As Double
    Dim max As Double, min As Double
    MsgBox " Вводьте числа. Нуль - кінець введення!"
    x = InputBox("Введіть число", "Введення")
```

```
    If x = 0 Then
        MsgBox "А нічого не введено!"
        Exit Sub
    End If
    max = x: min = x
    Do
        If x > max Then max = x
        If x < min Then min = x
        x = InputBox("Введіть число", "Введення")
    Loop Until x = 0
    x = max - min
    MsgBox "Різниця=" & x, vbOKOnly, "Результат"
End Sub
```

Приклад 3.11. Обчислити $s = 2^2 + 4^2 + ... + (2 \cdot n)^2$, де $n > 5$. У програмі контролюватимемо дотримання вимоги щодо величини *n*:

```
Public Sub main()
  Dim i As Byte, n As Byte
  Dim s As Long
  Do
    MsgBox " Вводьте n>5"
    n = InputBox("Введіть число", "Введення")
  Loop Until n > 5
  s = 0
  For i = 2 To 2 * n Step 2
      s = s + i * i
  Next i
  MsgBox "s=" & s, vbOKOnly, "Результат"
End Sub
```

На місці операторів_тіла_циклу у циклі може стояти інший цикл (отримуємо *вкладені цикли*). В усіх циклах усередині операторів_тіла_циклу можна записати оператор **Exit Do**. Якщо під час виконання циклу цей оператор отримує керування, то відбувається негайний вихід з циклу і перехід на наступний оператор, записаний за зазначеним циклом.

Приклади використання циклів для розв'язання деяких типових задач наведено у §§ 3.5 – 3.7.

3.5. Програми з простим повторенням

У цьому параграфі розглянемо задачі, для розв'язання яких треба виконати перебір певної послідовності значень, причому кожне значення використовується тільки один раз.

Для перебору значень використовуватимемо відповідний оператор циклу. Оскільки конкретне значення послідовності використовується тільки один раз на певній ітерації циклу, то заводити масиви для таких задач недоречно.

Приклад 3.12. Ввести 10 цілих чисел і полічити серед них кількість додатних:

```
Public Sub main()
  Dim i As Byte, k As Byte, n As Integer
  k = 0
  For i = 1 To 10
      n = InputBox("Введіть число", "Введення")
      If n > 0 Then k = k + 1
  Next i
  MsgBox "Додатних:" & k, vbOKOnly, "Результат"
End Sub
```

Приклад 3.13. Вивести підряд усі букви від "Z" до "A":

```
Public Sub main()
  Dim i As Integer, s As String
  s = ""
  For i = Asc("Z") To Asc("A") Step -1
      s = s & Chr(i)
  Next i
  MsgBox s, vbOKOnly, "Результат"
End Sub
```

Приклад 3.14. Пари додатних раціональних чисел вводять з клавіатури. Обчислити добуток кожної пари і суму всіх чисел:

```
Public Sub main()
  Dim x As Double, y As Double
  Dim sum As Double, dob As Double
  Dim s As String
  sum = 0
```

```
  Do
    x = InputBox("Введіть число", "Введення")
    y = InputBox("Введіть число", "Введення")
    If x > 0 And y > 0 Then
       dob = x * y
       Debug.Print "dob ="; dob
       sum = sum + x + y
    End If
    s=MsgBox("Продовжувати?", vbYesNo, "Продовження")
  Loop While s = vbYes
  Debug.Print "sum ="; sum
End Sub
```

Приклад 3.15. Протабулювати значення функції $y = \sqrt{x^2 + 1} \cdot \cos x$ на проміжку $[a; b]$ з кроком h.

```
Public Sub main()
  Dim a As Double, b As Double
  Dim h As Double, x As Double, y As Double
  Do
    a = InputBox("Введіть a", "Введення")
    b = InputBox("Введіть b (b>a)", "Введення")
    h = InputBox("Введіть h (h>0)", "Введення")
  Loop Until b > a And h > 0
  Debug.Print " ---------------------------------- "
  Debug.Print " |   x          |      y           |"
  Debug.Print " ---------------------------------- "
  For x = a To b Step h
    y = Sqr(x * x + 1) * Cos(x)
    Debug.Print x, y
  Next x
  Debug.Print " ---------------------------------- "
End Sub
```

3.6. Програмування задач цілочислової арифметики

Приклад 3.16. Обчислити N-е число Фіббоначчі. Числа Фіббоначчі визначають за рекурентною формулою:

F(0)=F(1)=1; F(i)=F(i-1)+F(i-2), якщо i≥2.

```
Public Sub main()
  Dim a As Integer, b As Integer
```

```
  Dim z As Integer, i As Integer, n As Integer
  Do
    n = InputBox("Введіть номер числа>=0","Введення")
  Loop Until n >= 0
  a = 1 ' a=F(0), a відповідає за F(i-2)
  b = 1 ' b=F(1), b відповідає за F(i-1)
  For i = 2 To n
      z = a + b       ' z відповідає за F(i)
      a = b: b = z    ' наступна пара чисел
  Next i
  Debug.Print n; "-е число Фіббоначчі= "; b
End Sub
```

Приклад 3.17. Дано натуральне число. Знайти кількість і суму десяткових цифр у записі цього числа.

```
Public Sub main()
  Dim a As Long, s As Byte, k As Byte
  Do
   a=InputBox("Введіть натуральне число", "Введення")
  Loop Until a >= 1
  k = 0
  While a > 0
   s = s + a Mod 10
   k = k + 1
   a = a \ 10
  Wend
  Debug.Print "Кількість цифр= "; k
  Debug.Print "Сума цифр= "; s
End Sub
```

Для отримання цифр числа використаємо операцію отримання залишку (**Mod**) від цілочисельного ділення числа на 10. Під час першого ділення отримаємо цифру з розряду одиниць (наприклад, `167 Mod 10=7`). Операція цілочисельного ділення вилучає цю цифру з розряду одиниць (`167\10=16`), а наступна операція отримання залишку поверне число десятків (`16 Mod 10=6`) і т. д. Зауважимо, що при цьому отримуємо цифри числа у зворотному порядку. Цей факт можна використати при розв'язанні відомої задачі про визначення *паліндрома* – числа, величина якого не змінюється, якщо порядок цифр у його записі змінити на протилежний.

Приклад 3.18. Дано натуральне число. Перевірити, чи воно є паліндромом.

```
Public Sub main()
  Dim a As Long, b As Long, s As Long
  Do
    a = InputBox("Введіть натуральне число", "Введення")
  Loop Until a >= 1
  b = a ' копія числа
  While a > 0
   s = s * 10 + a Mod 10
   a = a \ 10
  Wend
  If b = s Then
    MsgBox "Паліндром "
  Else
    MsgBox "Не паліндром "
  End If
End Sub
```

Подібний алгоритм використовують і при переведенні цілого десяткового числа в іншу систему числення: обчислюємо остачу та частку від ділення числа на нову основу, частку ділимо знову на нову основу – отримуємо наступні остачу та частку і так далі, доки чергова частка дорівнюватиме нулю. Остання частка й отримані остачі у зворотному порядці формують число за новою основою.

Приклад 3.19. Дано десяткове натуральне число. Отримати його запис у вісімковій системі числення.

```
Public Sub main()
  Dim a As Long, b As Long, s As Long
  Do  : a = InputBox("Введіть натуральне число")
  Loop Until a >= 1
  b = 1 ' b - степінь 10; спочатку 10^0=1
  s = 0 ' s - результат; 8-ове число
  While a > 0
     s = s + (a Mod 8) * b
     a = a \ 8 :  b = b * 10
  Wend
  MsgBox "8-ове число: " & s
End Sub
```

Приклад 3.20. Дано натуральне число, більше одиниці. Розкласти його на прості множники.

```
Public Sub main()
  Dim a As Long, k As Integer
  Dim s As String
  Do
    a = InputBox("Введіть натуральне число", "Введення")
  Loop Until a >= 1
  k = 2 ' вилучаємо множники кратні 2; потім 3 і т.д.
  s = a & "="
  While a > 1
    If a Mod k = 0 Then
       s = s & k & "*"
       a = a \ k
    Else
       k = k + 1
    End If
  Wend
  MsgBox s & "1"
End Sub
```

3.7. Сумування рядів

Приклад 3.21. Написати програму обчислення функції $\mathrm{Ch}\,x$ (косинус гіперболічний) за допомогою ряду Тейлора з заданою точністю $\varepsilon > 0$ за формулою: $y = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \frac{x^{2n}}{(2n)!} + \ldots$

Цей ряд є збіжним при $|x| < \infty$. Для збіжного ряду модуль загального члена $u_n = \frac{x^{2n}}{(2n)!}$ при збільшенні n прямує до нуля. Для досягнення необхідної точності треба сумувати члени ряду, абсолютна величина яких більша за ε. Для деякого n нерівність $|u_n| \geq \varepsilon$ не виконується, обчислення припиняють.

Пряме обчислення ряду (визначення степенів і факторіала) є недоцільним: надто швидко отримуються великі числа, унаслідок ділення яких втрачають точність. Для обчислення необхідно ско-

ристатися рекурентною формулою $u_{n+1} = u_n \times T,$ де T – деякий множник. Визначимо множник T:

$$T = \frac{u_{n+1}}{u_n} = \frac{x^{2(n+1)}}{(2(n+1))!} \cdot \frac{(2n)!}{x^{2n}} = \frac{x^2}{(2n+1)(2n+2)}.$$

Наперед передбачити, скільки членів ряду треба просумувати, неможливо. У циклах такого роду з різних причин можливе зациклення. Тому для надійності у таких програмах треба задавати величину максимальної кількості ітерації, яка слугуватиме для аварійного виходу з циклу.

```
Public Sub main()
  Dim x As Single, y As Single, u As Single
  Dim n As Integer
  Const MaxITER As Integer = 500 ' Максимум ітерацій
  Const eps As Single = 0.000001 ' точність обчислень
  x = InputBox("Введіть аргумент", "Введення")
  u = 1: y = u ' Перший член ряду і поч. знач. суми
  n = 0
  While Abs(u) > eps And n < MaxITER
    u = u * x * x / ((2 * n + 1) * (2 * n + 2))
    y = y + u: n = n + 1
  Wend
  If n < MaxITER Then
    MsgBox "y= " & y
  Else
    MsgBox "Не збігається "
  End If
End Sub
```

? Запитання для самоперевірки

1. Перелічіть оператори керування.
2. Що таке оператор визначення?
3. Які форми має умовний оператор?
4. Для розв'язання яких задач доцільно використовувати оператор вибору?
5. Запишіть і охарактеризуйте оператор циклу **For – Next**.
6. Запишіть і охарактеризуйте оператор циклу **Do – Loop**.
7. Запишіть і охарактеризуйте оператор циклу **While – Wend**.

8. У яких випадках застосовують оператор **`Exit Do`**?
9. Запишіть і охарактеризуйте оператор **`Goto`**.

Завдання для програмування

Завдання 3.1. Скласти програми розв'язання таких задач:

1. Прочитати з клавіатури групу з 12-ти чисел. Знайти серед них найменше та номер його першого входження у групу.
2. Прочитати з клавіатури групу з 10-ти чисел. Знайти серед них найбільше та номер його останнього входження у групу.
3. Прочитати з клавіатури 14 чисел і обчислити суму всіх невід'ємних значень.
4. Серед 15-ти заданих чисел визначити кількість додатних, від'ємних і нульових.
5. Ввести з клавіатури групу чисел. Перше число **N**>0 означає розмір наступної (за ним) групи чисел. Обчислити суму чисел, що належать проміжку [–3; 5].
6. Ввести з клавіатури групу чисел. Перше число **N**>0 означає розмір наступної (за ним) групи чисел. Визначити відсоток додатних чисел.
7. Прочитати з клавіатури групу з 15-ти чисел. Знайти середнє арифметичне чисел, розташованих за порядком за першим ненульовим. Вважати, що обов'язково є хоча б одне ненульове значення, причому не на останньому місці.
8. Прочитати з клавіатури групу з 15-ти чисел. Знайти середнє арифметичне чисел, розташованих за порядком за першим ненульовим. Вважати, що ненульових значень може не бути взагалі.
9. Прочитати з клавіатури групу з 12-ти чисел. Знайти різницю між найбільшим і найменшим числами.
10. Обчислити суму добутків таких пар чисел: 4·5, 5·6, ..., 21·22.
11. Прочитати з клавіатури групу з 15-ти чисел. Знайти перше за порядком число, що дорівнює нулю, визначивши його позицію.
12. Прочитати з клавіатури групу з 15-ти чисел. Знайти порядковий номер першого ненульового числа.
13. Прочитати з клавіатури групу з 15-ти чисел. Знайти порядковий номер останнього ненульового числа.
14. Прочитати з клавіатури групу, яка складається з-понад трьох чисел. Третє за порядком число означає кількість чисел у групі, у тім числі дане. Обчислити суму всіх чисел.
15. Прочитати з клавіатури групу з 10-ти чисел. Знайти суму модулів тих чисел, які передують першому нульовому, або усіх чисел групи, якщо нульових немає.

Завдання 3.2. Протабулювати значення функції $y = \mathrm{f}(x)$ на проміжку $[a; b]$ у n точках. Функція вибирається з переліку функцій завдання 2.1 (див. с. 40) за номером студента у списку студентів підгрупи.

Завдання 3.3. Скласти програми розв'язання таких задач:

1. Дано натуральне число. Знайти кількість і добуток десяткових цифр у записі цього числа.
2. Дано натуральне число. Знайти середнє арифметичне цифр у записі цього числа.
3. Дано натуральне число. Знайти першу за порядком позицію (зліва направо), у якій знаходиться найменша цифра у записі цього числа. Вивести також і цю найменшу цифру.
4. Дано натуральне число. Знайти останню за порядком позицію (зліва направо), у якій знаходиться найменша цифра у записі цього числа. Вивести також і цю найменшу цифру.
5. Дано натуральне число. Знайти першу за порядком позицію (зліва направо), у якій знаходиться найбільша цифра у записі цього числа. Вивести також і цю найбільшу цифру.
6. Дано натуральне число. Знайти останню за порядком позицію (зліва направо), у якій знаходиться найбільша цифра у записі цього числа. Вивести також і цю найбільшу цифру.
7. Знайти найменше спільне кратне двох натуральних чисел.
8. Дано десяткове натуральне число. Отримати його запис у сімковій системі числення.
9. Дано вісімкове натуральне число. Отримати його запис у десятковій системі числення.
10. На проміжку $[a; b]$, де a і b – натуральні числа, вивести всі трійки Піфагорових чисел (тобто $k^2 + l^2 = m^2$).
11. Дано десяткове натуральне число. Отримати його запис у трійковій системі числення.
12. Дано четвіркове натуральне число. Отримати його запис у десятковій системі числення.
13. Обчислити суму часток і суму остач таких виразів: $(5p^2+3)/(3p)$, p=1,...,10.
14. Обчислити середнє арифметичне остач таких виразів: $(4p^2+2)/(2p)$, p=1,...,12.
15. Дано натуральне число $m > 1$. Знайти найбільше натуральне k, при якому $2^k < m$.

Завдання 3.4. Протабулювати значення функції $y = \mathrm{f}(x)$ на проміжку $[a; b]$ у n точках. Обчислення функції реалізувати за допомогою розкладу у ряд Тейлора. Обчислені значення порівнювати зі значеннями відповідних бібліотечних функцій. Функцію з поданого переліку обирати за номером студента у списку студентів підгрупи:

1. $y = \ln\dfrac{x+1}{x-1} = 2\sum\limits_{n=0}^{\infty}\dfrac{1}{(2n+1)x^{2n+1}};\ |x| > 1.$

2. $y = \mathrm{e}^{-x} = \sum\limits_{n=0}^{\infty}\dfrac{(-1)^n \cdot x^n}{n!};\ |x| < \infty.$ 3. $y = \ln(1-x) = -\sum\limits_{n=0}^{\infty}\dfrac{x^n}{n};\ -1 \le x < 1.$

4. $y = \ln(1-x) = -\sum\limits_{n=0}^{\infty}\dfrac{x^n}{n};\ -1 \le x < 1.$

5. $y = \ln\dfrac{x+1}{1-x} = 2\sum\limits_{n=0}^{\infty}\dfrac{x^{2n+1}}{2n+1};\ |x| < 1.$

6. $y = \operatorname{arctg} x = \dfrac{\pi}{2} + \sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1}}{(2n+1)x^{2n+1}};\ |x| > 1.$

7. $y = \operatorname{arc\,ctg} x = \dfrac{\pi}{2} + \sum\limits_{n=0}^{\infty}\dfrac{(-1)^{n+1} \cdot x^{2n+1}}{2n+1};\ |x| \le 1.$

8. $y = \operatorname{arctg} x = \sum\limits_{n=0}^{\infty}\dfrac{(-1)^n \cdot x^{2n+1}}{2n+1};\ |x| \le 1.$ 9. $y = \cos x = \sum\limits_{n=0}^{\infty}\dfrac{(-1)^n \cdot x^{2n}}{(2n)!};\ |x| < \infty.$

10. $y = \mathrm{e}^{x} = \sum\limits_{n=0}^{\infty}\dfrac{x^n}{n!};\ |x| < \infty.$ 11. $y = \dfrac{\sin x}{x} = \sum\limits_{n=0}^{\infty}\dfrac{(-1)^n \cdot x^{2n}}{(2n+1)!};\ |x| < \infty.$

12. $y = \ln(1-x) = -\sum\limits_{n=0}^{\infty}\dfrac{x^n}{n};\ -1 \le x < 1.$ 13. $y = \mathrm{e}^{-x} = \sum\limits_{n=0}^{\infty}\dfrac{(-1)^n \cdot x^n}{n!};\ |x| < \infty.$

14. $y = \ln\dfrac{x+1}{1-x} = 2\sum\limits_{n=0}^{\infty}\dfrac{x^{2n+1}}{2n+1};\ |x| < 1.$ 15. $y = \mathrm{e}^{x} = \sum\limits_{n=0}^{\infty}\dfrac{x^n}{n!};\ |x| < \infty.$

4. Структуровані типи даних

План викладу матеріалу:

1. Масиви.
2. Одновимірні масиви (вектори).
3. Алгоритми пошуку в одновимірному масиві.
4. Алгоритми сортування в одновимірному масиві.
5. Матричні задачі.
6. Записи (типи, визначені програмістом).

Ключові терміни розділу

- *Означення масиву*
- *Розмірність масиву*
- *Стандартні задачі на векторах*
- *Двійковий пошук у векторі*
- *Головні матричні операції*
- *Оголошення статичного масиву*
- *Визначення динамічного масиву*
- *Лінійний пошук у векторі*
- *Методи сортування вектора*
- *Записи*

4.1. Масиви

4.1.1. Статичні масиви. *Масив* – це впорядкована структура *однотипних* даних, яка зберігає ці дані у пам'яті комп'ютера послідовно. Доступ до елемента масиву здійснюється через його *індекси*. Розрізняють *статичні* і *динамічні* масиви. Межі статичного масиву встановлюють на етапі розробки програми, вони можуть змінюватися тільки у новій версії програми. Динамічні масиви можуть змінювати свої межі під час виконання програми. Оголошення одновимірного статичного масиву зазвичай має вигляд:

```
{Static | Public | Dim} назва_масиву (верхня_межа) [As тип]
```

У Visual Basic індексування за домовленістю розпочинається з нуля, тобто індекс 0 позначає перший елемент масиву, індекс 1 – другий і т. д. `Верхня_межа` позначає максимальний індекс і може бути тільки *константою*.

Приклад 4.1. Оголошення одновимірного статичного масиву:

```
Dim aName(150) As String
```

У цьому прикладі оголошено масив `aName`, що налічує 151 елемент типу **`String`**: `aName(0)`, `aName(1)`, ..., `aName(150)`, `aName(151)`.

Те, що говорилося про час життя й області визначення змінних, стосується здебільшого і масивів. Однак *статичні* масиви не можна визначити локально усередині процедури, а тільки глобально чи на рівні контейнера. Оператор **Option Base** дає змогу задати індексацію масиву з одиниці: **Option Base** 1

Необхідно щоб, цей оператор перебував у секції *General-Declarations* контейнера (форми, модуля, класу). Припустимими значеннями для **Option Base** є тільки 0 і 1. З метою встановлення інших меж масиву необхідно використати такий синтаксис:

```
{Static | Public | Dim} назва_масиву _
        ([нижня_межа To] верхня_межа)[As тип]
```

З визначенням верхньої і нижньої межі можна задати будь-який діапазон індексу. Це зручно, якщо індекс несе певне навантаження (дата, номер, вік тощо).

Увага! Часто, для зручності, елемент одновимірного масиву з нульовим індексом ігнорують (див. приклад 4.2). У таких випадках програма не залежатиме від **Option Base**.

Visual Basic дає змогу також створювати багатовимірні масиви. При оголошенні багатовимірного масиву описи кожної розмірності розділяються комами.

Приклад 4.2. Оголошення багатовимірного статичного масиву:

```
Dim aName(10, 25) As String '11*26=286 різних значень
```

Приклад 4.3. Оголошення багатовимірного статичного масиву типу **Variant** з зазначенням діапазонів індексів:

```
Dim aAr(80 To 120, 40 To 45, 1 To 256, 3, 1997 To 2050)
```

Кількість діапазонів в описі масиву (кількість індексів) визначає *розмірність* масиву. Одновимірні масиви, за аналогією з термінами лінійної алгебри, називають *векторами*, а двовимірні – *матрицями* (перший індекс задає номер рядка, а другий – номер стовпця). Розмірність при визначенні масивів може становити 60 (на практиці – не більше трьох).

Приклад 4.4. Ввести послідовність дійсних чисел (від 4-х до 20-ти чисел) і замінити перші три члени послідовності середнім арифметичним чисел цієї послідовності. У програмі послідовність дійсних чисел зберігати у статичному одновимірному масиві.

```
Public Sub main()
  Dim mas(20) As Single, sa As Single
  Dim n As Byte, i As Byte
  Do
   n =InputBox("Введіть n > 3 і n < 21", "Введення")
  Loop Until n > 3 And n < 21
  sa = 0
  For i = 1 To n
   mas(i)=InputBox("Введіть число!", "Введення")
   sa = sa + mas(i)
  Next i
  Debug.Print "Початковий масив:";
  For i = 1 To n
   Debug.Print mas(i);" "; ' виводимо масив в один рядок
  Next i
  Debug.Print " " ' перехід на новий рядок
  sa = sa / (n + 1) ' середнє арифметичне
  For i = 1 To 3 ' заміна елементів
   mas(i) = sa
  Next i
  Debug.Print "Кінцевий масив:";
  For i = 1 To n
   Debug.Print mas(i);" ";
  Next i
  Debug.Print " "
End Sub
```

4.1.2. **Динамічні масиви**. Іноді при оголошенні масиву його розмір і/або розмірність наперед невідомі (наприклад, визначаються користувачем у діалозі). У цьому випадку варто створити динамічний масив, що дає змогу змінювати його розмір чи розмірність під час виконання програми. Динамічний масив визначається за два кроки. Спочатку масив описують у секції *General-Declarations* контейнера (форми, модуля, класу) без зазначення розмірності й верхніх меж, наприклад:

```
Dim aDynArray() As Single
```

Потім за допомогою оператора **ReDim** у деякій процедурі установлюють його фактичний розмір і/або розмірність.

Синтаксис оператора **ReDim**:

```
ReDim [Preserve] назва_змінної (межі) [As тип_даних]
```

Оператор **ReDim** використовують тільки у процедурах. Щодо цього тип даних зазначати не обов'язково, зокрема, якщо він уже визначений оператором **Dim**. Однак не можна оголосити масив з даними одного типу, а потім використовувати **ReDim** для приведення масиву до іншого типу, за винятком випадку, коли масив має тип **Variant**. У цьому випадку приведення масиву до явного типу даних припустимо. Отже, розмірність масиву можна за необхідності змінити. Проте тоді виникає небезпека втратити значення його елементів, тому що після зміни розмірності елементам масиву присвоюються значення за домовленістю.

Однак Visual Basic дає змогу змінювати розмірність масиву без втрати значень. З цією метою використовують **ReDim** разом із ключовим словом **Preserve**. Однак зауважимо, що у багатовимірних масивах можна змінювати *тільки останній* вимір.

Приклад 4.5. Використання **ReDim** для багатовимірних масивів:

```
Dim aArray () As Variant ' General-Declarations
Private Sub Cornmandl_Click()
ReDim aArray (10, 10)
…
ReDim Preserve aArray (10, 15) ' Все добре!
…
ReDim Preserve aArray (15, 15) ' Помилка!
End Sub
```

Область дії динамічних масивів визначається способом їхнього оголошення – якщо за допомогою оператора **Public**, то масив буде глобальним, якщо за допомогою **Dim**, то – контейнерним.

Приклад 4.6. Переробити програму прикладу 4.4, увівши одновимірний динамічний масив:

```
Option Explicit
Public mas() As Single ' General-Declarations)
Public Sub main()
  Dim sa As Single, mas(20)
...
End Sub
```

Visual Basic 6.0 дає змогу присвоїти значення одного масиву іншому так само, як це має місце для змінних. Наприклад:

```
Sub ByteCopy(oldCopy() As Byte, newCopy() As Byte)
   newCopy = oldCopy
End Sub
```

При присвоєнні масивів необхідно, щоб розмірності масивів і кількість їхніх елементів були однаковими.

4.2. Одновимірні масиви (вектори)

Над одновимірним масивом (*вектором*) виконують дві базові операції: *отримання* деякого елемента з масиву і *розміщення* елемента у масиві (порівняння і присвоювання). Visual Basic не має засобів введення/виведення усіх елементів масиву відразу, тому введення і виведення значень здійснюють поелементно. *Ініціалізують* масиви через введення даних і присвоювання.

Іноді в тексті навчальної програми уведення масиву займає більше місця, ніж запис алгоритму рішення, що утруднює сприйняття програми. Тому в прикладах цього параграфа ми здебільшого опускатимемо ініціалізацію масиву, а користуватимемося тільки математичним формулюванням задачі типу: "Дано натуральне число n і дійсні числа a_1, a_2,..., a_n. Знайти …".

Зупинимося докладніше на введенні та використанні у програмі числа n і чисел a_1, a_2,..., a_n. У Visual Basic кількість *елементів масиву, їхню розмірність і тип* необхідно *явно* задати до моменту використання масиву в програмі. Якщо кількість чисел (n) визначається у діалозі з користувачем, то для зберігання чисел a_1, a_2,..., a_n використовують *динамічний* одновимірний масив з верхньою межею n, ігноруючи при цьому елемент масиву з нульовим індексом.

Існує чимало розмаїтих задач на одновимірні масиви. Як і всі задачі загалом, умовно їх можна розділити на три види:

1) прості задачі, які розв'язують в "одне розуміння";
2) стандартні задачі;
3) задачі, рішення яких потребує знання допоміжних алгоритмів, спеціальних методів і прийомів.

Очевидно, що без уміння вирішувати задачі перших двох видів неможливо вирішувати нестандартні задачі. Розглянемо кілька прикладів задач першого виду.

Приклад 4.7. У масиві *a* кожен елемент дорівнює 0 чи 1. Замінити усі нулі одиницями, і навпаки.

Досить одного оператора присвоювання у тілі циклу:

```
For i= 1 To n
    a(i)=1-a(i)
Next i
```

Приклад 4.8. У масиві кожен елемент дорівнює 0, 1 чи 2. Переставити елементи масиву так, щоб послідовно розташувати усі нулі, потім усі одиниці та усі двійки. Додаткового масиву не заводити.

Можна не переставляти елементи масиву, а обчислити кількості 0, 1 і заповнити масив заново необхідним способом:

```
k0=0 : k1=0
For i= 1 To n
   If a(i) = 0 Then ' кількість 0
                    k0=k0+1
   ElseIf a(i) = 1 Then ' кількість 1
                    k1=k1+1
   End If
Next i
For i= 1 To n ' заповнення всього масиву 2
   a(i)=2
Next i
For i= 1 To k0 ' початкове заповнення 0
   a(i)=0
Next i
For i= k0+1 To k0+k1 ' заповнення 1
   a(i)=1
Next i
```

Перелічимо деякі *стандартні* задачі на одновимірні масиви:

1) визначення найбільшого/найменшого елемента;
2) визначення суми/добутку елементів (безумовне та умовне);
3) підрахунок/заміна елементів, що відповідають заданій умові;

4) визначення заданого порядку розміщення елементів;
5) видалення елемента, введення елемента в задану позицію;
6) перерозташування (інвертування, циклічне зсування) елементів;
7) випадкова вибірка елементів (з повтореннями і без повторень);
8) злиття двох упорядкованих масивів в один;
9) пошук заданого елемента:
   а) у неупорядкованому масиві; б) в упорядкованому масиві;
10) сортуваня масиву (прості методи).

Задачі 1 – 8 розглянемо в прикладах цього параграфа. Задачам 9 і 10 присвятимо окремо наступні два параграфа. До складних задач можна зачислити поліпшені методи сортування масивів, пошук моди і медіани масиву; алгоритми генерування комбінаторних об'єктів, алгоритми на графах і т. д.

До стандартних задач зачислимо і задачу *обчислення значення многочлена* у деякій точці x, оскільки коефіцієнти многочлена зазвичай зберігають в одновимірному масиві.

Приклад 4.9. Обчислення значення многочлена степеня `n` у точці x, коефіцієнти якого перебувають в масиві A, за схемою Горнера:

$$P_n(x) = A_0 \cdot x^n + A_1 \cdot x^{n-1} + \ldots + A_{n-1} \cdot x + A_n =$$
$$= (\ldots(((0 \cdot x + A_0) \cdot x + A_1) \cdot x + A_2) \cdot x + \ldots + A_{n-1}) \cdot x + A_n.$$

```
…
p = 0
For i = 0 To n
        p = p * x + a(i)
Next i
…
```

Приклад 4.10. Дано натуральне число `n` і послідовність дійсних чисел $a_1, a_2, \ldots, a_n$. Знайти перше від початку послідовності найменше число a_i (`i`=1..`n`) і останнє за порядком у послідовності найбільше число a_j (`j`=1..`n`) і поміняти їх місцями.

Для розв'язання задачі достатньо визначити *порядкові номери* першого від початку найменшого числа і останнього за порядком найбільшого числа послідовності.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim x As Single
 Dim n As Byte, i As Byte, kmin As Byte, kmax As Byte
 Do
  n = InputBox("Введіть n <= 20", "Введення")
 Loop Until n <= 20
 ReDim a(n)
 For i = 1 To n
  a(i) = InputBox("Введіть " & i & " число!")
 Next i
 Debug.Print "Початковий масив: ";
 For i = 1 To n
        Debug.Print a(i); " ";
 Next i
 Debug.Print " "
 kmin = 1  :  kmax = 1
 For i = 2 To n
     If a(kmin) > a(i)  Then kmin = i
     If a(kmax) <= a(i) Then kmax = i
 Next i
 x = a(kmin): a(kmin) = a(kmax): a(kmax) = x
 Debug.Print "Кінцевий масив: ";
 For i = 1 To n
        Debug.Print a(i); " ";
 Next i
 Debug.Print " "
End Sub
```

Приклад 4.11. Дано натуральне число n і послідовність дійсних чисел a_1, a_2,..., a_n. Знайти суму додатних чисел і середнє арифметичне від'ємних чисел. Значення суми присвоїти першому елементові масиву, а середнє арифметичне – останньому елементові масиву.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim sd As Single, sv As Single
 Dim n As Byte, i As Byte, nv As Byte, nd As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
  sd = 0 : sv = 0 : nv = 0 : nd = 0
 For i=1 To n
   a(i) = InputBox("Введіть " & i & " число!")
   If a(i)>0 Then sd = sd+a(i) : nd = nd+1
   If a(i)<0 Then sv = sv+a(i) : nv = nv+1
 Next i
 If nd=0 Then MsgBox "Нема додатних"  Else a(1) = sd
 If nv=0 Then MsgBox "Нема від'ємних" Else a(n) = sv/nv
  ' Виведення масиву a - див. приклад 4.10!
End Sub
```

Приклад 4.12. Дано натуральне число n і послідовність натуральних чисел a_1, a_2,..., a_n. Замінити всі члени послідовності a_1, a_2,..., a_n, більші від 100, числом 101. Визначити кількість таких замін.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim n As Byte, i As Byte, n100 As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
 n100 = 0
 For i=1 To n
  a(i) = InputBox("Введіть " & i & " число!")
  If a(i)>100 Then a(i) = 101 : n100 = n100 + 1
 Next i
 MsgBox "Зроблено " & Str(n100) & " замін!"
 ' Виведення масиву a - див. приклад 4.10!
End Sub
```

Приклад 4.13. Дано натуральне число n і послідовність цілих чисел a_1, a_2,..., a_n. Визначити, скільки разів трапляється у цій послідовності найменше число. Замінити цим найменшим числом відповідну кількість елементів від початку масиву.

```
Option Explicit
Dim a() As Integer
Public Sub main()
 Dim min As Integer
 Dim nmin As Byte, i As Byte, n As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
 ' Введення послідовності чисел - див. приклад 4.10!
 min = a(1) : nmin = 1
 For i=2 To n
   If a(i) < min Then
        min = a(i) : nmin = 1
   ElseIf a(i)=min Then
        nmin = nmin + 1
   End If
 Next i
 For i=1 To nmin
        a(i) = min
 Next i
 ' Виведення масиву a - див. приклад 4.10!
End Sub
```

На перший погляд здається, що для визначення кількості однакових найменших значень необхідно організувати два цикли:

- перший – для визначення найменшого значення (мінімуму);
- другий – для визначення кількості повторень цього мінімуму.

Однак, скориставшись спеціальним прийомом *скидання накопиченого значення*, можна обійтися одним циклом. Якщо умова `a(i) < min` справджується, то знайдено *нове* найменше число і відлік кількості повторень найменшого числа необхідно починати з одиниці (*скидання накопиченого значення* кількості повторень).

Якщо ж умова `a(i) < min` не справджується, то з двох можливих значень (`a(i) = min` чи `a(i) > min`) нас цікавить одне: `a(i) = min` – означає, що знайдено елемент, який збігається з *поточним* найменшим числом (збільшуємо лічильник кількості повторень).

Скидання накопиченого значення – досить поширений прийом, який використовують при розв'язуванні багатьох задач. Скористаємося ним і при розв'язанні наступного прикладу.

Приклад 4.14. Дано натуральне число `n` і послідовність цілих чисел a_1, a_2,..., a_n. Визначити найбільшу кількість однакових чисел, розташованих поруч.

```
Option Explicit
Dim a() As Integer
Public Sub main()
 Dim i As Byte, n As Byte, q As Byte, k As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
 ' Введення послідовності чисел - див. приклад 4.10!
 ' Послідовність одинакових чисел назвемо серією
 k=1 ' Кількість елементів поточної серії
 q=1 ' Шукана величина
 For i=2 To n
  If a(i)=a(i-1) Then
     k=k+1 ' Продовження/початок серії
  Else      ' Кінець серії
       If q<k Then q=k ' Формування max
       k=1   '  Нова ініціалізація
  End If
 Next i
 If q<k Then q = k ' Врахування останньої серії
  MsgBox "Максимум повторів= " & Str(q)
End Sub
```

Приклад 4.15. Дано натуральне число `n` і послідовність цілих чисел a_1, a_2,..., a_n. Визначити, чи є у послідовності розташовані поруч два однакових числа, що дорівнюють 12.

```
Option Explicit
Dim a() As Integer
Public Sub main()
  Dim n As Byte, i As Byte
  Dim f As Boolean
  ' Введення n - див. приклад 4.10!
  ReDim a(n)
  ' Введення послідовності чисел - див. приклад 4.10!
  i = 0  :  f = False
  Do
    i = i + 1
    If a(i) = 12 Then i = i + 1: f = a(i) = 12
  Loop Until f Or i >= n - 1
  If f Then MsgBox "Так " Else MsgBox "Ні "
End Sub
```

Приклад 4.16. Включення нового елемента у задану позицію k.
При включенні нового елемента (рис. 4.1) необхідно попередньо розсунути масив, тобто пересунути підмасив a_k,..., a_n вправо на одну позицію. Переміщення підмасиву необхідно виконувати з кінця і перевіряти, чи достатньо пам'яті виділено під масив.

6	9	5	1	12	18	24		
6	9	5	1	1000	12	18	24	

Рис. 4.1. Включення елемента (1000) у п'яту позицію

```
Option Explicit
Const nmax = 20
Public Sub main()
  Dim a(1 To nmax) As Single, x As Single
  Dim n As Byte, k As Byte, i As Integer
  Do
    n = InputBox("Введіть n <" & nmax)
  Loop Until n < nmax And n >= 1
  For i = 1 To n
    a(i) = InputBox("Введіть" & i & "число")
  Next i
```

```
  Debug.Print "Початковий масив: ";
  For i = 1 To n
        Debug.Print a(i); " ";
  Next i
  Debug.Print " "
  x = InputBox("Введіть нове число!", "Введення")
  Do
    k = InputBox("Введіть позицію <=" & n, "Введення")
  Loop Until k <= n And k >= 1
  For i = n To k Step -1
         a(i + 1) = a(i)
  Next i
  a(k) = x  :  n = n + 1
  Debug.Print "Кінцевий  масив: ";
  For i = 1 To n
         Debug.Print a(i); " ";
  Next i
     Debug.Print " "
End Sub
```

При видаленні k-го елемента з масиву необхідно зсунути підмасив $a_{k+1},..., a_n$ вліво на одну позицію. Відповідний фрагмент програми виглядає так:

```
…
n = n - 1
for i=k to n
   a(i) = a(i+1)
Next i
…
```

Циклічне зсування масиву відзначається тим, що розмір масиву не змінюється. Циклічне зсування масиву вправо на одну позицію зображено на рис. 4.2. При зсуванні вправо напрям перегляду масиву і команди присвоювання у циклі такі ж самі, як і при

включенні елемента у масив, а при зсуванні вліво – такі ж самі, як і при видаленні елемента з масиву.

6	9	5	1	12	18	24		
24	6	9	5	1	12	18		

Рис. 4.2. Циклічне зсування масиву вправо на одну позицію

Фрагмент програми циклічного зсування масиву вправо на k позицій виглядатиме так:

```
…
for j=1 to k
    x = a(n)  ' x – проміжна величина
    for i=n to 2 Step -1
        a(i) = a(i-1)
    Next i
    a(1) = x
 Next j
…
```

Приклад 4.17. Дано натуральне число n і послідовність дійсних чисел a_1, a_2,..., a_n. Переставити члени послідовності так, щоб спочатку розмістилися всі її невід'ємні члени, а потім – всі від'ємні. Порядок розміщення як серед невід'ємних членів, так і серед від'ємних має зберегтися.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim x As Single
 Dim n As Byte, i As Byte, j As Byte, k As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
 ' Введення послідовності чисел - див. приклад 4.10!
 i = 1 ' початковий індекс невід'ємного члена
 For k = 1 To n
   If a(i) < 0 Then
      x = a(i) ' збереження від'ємного члена
      ' зсування масиву вліво на 1 позицію
```

```
      For j = i To n - 1
            a(j) = a(j + 1)
      Next j
      a(n) = x ' від'ємний член у кінець масиву
   Else
      i = i + 1
   End If
 Next k
 Debug.Print "Кінцевий  масив: ";
 For i = 1 To n
       Debug.Print a(i); " ";
 Next i
 Debug.Print " "
End Sub
```

Якщо `a(i)<0`, то ми розміщуємо його наприкінці масиву, попередньо зсунувши підмасив від `a(i+1)` до `a(n)` на одну позицію вліво. Перший від'ємний член спочатку займе останнє місце, а в подальшому при виявленні нового від'ємного члена зсуватиметься на одну позицію вліво.

Приклад 4.18. Дано натуральне число `n` і послідовність дійсних різних чисел a_1, a_2,..., a_n. Вибрати випадково k $(1 \le k \le n)$ чисел так, щоб жодне з них не повторилося.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim x As Single
 Dim n As Byte, i As Byte, j As Byte, k As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n)
 ' Введення послідовності чисел - див. приклад 4.10!
 Do : k=InputBox("Введіть кількість обраних чисел <= " & n)
 Loop Until k <= n And k >= 1
 For i = 1 To k
     j = i + Rnd * (n - i + 1) ' випадковий індекс
     x = a(j): a(j) = a(i): a(i) = x ' обмін
 Next i
```

```
 Debug.Print "Обрані числа: ";
 For i = 1 To k
     Debug.Print a(i); " ";
 Next i
 Debug.Print " "
End Sub
```

Вибираємо випадковий індекс `j` з проміжку [1; *n*] і відповідний йому елемент `a(j)` міняємо місцями з першим елементом масиву. Наступного разу індекс `j` обирається з проміжку [2; *n*] і елемент `a(j)` міняється місцями з другим елементом масиву і т.д. Вибрані елементи розташовуються на початку масиву і не беруть участі у наступних вибірках (вибірка без повторень).

Приклад 4.19. Дано натуральні числа `n`, `m` і два масиви дійсних чисел $a_1, a_2,..., a_n$ і $b_1, b_2,..., b_m$, впорядковані за неспаданням. Утворити з елементів цих масивів впорядкований за неспаданням масив *c*.

```
Option Explicit
Dim a() As Single
Dim b() As Single
Dim c() As Single
Public Sub main()
 Dim n As Byte, i As Byte, j As Byte, k As Byte, m As Byte
 Do  :  n = InputBox("Введіть n <=20")
 Loop Until n < 21 And n >= 1
 ReDim a(n)
 ' Введення першого масиву чисел - див. приклад 4.10!
 Do  :  m = InputBox("Введіть m <=20")
 Loop Until m < 21 And m >= 1
 ReDim b(m)
 ' Введення другого масиву чисел - див. приклад 4.10!
 ReDim c(n + m)
 i = 1:  j = 1
 Do While i <= n And j <= m
   If a(i) <= b(j) Then
          c(i + j - 1) = a(i): i = i + 1
   Else : c(i + j - 1) = b(j): j = j + 1
   End If
 Loop
```

```
 While i <= n : c(m + i) = a(i): i = i + 1 :  Wend
 While j <= m : c(n + j) = b(j): j = j + 1 :  Wend
 Debug.Print " "
 Debug.Print "Масив після злиття: ";
 For i = 1 To n + m
        Debug.Print c(i); " ";
 Next i
 Debug.Print " "
End Sub
```

4.3. Алгоритми пошуку в одновимірному масиві

Алгоритми пошуку застосовуються для визначення у масиві індексу елемента з потрібними властивостями. Розрізняють постановки задачі пошуку для першого та останнього входження елемента. В усіх нижче викладених алгоритмах вважатимемо, що проводиться пошук у масиві *a* з *n* цілих чисел індексу елемента, який дорівнює *x*.

Розрізняють також пошук у впорядкованому та невпорядкованому масивах. У невпорядкованому масиві пошук виконують за допомогою послідовного перегляду всього масиву, це називають *лінійним* пошуком.

Лінійний пошук здійснюється у циклі з умовою **While** або **Until** з подвійною умовою. Перша умова контролює індекс на приналежність масиву (`i<=n`). Друга умова – це умова пошуку: у циклі з **While** – умова продовження пошуку (`a(i)<>x`), а у циклі з **Until** – умова завершення пошуку (`a(i)=x`). У тілі ж циклу є тільки один оператор, який змінює індекс масиву. Після виходу з циклу необхідно перевірити, за якою з умов ми вийшли. В операторі **If** зазвичай повторюють першу умову циклу.

Приклад 4.20. Лінійний пошук в одновимірному масиві першого входження елемента, який дорівнює *x*.

```
Option Explicit
Dim a() As Integer
Public Sub main()
 Dim x As Integer
 Dim n As Byte, i As Byte
```

```
' Введення n - див. приклад 4.10!
 ReDim a(n)
 ' Введення послідовності чисел - див. приклад 4.10!
 x = InputBox("Введіть число", "Введення")
 i=1
 While i<=n and a(i)<>x
     i = i + 1
 Wend
 If i<=n Then
  MsgBox "1-е входження " & Str(x) & " у" & Str(i) & " місці"
 Else
  MsgBox " Не знайдено!"
 End If
End Sub
```

Приклад 4.21. Лінійний пошук в одновимірному масиві останнього входження елемента, який дорівнює *x*. Подаємо основний фрагмент програми.

```
...
 i=n+1
 Do
   i = i - 1
 Loop Until i<1 Or a(i)=x
 If i>=1 Then
 MsgBox "Останнє входж. " & Str(x) & " у" & Str(i) & " місці"
 Else
  MsgBox " Не знайдено!"
 End If
```

Існує так званий *пошук з бар'єром*. Ідея пошуку з бар'єром полягає в тому, щоб не перевіряти кожен раз у циклі умову, зв'язану з межами масиву. Це можна забезпечити, встановивши у масиві так званий *бар'єр*: будь-який елемент, який задовольняє умові пошуку. Тим самим буде обмежена зміна індексу. Існує два способи встановлення бар'єра: додатковим елементом або замість крайнього елемента масиву.

Вихід з циклу, в якому тепер залишається тільки умова пошуку, може відбутися або на знайденому елементі, або на бар'єрі.

Отже, після виходу з циклу перевіряється, чи не бар'єр ми знайшли? Обчислювальна складність пошуку з бар'єром є меншою, ніж у лінійного пошуку, однак є величиною того ж порядку, що і n – кількість елементів масиву.

Приклад 4.22. Лінійний пошук з бар'єром в одновимірному масиві першого входження елемента, який дорівнює *x*.

```
Option Explicit
Dim a() As Integer
Public Sub main()
 Dim x As Integer
 Dim n As Byte, i As Byte
 ' Введення n - див. приклад 4.10!
 ReDim a(n+1)
 ' Введення послідовності n чисел - див. приклад 4.10!
 x = InputBox("Введіть шукане число!")
 a(n + 1) = x: i = 1
 While a(i) <> x
     i = i + 1
 Wend
 If i <= n Then
   MsgBox "1-е входження " & Str(x) & " у " & Str(i) & " місці"
 Else
   MsgBox " Не знайдено!"
 End If
End Sub
```

Подаємо також головний фрагмент програми, якщо бар'єр є на останньому місці масиву.

```
     ...
 y = a(n) ' збереження останнього елемента
 a(n) = x ' установка бар'єра на останнє місце масиву
 i = 1
 While a(i) <> x
     i = i + 1
 Wend
```

```
If i < n Or y = x Then
  MsgBox "1-е входження " & Str(x) & " у " & Str(i) & " місці"
Else
  MsgBox " Не знайдено!"
End If
a(n) = y ' відновлення останнього елемента масиву
  …
```

Розглянемо пошук у *впорядкованому* масиві. У цьому випадку його можна значно прискорити, застосовуючи метод половинного ділення масиву (*двійковий* або *бінарний пошук*).

Нехай маємо масив *a* з n елементів, який впорядковано за неспаданням ($a_i \le a_{i+1}$). Ідея алгоритму полягає в тому, що масив кожен раз ділиться навпіл і вибирається та частина, де може перебувати потрібний елемент. Ділення продовжується доти, доки частина масиву для пошуку складатиметься з одного елемента. Після цього залишається тільки перевірити цей елемент, що залишився, на виконання умови пошуку. Подаємо фрагмент програми, що реалізує метод половинного ділення масиву:

```
  …
left = 1 : right=n ' ліва і права межі пошуку
While left < right
  m = (left + right) \ 2 ' середина проміжку
  If x > a(m) Then left = m + 1 Else right = m
Wend
If a(left)=x Then
  MsgBox "1-е входження " & Str(x) & " у " & Str(i) & " місці"
Else
  MsgBox " Не знайдено!"
End If
```

Наведений фрагмент програми дає змогу встановити позицію першого входження *x* у масив *a*. Наступні входження *x* визначають за допомогою лінійного пошуку в *a*, починаючи з позиції `left+1`.

У процесі роботи алгоритму двійкового пошуку розмір фрагмента, де цей пошук повинен продовжуватися, кожен раз змен-

шується приблизно удвічі. Це забезпечує обчислювальну складність алгоритму порядку $\log_2 n$, де n - кількість елементів масиву.

4.4. Алгоритми сортування в одновимірному масиві

Сортування масиву – це перерозташування елементів масиву у заданому порядку (домовимося розглядати випадок впорядкування масиву за неспаданням: $a_1 \le a_2 \le ... \le a_n$). Під *внутрішнім* сортуванням розуміють сортування, яке виконується в оперативній пам'яті комп'ютера без використання допоміжних масивів. *Головна мета* сортування – *полегшити подальший пошук*.

Сортування за допомогою *вибору*. Ідея методу полягає у *визначенні* максимального елемента масиву, який міняють місцями з останнім елементом (з номером n). Потім максимум обирають серед елементів з першого до передостаннього і ставлять на n–1-ше місце і т. д. Необхідно знайти n–1 максимум. Можна шукати не максимум, а мінімум і ставити його на перше місце, друге і т. д. Обчислювальна складність сортування за допомогою вибору – величина порядку $n \cdot n$, що записують як $O(n^2)$. Це пояснюється тим, що кількість порівнянь при пошуку першого максимуму дорівнює n–1; потім n–2, n–3, …, 1; разом: $n \cdot (n–1)/2$.

Подаємо фрагмент програми, що реалізує сортування за допомогою *вибору*:

```
For k=n To 2 Step -1
  m = 1 ' m - місце max
  For i=2 To k
    If a(i) > a(m) Then m = i
  Next i
  x = a(m) : a(m) = a(k) : a(k) = x
Next k
```

Сортування за допомогою *обміну* (метод "*бульбашки*"). Ідея методу полягає в тому, що послідовно порівнюють пари сусідніх елементів масиву. Якщо вони розташовані не в тому порядку, то здійснюють перестановку, міняючи місцями пару сусідніх елементів. Після одного такого проходу на останньому місці (но-

мер n) виявиться максимальний елемент (“спливає” перша “бульбашка”). Такий прохід повинен розглядати елементи до передостаннього і так далі. Усього потрібно n–1 проходів. Обчислювальна складність такого сортування $O(n^2)$.

Подаємо фрагмент програми, що реалізує сортування за допомогою *обміну*:

```
For k=n-1 To 1 Step -1 ' k - кількість пар порівняння
 For i=1 To k
   If a(i) > a(i+1) Then 'міняємо місцями сусідні елементи
   x = a(i) : a(i) = a(i + 1) : a(i + 1) = x
   End If
 Next i
Next k
```

Сортування за допомогою *впровадження*. Ідея цього методу полягає в тому, що кожен раз, маючи вже впорядкований масив з k елементів, ми додаємо ще один елемент, включаючи його в масив так, щоб впорядкованість не порушилася. Сортування може проводитися одночасно з введенням масиву. На початку сортування впорядкована частина масиву містить тільки один елемент. Різні методи пошуку місця для елемента, що впроваджується, спричиняють до різних модифікацій цього методу сортування. При використанні лінійного пошуку обчислювальна складність сортування впровадженням становить $O(n^2)$, а при використанні двійкового пошуку – $O(n \cdot \log_2 n)$, де n – кількість елементів.

Подаємо фрагмент програми, що реалізує сортування за допомогою впровадження при *лінійному пошукові* місця впровадження:

```
// k - кількість елементів у впорядкованій частині масиву
For k=1 To n-1
  x=a(k+1) : i=k
  While i>0 And a(i)>x
    a(i+1)=a(i) : i=i-1
  Wend
  a(i+1)=x
Next k
```

А тепер подаємо фрагмент програми, що реалізує сортування за допомогою *впровадження* при *бінарному пошуку* місця впровадження:

```
For k=1 To n-1
  x=a(k+1) : left=1 : right=k
  While left<right ' двійковий пошук останнього входження
    m=(left+right+1)\2
    if x>=a(m) Then left=m Else right=m-1
  Wend
  if x>=a(left) then left=left+1
  ' зсуваємо на одиницю праворуч частину масиву,
  ' звільняючи місце для включення x
  For i=k To left Step -1
      a(i+1)=a(i)
  Next i
  A(left)=x
Next k
```

4.5. Матричні задачі

У стандартній матричній задачі виникає необхідність:

- отримати нову матрицю за певним правилом;
- знайти деяку величину, використовуючи елементи матриці;
- перетворити матрицю певним способом;
- визначити, чи володіє матриця певною властивістю;
- виконати над матрицею (матрицями) певну операцію.

При реалізації матричних задач використовують *вкладені* (складні) цикли, які містять у собі один або декілька інших циклів. Такі цикли використано уже в алгоритмах сортування.

Приклад 4.23. Дано натуральне число n. Отримати дійсну матрицю $a[i,\ j]_{i,\ j=1,\ldots,n}$, для якої виконуються умови:

$$a[i,j]=\begin{cases}(i+j)^2, & \text{якщо } i<j;\\ 2, & \text{якщо } i=j;\\ 2i+5j, & \text{якщо } i>j.\end{cases}$$

Індекс *i* позначає рядок квадратної матриці, а *j* – стовпець. Рівні індекси (*i=j*) мають елементи головної діагоналі. Умова *i<j* вказує на те, що елементи розташовані над головною діагоналлю, а *i>j* – елементи розташовані під головною діагоналлю.

```
Option Explicit
Dim a() As Integer
Public Sub main()
   Dim i As Byte, j As Byte, n As Byte
   Do:   n = InputBox("Введіть n < 10", "Введення")
   Loop Until n < 10
   ReDim a(n, n)
   For i = 1 To n : a(i, i) = 2 :  Next i
   For i = 1 To n - 1
     For j = i + 1 To n
        a(i, j) = (i + j) ^ 2
     Next j
   Next i
   For i = 2 To n
    For j = 1 To i - 1
      a(i, j) = 2 * i + 5 * j
    Next j
   Next i
   For i = 1 To n
     Debug.Print ""
     For j = 1 To n
        Debug.Print a(i, j);
     Next j
   Next i
End Sub
```

Приклад 4.24. Дано натуральне число n. Визначити, скільки додатних елементів містить матриця $a[i,\ j]_{i,\ j=1,\ldots,n}$, у якої $a[i,\ j] = \sin(i + j)$.

Шукане число можна знайти і без оголошення масиву, використовуючи тільки формулу, що задає вигляд елементів масиву:

```
Option Explicit
Public Sub main()
  Dim i As Byte, j As Byte, k As Byte, n As Byte
  Do:   n = InputBox("Введіть n < 10", "Введення")
   Loop Until n < 10
  k =0
  For i = 1 To n
     For j = 1 To n
        If sin(i + j) > 0 Then k = k + 1
     Next j
  Next i
  Debug.Print "k="; k
End Sub
```

Приклад 4.25. Дано натуральне число n. Визначити суму елементів матриці $a[i,\ j]_{i,\ j=1,\ldots,n}$, розташованих на головній діагоналі або над нею, що перевищують усі елементи, розташовані під головною діагоналлю.

Серед елементів матриці, розташованих під головною діагоналлю, знайдемо найбільший елемент. Тоді елементи, які розташовані на головній діагоналі або над нею і є більшими за знайдений найбільший елемент, задовольнятимуть умові задачі.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim i As Byte, j As Byte, n As Byte
 Dim max As Single, s As Single
 Do : n = InputBox("Введіть n < 10", "Введення")
 Loop Until n < 10
 ReDim a(n, n)
 For i = 1 To n
  For j = 1 To n
```

```
    a(i,j) = Rnd * 10 ' ініціал. матриці випадковими числами
  Next j : Next i
 max = a(2, 1)
 For i = 3 To n ' визначення максимуму
  For j = 1 To i-1
    If a(i, j) > max Then max = a(i, j)
  Next j : Next i
s = 0
 For i = 1 To n
  For j = i To n
    If a(i, j) > max Then s = s + a(i, j)
  Next j
 Next i
 Debug.Print "s="; s
End Sub
```

Приклад 4.26. Дано натуральне число n і деяка квадратна матриця $a[i, j]_{i,j=1,\ldots,n}$. Перетворити цю матрицю так, щоб перший рядок помінявся місцями з останнім, другий – з передостаннім і т.д.

```
Option Explicit
Dim a() As Single
Public Sub main()
 Dim i As Byte, j As Byte, n As Byte, x As Single
 Do : n = InputBox("Введіть n < 10", "Введення")
 Loop Until n < 10
 ReDim a(n, n)
 For i = 1 To n
  For j = 1 To n : a(i,j) = i+j  ' ініціалізація матриці
  Next j
 Next i
 For i = 1 To n\2 ' перестановка рядків матриці
  For j = 1 To n
   x = a(i,j) : a(i,j) = a(n-i+1,j) : a(n-i+1,j) = x
  Next j
 Next i
   For i = 1 To n   '  виведення матриці
     Debug.Print ""
```

```
      For j = 1 To n
         Debug.Print a(i, j);
      Next j
    Next i
End Sub
```

Приклад 4.27. Написати фрагменти програм, які реалізують головні операції над матрицями: 1 – додавання (віднімання) двох матриць; 2 – множення матриці на скаляр; 3 – множення двох матриць; 4 – транспонування квадратної матриці.

1. Матриці, які беруть участь в операціях додавання (віднімання), повинні мати одинакові розміри, адже ці операції виконуються поелементно:

```
For i = 1 To m
  For j = 1 To n
   c(i, j) = a(i, j) + b(i, j)
  Next j
Next i
```

Для визначення різниці матриць достатньо змінити "+" на "–".

2. Множення матриці на скаляр також виконується поелементно:

```
For i = 1 To m
  For j = 1 To n
   c(i, j) = a(i, j) * p
  Next j
Next i
```

3. Матриці, які беруть участь в операції множення, мають такі розміри: $m\overbrace{A}^{n} \times n\overbrace{B}^{k} = m\overbrace{C}^{k}$. Тоді будь-який елемент c_{ij} $(i = 1...m; j = 1...k)$ визначають за формулою $c_{ij} = \sum_{l=1}^{n} a_{il} \cdot b_{lj}$:

```
For i = 1 To m
  For j = 1 To k
   s = 0
```

```
    For l = 1 To n
      s = s + a(i, l) * b(l, j)
    Next l
    c(i, j) = s
  Next j
Next i
```

4. Транспонування квадратної матриці полягає у заміні рядків матриці відповідними стовпцями, і навпаки (тобто елементи d_{ij} і d_{ji} мають помінятися місцями).

Під час транспонування головна діагональ матриці залишається без змін, а обмін здійснюємо тільки для елементів, розташованих над цією діагоналлю (елементи під діагоналлю поміняються автоматично внаслідок симетрії):

```
For i = 1 To n-1
  For j = i+1 To n
   s = d(i, j)
   d(i, j) = d(j, i)
   d(j, i) = s
  Next j
Next i
```

4.6. Власні типи даних (записи)

У процесі розробки програми можна створювати свої власні нові типи даних, комбінуючи вже наявні вбудовані типи. Такі типи називають по-різному:

- типами, визначеними користувачем;
- власними типами даних;
- спеціальними типами;
- записами.

Всередині спеціального типу можна використовувати масиви постійної або змінної довжини, а також раніше описані типи, визначені користувачем. Синтаксис опису типу даних, визначених користувачем, набуває вигляду:

```
[Private | Public] Type назва_типу
          Елемент1 (([Розмірність])) As Тип
```

```
        [Елемент2 [([Розмірність])] As Тип]
        …
End Type
```

Визначення власного типу даних можливе тільки у секції *General–Declarations* контейнера, а глобального (`Public`) – у цій же секції, тільки для модуля. Визначивши власний тип даних, можна використовувати його для оголошення змінних цього типу, які слугують локальними, глобальними чи змінними контейнера. Доступ до елементів змінної власного типу здійснюється шляхом указівки крапки після назви змінної.

Приклад 4.28. Уведення власного типу та його використання:

```
' General – Declarations (Module)
Public Type usrType
  Name As String * 40
  Price As Currency
End Type
' General – Declarations) (Form)
Dim usrTools As usrType
Private Sub Commandl_Click()
  usrTools.Name = "Молоток"
  usrTools.Price =9.99
End Sub
```

Приклад 4.29. Уведення власного типу `Client` і його використання при створенні масиву даних про клієнтів:

```
Public Type Client
  Name As String
  Address As String
  CityState As String
  Phone As string
End Type
Public Sub main()
Dim MyClients(100) As Client
…
  MyClients(10).Name="Перебендя Іван"
  MyClients(10).Adress="в. С.Бандери, 7/45"
  MyClients(10).CityState="м.Львів, Україна"
```

```
    MyClients(10).Phone="(0322)789-439"
…
End Sub
```

- **? Запитання для самоперевірки**

1. Що таке масив?
2. Як можна задати нижню межу масиву?
3. Як описують одновимірний масив?
4. Як описують багатовимірний масив?
5. Що таке елемент масиву?
6. Що таке лінійний пошук?
7. Що таке пошук з бар'єром?
8. Що таке бінарний пошук?
9. Охарактеризуйте спосіб сортування вектора вибором.
10. Охарактеризуйте спосіб сортування вектора обміном.
11. Охарактеризуйте спосіб сортування вектора включенням.
12. Напишіть фрагмент програми множення матриць.
13. Напишіть фрагмент програми транспонування матриці.
14. Що таке запис?
15. Як описують власний тип даних?
16. Для чого використовують власний тип даних?

- **Завдання для програмування**

Завдання 4.1. Дано натуральне число `n` і дійсні числа a_1, a_2,..., a_n. Скласти програми розв'язання таких задач (у задачах 1-5 передбачити опрацювання ситуації відсутності від'ємних елементів):

1. Знайти добуток елементів масиву, розташованих після останнього від'ємного числа.
2. Знайти середнє арифметичне елементів масиву, розташованих після останнього від'ємного числа.
3. Замінити всі елементи, розташовані перед першим від'ємним числом, на число 2.
4. Замінити всі елементи, розташовані після останнього від'ємного числа, на число 10.
5. Знайти середнє арифметичне елементів масиву, розташованих між першим та останнім від'ємними числами.

6. Замінити всі від'ємні числа на їхні квадрати і впорядкувати після цього масив за незростанням.
7. Замінити всі від'ємні числа на їхні модулі і впорядкувати після цього масив за незростанням.
8. Знайти добуток елементів масиву, розташованих після останнього входження найменшого числа.
9. Знайти суму елементів масиву, розташованих перед останнім входженням найменшого числа.
10. Знайти середнє арифметичне елементів масиву, розташованих після останнього входження найбільшого числа.
11. Знайти добуток елементів масиву, розташованих перед останнім входженням найбільшого числа.
12. Знайти середнє арифметичне елементів масиву, розташованих між останніми входженнями найбільшого та найменшого чисел.
13. Знайти суму елементів масиву, розташованих між першими входженнями найбільшого та найменшого чисел.
14. Замінити всі додатні числа на їхні квадрати і впорядкувати після цього масив за незростанням.
15. Замінити всі нульові числа на число 3 і впорядкувати після цього масив за незростанням.

Завдання 4.2. Дано натуральне число `n` і цілі числа a_1, a_2,..., a_n. Скласти програми розв'язання таких задач:

1. Впорядкувати масив за незростанням методом обміну.
2. Впорядкувати масив за незростанням методом вибору.
3. Впорядкувати масив за незростанням методом впровадження.
4. Стиснути масив, видаливши з нього числа, модуль яких менший за 3.
5. Стиснути масив, видаливши з нього всі нулі.
6. Стиснути масив, видаливши з нього всі від'ємні числа.
7. Після кожного нуля вставити у масив число 100.
8. Після кожного від'ємного числа вставити у масив два нулі.
9. Кожне від'ємне число у масиві оточити з обох боків нулями.
10. Впорядкувати масив за неспаданням модулів чисел методом обміну.
11. Впорядкувати масив за неспаданням модулів чисел методом вибору.
12. Впорядкувати масив за неспаданням модулів чисел методом впровадження.

13. Впорядкувати масив за незростанням модулів чисел методом обміну.
14. Впорядкувати масив за незростанням модулів чисел методом вибору.
15. Впорядкувати масив за незростанням модулів чисел методом впровадження.

Завдання 4.3. Дано цілочисельну прямокутну матрицю. Скласти програми розв'язання таких задач:

1. Визначити кількість рядків, які не містять нулів.
2. Визначити кількість стовпців, які не містять нулів.
3. Визначити кількість стовпців, які містять хоча б один нуль.
4. Визначити кількість рядків, які містять хоча б один нуль.
5. Визначити добуток елементів у кожному рядку, що не містить нуля.
6. Визначити добуток елементів у кожному стовпці, що не містить нуля.
7. Визначити суму елементів у кожному рядку, що не містить від'ємних чисел.
8. Визначити суму елементів у кожному стовпці, що не містить від'ємних чисел.
9. Визначити середнє арифметичне елементів у кожному рядку, який містить хоча б один нуль.
10. Визначити середнє арифметичне елементів у кожному стовпці, який містить хоча б один нуль.
11. Визначити суму елементів у кожному рядку, який містить хоча б одне від'ємне число.
12. Визначити суму елементів у кожному стовпці, який містить хоча б одне від'ємне число.
13. Знайти номер першого з рядків, що не містить від'ємних чисел.
14. Знайти номер останнього з рядків, що не містить від'ємних чисел.
15. Знайти номер останнього зі стовпців, що не містить нуля.

5. Процедури та функції

План викладу матеріалу:

1. Загальні положення.
2. Опис підпрограм.
3. Виклик підпрограм.
4. Механізм взаємозв'язку параметрів і аргументів.
5. Підпрограми з довільною кількістю параметрів.
6. Функції роботи з масивами.
7. Спеціальні випадки задання аргументів.
8. Рекурсія.

Ключові терміни розділу

- *Допоміжні алгоритми*
- *Функція, опис функції*
- *Оператор виклику процедури*
- *Типи параметрів*
- *Передача аргументів за назвою*
- *Аргументи за домовленістю*
- *Аргументи-записи*
- *Процедура, опис процедури*
- *Структура модуля, пошук підпрограм у модулях*
- *Виклик функції*
- *Передача аргументів за значенням*
- *Довільна кількість параметрів*
- *Аргументи-масиви*
- *Поняття рекурсії*

5.1. Загальні положення

Visual Basic дає змогу виокремити фрагменти програми у допоміжні алгоритми, завдяки чому виникають добре структуровані програми. Мови програмування, в яких передбачено допоміжні алгоритми, називають *процедурно-орієнтованими*. Структуровані програми зазвичай простіші для розуміння та налагодження.

Наявність допоміжних алгоритмів у мові програмування дає змогу застосовувати спеціальні методи проектування і розробки складних програмних комплексів. Перший називають *методом низхідного програмування* або розробкою програми "зверху – вниз": спочатку створюють головну програму, в якій передбачено наявність деяких допоміжних алгоритмів, що вирішують певні задачі; потім переходять до детальної розробки перелічених вище допоміжних алгоритмів.

Іншим підходом проектування і розробки програм *є метод висхідного програмування* або проектуванням "знизу – вгору". У

цьому випадку все розпочинається зі створення невеликих допоміжних алгоритмів, з яких потім створюють більш складні допоміжні алгоритми і, зрештою, головну програму.

У Visual Basic допоміжні алгоритми оформляють у вигляді *процедур* або *функцій*, які за своєю структурою подібні до звичайних програм і мають загальну назву – *підпрограми*. Підпрограми зазвичай розташовують на початку програми.

5.2. Опис підпрограм

Процедури та функції є головними компонентами модулів VBA. Загальний вигляд процедури:

```
[Специфікатор] Sub НазваПроцедури([СписокПараметрів])
    ТілоПроцедури
End Sub
```

- `Специфікатор` – це одне з можливих значень:
  `[Private | Public] [Static]`
  Використання специфікаторів **Private**/**Public** розглянуто у другому розділі. Специфікатор **Static** означає, що всі локальні змінні процедури будуть статичними.
- Необов'язковий елемент `СписокПараметрів` – це послідовність оголошень параметрів процедури, розділених комами:
  `ОголошенняПараметра1 [, ОголошенняПараметра2] …`
  `ОголошенняПараметра` має вигляд:
  `[СпецифікаторПараметра] Параметр[( )] [As тип] [=Значення]`
  де `СпецифікаторПараметра` має вигляд:
  `[Optional] [ByVal | ByRef] [ParamArray]`
  Використання цих значень розглянемо згодом.
- `Параметр` – назва змінної. Водночас з викликом процедури її аргументи (фактичні параметри) можуть бути не тільки назвами змінних, але й виразами. Результати роботи підпрограми (певні *значення*) передаються у точку виклику процедури за допомогою одного чи декількох параметрів.
- Якщо після параметра записані *круглі дужки*, то це означає, що цей параметр є масивом.

- `ТілоПроцедури` задає послідовність операторів, серед яких можуть міститися оператори оголошення локальних даних процедури (змінні, константи, масиви тощо).
- У тіло процедури можуть входити оператори **`Exit Sub`**, які ініціюють негайний вихід з процедури і передачу керування у викликаючу процедуру/функцію, наприклад:

  ```
  If a<0 Then c=0: Exit Sub Else c=Sqr(a)
  ```

Функція відрізняється від процедури тим, що її назву використовують як змінну, що повертає значення функції у точку виклику. Крім того, замість ключового слова **`Sub`** вживають слово **`Function`**, а після списку параметрів може зазначатися тип результату функції. Якщо тип результату функції не зазначено, то за домовленістю приймається тип **`Variant`**. Функція повертає тільки одне значення, на відміну від процедури, що може повертати декілька значень. Загальний вигляд функції:

```
[Специфікатор] Function _
          НазваФункції([СписокПараметрів]) [As тип]
  ТілоФункції
End Sub
```

- У тілі функції має бути оператор присвоювання типу:

  ```
  НазваФункції = Значення
  ```

  `Значення` – деякий вираз. Вираз обчислюється, і отримане *значення* передається у точку виклику функції через її назву. Якщо при виході з функції *значення* `НазвіФункції` явно не присвоєно, то функція повертає *значення за домовленістю* для відповідного типу: для числових типів – нуль, для рядків – порожній рядок і т.д.
- У `ТілоФункції` можуть входити оператори **`Exit Function`**, які ініціюють негайний вихід з функції та передачу керування у викликаючу підпрограму.

Програму на мові VB сформовано з одного або декількох модулів. Модулі складаються з підпрограм. Підпрограми можуть викликати інші підпрограми. Коли одна підпрограма викликає іншу, то VB спочатку шукає підпрограму, що викликається, у тому самому модулі, в якому знаходиться підпрограма, що викликає. Якщо

VB не знаходить необхідної підпрограми в цьому модулі, то переглядаються усі модулі поточного проекту. З метою прискорення процесу виклику підпрограм, розташованих у різних модулях, можна зазначати назву модуля (через крапку) у вигляді префікса назви підпрограм.

Зазвичай текст модуля на мові VB розпочинається з опцій, які керують описом змінних, способом порівняння рядків і т.д. Після цього йде оголошення глобальних змінних і/або констант цього модуля. Далі йде текст функцій і процедур, які формують модуль.

Якщо всі підпрограми, що містяться у декількох модулях одного проекту, мають різні назви, то не виникає проблем під час виклику будь-якої підпрограми. Однак інколи необхідно мати дві підпрограми з однаковими назвами, які знаходяться у різних модулях. Під час виклику такої підпрограми треба вказувати назву підпрограми та префікс (через крапку) – *назву модуля*, в якому підпрограму розташовано.

5.3. Виклик підпрограми

З метою використання підпрограми її викликають. Процедуру з параметрами можна викликати тільки з іншої підпрограми, використавши її назву зі списком *аргументів* (фактичних параметрів), які замінюють *параметри* (або присвоюються параметрам), за допомогою оператора *виклику процедури*:

```
НазваПроцедури [Аргумент1, Аргумент2, ...]
```

або

```
Call НазваПроцедури([Аргумент1, Аргумент2, ...])
```

Наприклад, такі виклики процедури є еквівалентними:

```
FrontOut 8, i+5      і      Call FrontOut(8, i+5)
```

Функцію можна викликати так само, як і процедуру, за допомогою окремого оператора виклику, проте найчастіше використовують її специфічний виклик у вигляді конструкції

```
НазваФункції([Аргумент1, Аргумент2, ...])
```

Під час виклику підпрограми можна використовувати поіменовані аргументи, наприклад:

```
НазваПідпрограми Параметр1:=Аргумент1, Параметр2:=Аргумент2, ...
```

Порядок поіменованих аргументів у списку є довільним (не залежить від порядку розташування відповідних параметрів у заголовку підпрограми). Цей спосіб виклику підпрограми особливо зручний у випадку, коли підпрограма має необов'язкові параметри, які завжди розташовують наприкінці списку параметрів. Якщо деякі необов'язкові параметри не зазначено під час виклику, то замість них компілятор підставляє значення за домовленістю.

Якщо не подано тип параметра підпрограми, то за домовленістю йому буде приписано тип **Variant**. Щоб уникнути помилок, зумовлених неправильним використанням параметрів, необхідно вказувати їхній тип у заголовку підпрограми. Задання типів параметрів робить надійнішим програмний код, оскільки помилка, спричинена неправильною передачею аргументів у функцію або процедуру, виявляється до початку виконання програми. Бажано явно вказувати тип результату, який повертається функцією.

Приклад 5.1. Виклик процедури зі списком поіменованих аргументів:

```
Sub Mul(ByVal Vl As Single, ByVal V2 As Single, _
        Prod As Single)
  Prod = Vl * V2
End Sub
Sub Main()
 Dim Result As Single
 Mul Prod:=Result, Vl:=15.4, V2:=2 : MsgBox Result
End Sub
```

Викличемо цю ж процедуру з простим списком аргументів:

```
Sub Main()
  Dim Result As Single
  Call Mul(15.4, 2, Result) : MsgBox Result
End Sub
```

Приклад 5.2. Опис і виклик функції:

```
Function Mul(ByVal Vl As Single, ByVal V2 As Single) _
           As Single
  Mul = Vl * V2
End Function
Sub Main()
```

```
    Dim Result As Single
    Result = Mul(15.4, 2) : MsgBox Result
End Sub
```

5.4. Механізм взаємозв'язку параметрів і аргументів

Під час виклику підпрограми аргументами можуть слугувати деякі змінні. Альтернативні ключі **ByVal** і **ByFef** визначають спосіб передачі аргументів-змінних у підпрограму:

- Ключ **ByVal** означає передачу аргументу-змінної *за значенням*, тобто під час виклику підпрограми створюється локальна змінна – копія аргументу. Будь-яка зміна значення цієї локальної змінної не відображається на значенні аргументу-змінної. Передача аргументу-змінної за значенням можлива тільки для параметрів, які тільки передають інформацію у підпрограму, проте не відображають результати. Якщо аргументом є деякий вираз, зокрема константа, то відповідний параметр повинен описуватися як **ByVal** (радимо ще раз уважно проаналізувати приклади попереднього параграфа).
- Ключ **ByFef** означає передачу аргументу-змінної *за назвою* (за посиланням). Цей ключ встановлюється *за домовленістю*. Всередині процедури цим змінним може бути зроблено присвоєння деякого нового значення, яке зберігається у змінній і після повернення з процедури. При використанні аргументу за назвою у процедуру передається фізична адреса розташування цього аргументу у пам'яті (посилання на змінну). Аргументи, які є масивами, об'єктами чи структурами, завжди передаються за назвою.

Приклад 5.3. Передача змінної-аргументу за *назвою*:

```
Sub Param (K As Integer)
  K = K + 10
End Sub
Sub Main()
 Dim i As Integer
 i = 10 : Param I : MsgBox i ' Результат дорівнює 20
End Sub
```

Приклад 5.4. Передача змінної-аргументу за *значенням*:

```
Sub ParamMod (ByVal K As Integer)
  K = K + 10
End Sub
Sub Main()
  Dim i As Integer
  i = 10  :  ParamMod i
  MsgBox i  ' Результат дорівнює 10
End Sub
```

5.5. Підпрограми з довільною кількістю параметрів

Як і в інших мовах програмування, у VB можна створювати підпрограми з довільною кількістю параметрів. З цією метою перед останнім параметром у заголовку підпрограми необхідно записати ключове слово **ParamArray**, перетворивши його тим самим у масив змінних типу **Variant**. У самій підпрограмі для операцій зі всіма елементами цього масиву зазвичай використовують цикл **For Each – Next**.

Приклад 5.5. Процедура з довільною кількістю параметрів:

```
Sub MyMul(Product As Single, ParamArray Values())
  Dim Value As Variant, x As Single
  x = 1
  For Each Value In Values
     x = x * Value
  Next Value
  Product = x
End Sub
Sub Main()
  Dim res As Single
  MyMul  res, 10, 20, 30
  MsgBox res  ' Результат дорівнює 6000
  MyMul  res, 10, 20, 30, 40, 50
  MsgBox res  ' Результат дорівнює 1.2E+07
End Sub
```

5.6. Функції роботи з масивами

У мові Visual Basic є достатньо широкий набір вбудованих функцій роботи з масивами (табл. 5.1).

Таблиця 5.1. Вбудовані функції роботи з масивами

Функція	Опис
`Array(список_аргументів)`	Повертає дані типу **`Variant`** у вигляді масиву. `Список_аргументів` складається з довільної послідовності значень, розділених комами, які присвоюються елементам масиву, розташованому у змінній типу **`Variant`**. Якщо `список_аргументів` не містить значень, то створюється масив нульової довжини
`Erase(назва_масиву)`	Функція використовується для очищення масиву. Для масивів фіксованої довжини ця функція очищає всі значення масиву, проте не знищує пам'ять, яку займає масив. Для динамічного масиву ця функція не тільки очищає всі значення масиву, але й звільняє пам'ять, яку займав масив
`IsArray(ідентифікатор)`	Функція повертає `True`, якщо `ідентифікатор` є назвою масиву. Цю функцію найчастіше застосовують до зміних типу **`Variant`**
`IsMissing(ідентифікатор)`	`Ідентифікатор` позначає параметр з *аргументом за домовленістю*. Функція повертає `True`, якщо при виклику підпрограми аргумент для відповідного параметра не передавався
`LBound(масив [,розмірність])`	Функція повертає нижню межу індексу в `масиві` для зазначеної `розмірності`. За домовленістю `розмірність = 1`
`UBound(масив [,розмірність])`	Функція повертає верхню межу індексу в `масиві` для зазначеної `розмірності`. За домовленістю `розмірність = 1`

Функція `Array` дає змогу створити масив під час виконання програми без попереднього оголошення. Функція `Array` повертає одновимірний масив елементів типу **`Variant`** та виконує задання нижньої межі індексу відповідно до опції **`Option Base`**.

Приклад 5.6. Використання функції `Array`:

```
Sub Main()
Dim Data
  Data = Array("Петриченко", 90, #3/21/1913#)
  MsgBox Data(0) & ",  вік " & Data(1) & _
  ",  народився " & Data(2)
End Sub
```

5.7. Спеціальні випадки задання аргументів

У підпрограмах можна вводити параметри з *аргументами за домовленістю*. З цією метою при оголошенні параметра потрібно використати ключове слово **Optional**, яке засвідчує, що відповідний параметр має аргумент за домовленістю. Такі параметри ще називають *необов'язковими*.

Якщо при виклику підпрограми аргумент для відповідного параметра не зазначено, то обирається значення аргументу за домовленістю. Необов'язкові параметри можуть мати лише тип **Variant** і стояти тільки наприкінці списку оголошень параметрів.

Задання параметрові значення аргумента за домовленістю реалізують зазвичай в операторі галуження через перевірку відсутності аргументу (для цього слугує вбудована функція `IsMissing`).

Приклад 5.7. Процедура з *необов'язковим* параметром

```
Sub MulWp(P As Single, ByVal V1 As Single, Optional ByVal V2)
   If IsMissing(V2) Then V2 = 10
   P = V1 * V2
End Sub
Sub Main()
 Dim res As Single
 MulWp res, 10, 20 : MsgBox res ' Результат = 200
 MulWp res, 10     : MsgBox res ' Результат = 100
End Sub
```

Значення аргументу за домовленістю для параметра `V2` можна задати під час його оголошення, наприклад:

```
Sub MulWp(P As Single, ByVal Vl As Single, _
          Otional ByVal V2 = 10)
  P = VI * V2
End Sub
```

Аргументами підпрограми можуть слугувати *масиви*. Підпрограмі передається назва масиву, а його межі для кожної розмірності визначають за допомогою вбудованих функцій `LBound` і `Ubound`.

Приклад 5.8. Використання аргументів-масивів:

```
Option Base 1
Public Function SP(x() As Integer, y() As Integer) _
                                                  As Integer
 ' Обчислює скалярний добуток двох векторів
 Dim i As Integer, Sum As Integer
 Sum = 0
 For i = LBound(x) To UBound(x)
   Sum = Sum + x(i) * y(i)
 Next i
 SP = Sum
End Function
Public Sub Main()
 Dim a(1 To 5) As Integer, b(1 To 5) As Integer
 Dim c As Variant, Res As Integer, i As Integer
 c = Array(1, 2, 3, 4, 5)
 For i = 1 To 5  ' Заповнення масиву a
  a(i) = c(i)
 Next i
 c = Array(5, 4, 3, 2, 1)
 For i = 1 To 5  ' Заповнення масиву b
  b(i) = c(i)
 Next i
 Res = SP(a, b) : MsgBox "Скаляр. добуток=" & Str(Res)
End Sub
```

Подаємо приклад передачі аргументів – *змінних спеціального типу* (записів) під час виклику підпрограми. Такі аргументи, як і аргументи-масиви, можуть передаватися тільки за назвою.

Приклад 5.9. Процедура `Disp` виводить на екран значення полів запису типу `PersonalData`. Оголошення спеціального типу `PersonalData` розташовують у загальній області модуля:

```
Private Type PersonalData
  PName As String
  PAge As Integer
End Type
```

```
Sub Main()
  Dim User As PersonalData
  User.PName = "Симоненко" : User.PAge = 38
  Disp User.PName, User.PAge
End Sub
Sub Disp(UserName, UserAge)
  MsgBox UserName & ", вік " & UserAge & "."
End Sub
```

Отже, процедурі `Disp` передаються поля запису `User.Pname` та `User.PAg` і процедура виводить на екран обидва значення. Відповідні параметри `UserName` і `UserAge` мають тип `Variant`. Аргументом процедури `Disp` може слугувати і запис типу `PersonalData`. Тоді наша програма матиме вигляд:

```
Sub Main()
  Dim User As PersonalData
  User.PName = "Симоненко" : User.PAge = 41
  Disp User
End Sub
Sub Disp(Zapys As PersonalData)
  MsgBox Zapys.PName & ", вік " & Zapys.PAge & "."
End Sub
```

5.8. Рекурсія

Рекурсія – це спосіб організації обчислювального процесу, за якого підпрограма під час виконання звертається сама до себе. Рекурсія аналогічна методу математичної індукції: базі індукції відповідає база рекурсії; припущенню індукції відповідає припущення про те, що потрібна підпрограма вже написана; кроку індукції відповідає виклик рекурсивної підпрограми, що створюється. У будь-якій рекурсії необхідно передбачити умову завершення процесу, тобто момент відсутності виклику.

Приклад 5.10. Обчислити `n`-е число Фіббоначчі (див. розділ 3):

```
Function F(k As Byte) As Long
  If k < 2 Then F = 1 Else F = F(k - 1) + F(k - 2)
End Function
```

```
Sub Main()
Dim n As Byte, m As Long
  n = InputBox("Введіть n")
  m = F(n)
  MsgBox n & "-е число Фіббоначчі=" & m
End Sub
```

? Запитання для самоперевірки

1. З якою метою використовують допоміжні алгоритми?
2. Як описують процедуру?
3. Як описують функцію?
4. Що таке оператор процедури?
5. Як реалізовується виклик функції?
6. Що означає передача аргументу-змінної за значенням?
7. Що означає передача аргументу-змінної за назвою?
8. Що таке необов'язкові параметри?
9. З якою метою використовують функцію `IsMissing`?
10. З якою метою використовують функції `LBound` і `Ubound`?
11. З якою метою використовують функцію `Array`?
12. Як у підпрограму передаються аргументи-масиви?
13. Як у підпрограму передаються аргументи-записи?

💻 Завдання для програмування

Завдання 5.1. Виконати *завдання 3.2* (с. 59), оформивши обчислення відповідної функції у вигляді *підпрограми-процедури*.

Завдання 5.2. Виконати *завдання 3.2* (с. 59), оформивши обчислення відповідної функції у вигляді *підпрограми-функції*.

Завдання 5.3. Виконати *завдання 3.4* (с. 60), оформивши обчислення функції за допомогою розкладу у ряд Тейлора у вигляді *підпрограми-функції*.

Завдання 5.4. Виконати *завдання 4.1* (с. 90), оформивши розв'язання головної функціональної задачі у вигляді *підпрограми-функції*.

Завдання 5.5. Виконати *завдання 4.2* (с. 91), оформивши розв'язання головної функціональної задачі у вигляді *підпрограми-процедури*.

Завдання 5.6. Виконати *завдання 4.3* (с. 92), оформивши розв'язання головної функціональної задачі у вигляді *підпрограми-функції*.

6. Робота з рядками

План викладу матеріалу:

1. Вбудовані функції роботи з рядками.
2. Порівняння рядків.
3. Типові задачі опрацювання рядків символів.
4. Побудова підпрограм роботи з рядками символів.

Ключові терміни розділу

- *Функції* `Asc` *і* `Chr`
- *Функції зміни регістрів букв*
- *Функція* `Len`
- *Функція* `StrReverse`
- *Функція* `Val`
- *Функції пошуку підрядка*
- *Функції видалення пропусків*
- *Функція* `Space`
- *Функція* `Str`
- *Порівняння рядків символів*

6.1. Вбудовані функції роботи з рядками

Для роботи з рядками у Visual Basic використовують *операцію об'єднання* (або *конкатенації*) і вбудовані функції (таблиця 6.1).

Таблиця 6.1. Вбудовані функції роботи з рядками

Функція	Опис
1	2
`Asc(рядок)`	Повертає код першого символу `рядка`. Якщо `рядок` порожній, то виникне помилка
`Chr(код_символу)`	Повертає значення типу **`String`**, яке містить символ, що відповідає заданому `коду_символу`. Коди 0-31 відповідають стандартним символам керування
`InStr([старт,] pc1, pc2 [тип_порівняння])` `InStrRev([старт,] pc1, pc2 [тип_порівняння])`	Повертає позицію першого входження рядка символів `pc2` у рядок символів `pc1`. Якщо входження немає, то повертається нуль. Параметр `старт` задає позицію, з якої починається пошук (за домовленістю – перша позиція). `Тип_порівняння` набуває значення: 0 (за домовленістю) – порівняння типу `Binary` і 1 – порівняння типу `Text`. Функція `InStrRev` здійснює пошук останнього входження `pc2` у `pc1`

Закінчення табл. 6.1

1	2
`LCase(рядок)`	Повертає копію `рядка`, символи якого приведені до нижнього регістру
`Left(рядок, кількість)`	Повертає підрядок, який збігається з зазначеною `кількістю` символів `рядка` зліва
`Len(рядок)`	Повертає кількість символів у `рядку`
`LTrim(рядок)`	Повертає копію `рядка`, з якого видалені початкові пропуски
`Mid(рядок, позиція [,кількість])`	Повертає рядок символів, який збігається з зазначеною `кількістю` символів `рядка` від заданої `позиції`
`Right(рядок, кількість)`	Повертає рядок символів (підрядок), який збігається з зазначеною `кількістю` символів `рядка_символів` справа
`RTrim(рядок)`	Повертає копію `рядка`, з якого видалені кінцеві пропуски
`Trim(рядок)`	Повертає копію `рядка`, з якого видалені початкові та кінцеві пропуски
`Space(число)`	Повертає копію рядка символів, який складається із заданого `числа` пропусків
`Str(числовий_вираз)`	Перетворює `числовий_вираз` у рядок
`StrComp(pc1, pc2 [, тип-порівняння])`	Якщо рядок символів `pc1` менший, ніж рядок символів `pc2`, то результат дорівнює 1. Якщо рядки символів рівні, то результат дорівнює 0. Якщо рядок символів `pc1` більший, ніж рядок символів `pc2`, то результат дорівнює –1
`String(кількість, символ)`	Повертає рядок заданої `кількості` повторень одного і того ж `символу`
`StrReverse(pc)`	Змінює рядок символів `pc` на зворотний
`UCase(рядок)`	Повертає копію `рядка`, символи якого приведені до верхнього регістру

Під рядком тут розуміють вираз типу **String**, а під числом, кількістю, стартом і позицією – числові вирази, що набувають натуральних значень. Функції, які працюють з рядками символів, існують у двох варіантах. Усі перелічені функції повертають значення типу **Variant**. У другому варіанті назви функцій мають суфікс "**$**", наприклад: Mid$, LCase$ і т.д. У цьому випадку функції повертають значення типу **String** і виконуються швидше. Проте такі функції не можуть коректно працювати з рядками, що мають значення **Null**.

Функція Str перетворює числове значення у символьне зображення. Функція Val перетворює рядок символів у числове значення. При перетворенні рядка символів у число враховуються усі цифрові символи, розташовані в рядку зліва направо. Пропуски, що знаходяться на початку і наприкінці символьного рядка, ігноруються. Пропуски всередині рядка неприпустимі. Якщо перший символ виразу не є цифрою, функція Val поверне значення нуль. Функції LTrim, Rtrim і Trim використовують з метою видалення пропусків у символьному рядку.

Приклад 6.1. Використання функцій видалення пропусків у рядку:

```
Con= "   Видалення пропусків   "
Print LTrim(Con) ' Повертає "Видалення пропусків   "
Print RTrim(Con) ' Повертає "   Видалення пропусків"
Print Trim(Con)  ' Повертає  "Видалення пропусків"
```

За допомогою функцій Left, Right і Mid можна виокремити підрядок із зазначеного символьного рядка. Функції Left і Right виокремлюють рядок, починаючи з крайнього лівого або крайнього правого символу, а функція Mid дає змогу обрати будь-який підрядок.

Приклад 6.2. Використання функцій виокремлення підрядків:

```
Con = "Виділення підрядка"
Print Left(Con, 3)   ' Повертає  "Вид"
Print Right(Con, 5)  ' Повертає  "рядка"
Print Mid(Con, 11,3) ' Повертає  "під"
```

Функції UCase і LCase використовують у Visual Basic з метою перетворення символів нижнього регістру в символи

верхнього регістру, і навпаки. Крім цього, у Visual Basic є функція `StrConv`, яка перетворює вираз до власного імені, що починається з великої літери.

Приклад 6.3. Використання функцій перетворення регістру символів:

```
Con= "висновок"
Print UCase(Con)  ' ВИСНОВОК
Print UCase$(Con) ' ВИСНОВОК
Con=  "іванів іван Іванович"
Print StrConv(Con, vbProperCase) ' Іванів Іван Іванович
```

Visual Basic містить дві функції, що дають змогу здійснювати пошук рядка символів у іншому рядку: `InStr` і `InStrRev`. Ці функції відрізняються тим, що `InStr` здійснює пошук з початку рядка і до його кінця, а `InStrRev` здійснює пошук у зворотному напрямі, тобто від кінця рядка до його початку. Функції повертають число, яке вказує, відповідно, номер позиції першого/останнього входження рядка пошуку у початковому рядку, або повертають 0, якщо такого входження не виявлено.

Приклад 6.4. Використання функції `InStr`:

```
Print InStr ("Сьогодні прекрасна погода", "погода") '20
```

6.2. Порівняння рядків

Порівняння рядків символів є значно складнішим, ніж порівняння чисел. Два рядка символів вважають рівними тільки тоді, коли обидва рядка містять такі ж символи у такому ж порядку і мають однакову довжину. VB враховує початкові й/або кінцеві пропуски у рядку і порівнює кожен рядок зліва направо посимвольно. Результат порівняння визначається негайно, як тільки відповідні символи не збігаються. Порівняння символів відповідає звичному алфавітному порівнянню, за яким букву `"а"` вважають меншою за букву `"б"`, і так далі.

VB дає змогу використовувати два різні способи порівняння однакових символів різних регістрів: *двійковий* і *текстовий.* За домовленістю під час порівняння рядків відповідні символи порівнюють за їхніми двійковими значеннями (кодами). У цьому випадку рядок `"CCC"` вважають меншим за рядок `"ccc"`. Оператор

```
Option Compare Text
```

на початку модуля задає алфавітне порівняння символів у рядках, що не залежить від регістру (у цьому випадку рядки "CCC" і "ccc" вважатимуть рівними).

Операція **Like** дає змогу виконати порівняння рядка з *шаблоном.* У шаблоні записують символи підстановки, списки та інтервали символів. Символи підстановки наведено у таблиці 6.2.

Таблиця 6.2. Символи підстановки

Символи шаблону	Символи у рядку, які збігаються
?	Будь-який одиничний символ
*	Будь-яка кількість символів
#	Будь-яка цифра
[список_символів]	Будь-який одиничний символ у списку_символів
[!список_символів]	Будь-який одиничний символ, який не належить до списку_символів

Приклад 6.5. Використання шаблону.

а) перевірка правильності поштового коду Канади:

```
st="W1F 8G7"  ' Поштовий код Канади – правильний
If st Like "[A-Z]#[A-Z] #[A-Z]#" Then Print "Так" Else Print "Ні"
```

б) перевірка факту, що St не починається з голосної букви:

```
If st Like "[!AEIOUaeiou]*" Then Print "Так" Else Print "Ні"
```

Діапазон необхідно вказувати від найменшого до найбільшого символу. Наприклад, [a-f] – правильний діапазон, а [f-a] – неправильний.

Оскільки ліва квадратна дужка "[", знак питання "?", символ номера "#" і символ "*" мають особливе значення у рядку шаблону, необхідно взяти їх у квадратні дужки, якщо треба, щоб вони були часиною шаблону. Наприклад, якщо необхідно дізнатися, чи закінчується рядок AnyStr знаком питання, то необхідно використати вираз:

```
AnyStr Like "*[?]"
```

Права дужка `"]"` і знак оклику `"!"` також мають особливе значення у рядку шаблону; для досягнення збігу з цими символами необхідно вміщувати їх поза квадратними дужками списку символів. Наприклад, щоб визначити, чи закінчується рядок знаком оклику, необхідно використати вираз:

```
AnyStr Like "*!"
```

Для збігу зі знаком дефіса у рядку шаблону необхідно помістити дефіс на початку або наприкінці списку символів всередині квадратних дужок. Розташування дефіса у будь-якому іншому місці задає діапазон символів. Наступний вираз демонструє, як зіставляти символ рядка з символом дефіса (цей вираз має **`True`**, якщо `AnyStr` містить `"big-headed"`, `"pig-head"`, `"plug-ugly"` тощо):

```
AnyStr Like "*g[-]*".
```

На результат порівняння рядків, яке використовує операцію **`Like`**, впливає також інструкція **`Option Compare`**. Якщо задано двійкове порівняння рядків, то оператор **`Like`** розрізняє букви верхнього і нижнього регістрів. Якщо обрано установку текстового порівняння, то операція **`Like`** не чутлива до стану регістру.

6.3. Типові задачі опрацювання рядків символів

У цьому параграфі подано розв'язання деяких типових задач опрацювання рядків символів у VB.

Приклад 6.6. Полічити кількість повторень у рядку символу `"x"`.

```
Sub main()
  Dim I As Byte, k As Byte, Z As String
  Z = InputBox("Введіть рядок")
  k = 0
  For I = 1 To Len(Z)
    If Mid(Z, I, 1) = "x" Then k = k + 1
  Next I
  MsgBox k
End Sub
```

Приклад 6.7. Замінити у рядку кожен символ "!" на символ ".".

```
Sub main()
  Dim I As Byte, Z As String
  Z = InputBox("Введіть рядок")
  For I = 1 To Len(Z)
    If Mid(Z, I, 1) = "!" Then Mid(Z, I, 1) = "."
  Next I
  MsgBox Z
End Sub
```

Приклад 6.8. Замінити у рядку кожен символ "." на "...".

```
Sub main()
  Dim I As Byte, Z As String
  Z = InputBox("Введіть рядок")
  I = 1
  While I <= Len(Z)
   If Mid$(Z, I, 1) = "." Then
     Z = Left$(Z, I - 1) & "..." & Mid$(Z, I + 1)
     I = I + 2
   End If
   I = I + 1
  Wend
  MsgBox Z
End Sub
```

Приклад 6.9. Замінити у рядку всі "..." на ".".

```
Sub main()
  Dim I As Byte, Z As String
  Z = InputBox("Введіть рядок")
  I = 1
  While I <= Len(Z)
   If Mid$(Z, I, 3) = "..." Then
     Z = Left$(Z, I - 1) & "." & Mid$(Z, I + 3)
   End If
   I = I + 1
  Wend
  MsgBox Z
End Sub
```

Приклад 6.10. Вияснити, чи трапляються у рядку символи ",-".

```
Sub main()
 Dim Z As String
 Z = InputBox("Введіть рядок")
 If InStr(Z, ",-") Then MsgBox "Так" Else MsgBox "Ні"
End Sub
```

Приклад 6.11. Відомо, що у рядку є хоча б один символ "," (кома). Знайти номер першого і останнього входження "," у рядок.

```
Sub main()
  Dim I1 As Byte, im As Byte, Z As String
  Z = InputBox("Введіть рядок")
  I1 = InStr(Z, ","): im = InStrRev(Z, ",")
  MsgBox "1 входження=" & I1
  MsgBox "Остан. входження=" & im
End Sub
```

Приклад 6.12. Полічити у рядку усі букви "b" і видалити ті з них, перед якими стоїть буква "a".

```
Sub main()
Dim I As Byte, k As Byte, Z As String, c As String
 Z = InputBox("Введіть рядок")
 c = Chr(0): Z = c + Z: k = 0
 For I = 2 To Len(Z)
  If Mid$(Z, I, 1) = "b"   Then k = k + 1
  If Mid$(Z, I-1, 1) = "a" Then Z = Left$(Z,I-1) _
        & c & Mid$(Z, I + 1)
 Next I
 I = 2
 While I <= Len(Z)
  If Mid$(Z, I, 1) = c Then
    Z = Left$(Z, I - 1) & Mid$(Z, I + 1)
  End If
  I = I + 1
 Wend
 MsgBox k : MsgBox Z
End Sub
```

Приклад 6.13. Вияснити, чи рядок Z є паліндромом (без урахування пропусків).

```
Sub main()
 Dim I As Byte, Z As String
 Z = InputBox("Введіть рядок")
 Z = Trim(Z)
 For I = 1 To Len(Z)
  If Mid$(Z, I, 1) = " " Then _
      Z = Left$(Z, I - 1) & Mid$(Z, I + 1)
  Next I
 If Z = StrReverse(Z) Then MsgBox "Паліндром" _
                      Else MsgBox "Не паліндром"
End Sub
```

6.4. Побудова підпрограм роботи з рядками символів

Познайомившись з базовими вбудованими функціями роботи з рядками символів, можна використати їх для розв'язання дещо складніших задач. Виберемо декілька таких задач, які часто виникають при аналізі тексту. Розв'язок цих задач оформимо у вигляді підпрограм (процедур чи функцій).

Приклад 6.14. Функція CountInStr(strText, strFind, fCase) визначає кількість повторень підрядка strFind у рядку strText із заданим способом порівняння символів fCase (за домовленістю встановлено алфавітне порівняння символів).

```
Function CountInStr(strText As String, strFind As String _
 Optional ByVal fCase As Boolean = False) As Integer
 ' Визначення кількості повторень підрядка strFind
 ' у рядку strText; fCase задає спосіб порівняння
 Dim intCount As Integer ' Ініціалізація нулем (0)
 Dim intPos As Integer, intCase As Integer
 If Len(strFind) = 0 Then ' Підрядок не задано
    CountInStr = 0
    Exit Function
 End If
```

```
 ' Встановлення способу порівняння
 If fCase Then
            intCase = vbBinaryCompare
 Else
            intCase = vbTextCompare
 End If
 intPos = 1
 Do
   intPos = InStr(intPos, strText, strFind, intCase)
   If intPos > 0 Then
                intCount = intCount + 1
                intPos = intPos + Len(strFind)
   End If
 Loop While intPos > 0
 CountInStr = intCount
End Function
```

Для тестування підпрограми перелічимо кількість букв "а", "б" (незалежно від регістру) і великої букви "В" у заданому тексті.

```
Public Sub main()
Dim k As Byte, Z As String
  Z = InputBox("Введіть рядок")
  k = CountInStr(Z, "а") + CountInStr(Z, "б") _
                         + CountInStr(Z, "В", True)
  MsgBox k
End Sub
```

У мові Паскаль є спеціальна процедура Delete(s, n, count), яка видаляє з рядка s фрагмент тексту, який починається у позиції n і має довжину count. Інша процедура Insert(s, s1, n) вставляє у рядок s1 підрядок s, починаючи з заданої позиції n. Запрограмуємо ці процедури у Visual Basic.

Приклад 6.15. Процедура Delete(s, n, count) видаляє з рядка s фрагмент тексту, який починається з позиції n і має довжину count.

```
Public Sub Delete(s As String, ByVal n As Integer, _
                                ByVal count As Integer)
 ' Видаляє з рядка s фрагмент тексту, який
 ' починається у позиції n і має довжину count
 If Len(s) = 0 Or n <= 0 Or count <= 0 Then Exit Sub
```

```
 s = Left$(s, n - 1) & Mid$(s, n + count)
End Sub
```

Тестова програма:

```
Public Sub main()
Dim Z As String, i As Integer, m As Integer
 Z = InputBox("Введіть рядок")
 m = InputBox("Введіть позицію видалення ")
 j = InputBox("Введіть кількість символів видалення")
 Call Delete(Z, m, j) :  MsgBox Z
End Sub
```

Приклад 6.16. Процедура `Insert(s, s1, n)` вставляє у рядок `s1` підрядок `s`, починаючи з заданої позиції `n`.

```
Public Sub Insert(s As String, s1 As String, _
                                  ByVal n As Integer)
 ' Вставляє у рядок s1 підрядок s,
 ' починаючи з заданої позиції n
 If Len(s) = 0 Or Len(s1) = 0 Or n <= 0 Then Exit Sub
 s1 = Left$(s1, n - 1) & s & Mid$(s1, n)
End Sub
```

Тестова програма:

```
Public Sub main()
Dim Z As String, Z1 As String, m As Integer
   Z1 = InputBox("Введіть рядок")
   Z = InputBox("Введіть підрядок")
   m = InputBox("Введіть позицію вставки")
  Call Insert(Z, Z1, m) :  MsgBox Z1
End Sub
```

Приклад 6.17. Функція

```
Transl(strIn, strMapIn, strMapOut, fCase)
```

символам рядка `strMapIn` ставить у відповідність почергово символи рядка `strMapOut` і здійснює заміну символів у рядку `strIn` згідно з цією відповідністю. Рядок `strMapIn` містить символи, які необхідно знайти і замінити.

Рядок strMapOut містить символи заміни. Якщо довжина цього рядка є меншою за довжину strMapIn, то крайній справа символ рядка strMapOut повторюється функцією необхідну кількість разів для вирівнювання довжин обидвох рядків. Наприклад: виклик функції Transl(Z, "FGHJ", "W") еквівалентний виклику Transl(Z, "FGHJ", "WWWW").

Якщо список strMapOut є порожнім, то у рядку strIn знищуються символи, які перелічені у списку strMapIn. Наприклад, при виклику Transl(Z, "()", "") у рядку Z будуть видалені круглі дужки. Параметр fCase задає спосіб порівняння символів: **True** – двійкове порівняння (за домовленістю); **False** – текстове порівняння.

```
Function Transl(ByVal strIn As String, _
ByVal strMapIn As String, ByVal strMapOut As String, _
Optional fCase As Boolean = True) As String
 Dim i As Integer ' Параметр циклу
 Dim intMode As Integer ' Спосіб порівняння символів
 Dim intPos As Integer ' Знайдена позиція
 Dim strChar As String * 1 ' Черговий символ
 Dim strOut As String ' Рядок - результат роботи.
 ' Якщо список символів, які необхідно  замінювати, є
 ' порожнім, то виходимо з підпрограми.
 If Len(strMapIn) = 0 Then
    Transl = strIn: Exit Function
 End If
 ' Встановлення способу порівняння символів.
 If fCase Then
        intMode = vbBinaryCompare
 Else
        intMode = vbTextCompare
 End If
 ' Доповнення справа рядка strMapOut.
 If Len(strMapOut) > 0 Then
  strMapOut=Left$(strMapOut & String(Len(strMapIn), _
               Right$(strMapOut, 1)), Len(strMapIn))
 End If
```

```
 For i = 1 To Len(strIn)
   strChar = Mid$(strIn, i, 1)
   intPos = InStr(1, strMapIn, strChar, intMode)
   If intPos > 0 Then
    ' Якщо strMapOut є порожнім, то це не буде помил-
    'кою, адже Mid коректно опрацьовує порожні рядки.
     strOut = strOut & Mid$(strMapOut, intPos, 1)
   Else
     strOut = strOut & strChar
   End If
 Next i
 Transl = strOut
End Function
```

Тестова програма:

```
Public Sub main()
Dim Z As String, Z1 As String, Z2 As String
 Z = InputBox("Введіть рядок")
 Z1 = InputBox("Введіть рядок символів, що замінюються")
 Z2 = InputBox("Введіть рядок символів заміни")
 Z = Transl(Z, Z1, Z2) :  MsgBox Z
End Sub
```

Приклад 6.18. Функція `Alltrim(strIn, fTabs)` видаляє пропуски на початку і наприкінці рядка `strIn` і замінює будь-яку кількість пропусків (і, можливо, символів табуляції) всередині цього рядка одним пропуском. Параметр `fTabs` задає реакцію функції на символ табуляції: **`True`** – видаляти (за домовленістю); **`False`** – не видаляти.

```
Function Alltrim(ByVal strIn As String, Optional _
                 fTabs As Boolean = True) As String
 Dim i As Integer ' Параметр циклу
 Dim strChar As String * 1 ' Черговий символ
 Dim strTemp As String ' Допоміжний рядок
 Dim strOut As String  ' Рядок - результат роботи.
 ' Встановлення  реакції на символ табуляції.
 If fTabs Then
    strIn = Transl(strIn, vbTab, " ")
 End If
 strTemp = Trim(strIn)
```

```
 For i = 1 To Len(strTemp)
  strChar = Mid$(strTemp, i, 1)
  If Not (strChar=" " And Right$(strOut, 1)=" ") Then
     strOut = strOut & strChar
  End If
 Next i
 Alltrim = strOut
End Function
```

Тестова програма:

```
Public Sub main()
Dim Z As String
 Z = Alltrim(InputBox("Введіть рядок")) :  MsgBox Z
End Sub
```

? Запитання для самоперевірки

1. Що робить функція Len?
2. Що робить функція Mid?
3. Що робить функція Pos?
4. Що робить функція InStr?
5. Що робить функція InStrRev?
6. Назвіть і охарактеризуйте функції видалення пропусків.
7. Назвіть і охарактеризуйте функції виділення підрядків.
8. Назвіть і охарактеризуйте функції зміни регістру букв.
9. Як порівнюються рядки символів?
10. Як виконується операція **Like**?
11. Що таке шаблон? Які Ви знаєте символи шаблона?
12. Що означає символ "$" у назві функції?

💻 Завдання для програмування

Завдання 6.1. Скласти програми для розв'язання таких задач:

1. Відомо, що у рядку є хоча б один символ "*" (зірочка). Знайти номер першого входження "*" у рядок і полічити кількість усіх входжень "*" у рядок (величина *n*). Добавити символ "*" у кінець рядка, повторити його *n* разів.
2. Відомо, що у рядку є хоча б один символ "*" (зірочка). Знайти номер останнього входження "*"у рядок і полічити кількість усіх входжень "*" у рядок (величина *n*). Добавити символ "*" на початок рядка, повторити його *n* разів.

3. Відомо, що у рядку є букви й цифри. Перетворити рядок так, щоб спочатку розміщувалися букви, а потім цифри. Порядок символів має зберігатися.
4. Відомо, що у рядку є букви й цифри. Перетворити рядок так, щоб спочатку розміщувалися букви у прямому порядку, а потім цифри у зворотному порядку.
5. Відомо, що у рядку є букви й цифри. Перетворити рядок так, щоб спочатку розміщувалися цифри у прямому порядку, а потім букви у зворотному порядку.
6. Прочитати з клавіатури два рядки символів. Якщо вони однакові (за символами, пропуски не рахуються), то надрукувати слово "Так", інакше – "Ні".
7. Прочитати з клавіатури рядок символів. Ввести шаблон і вибрати з рядка усі слова, що відповідають шаблону. *Слова* у тексті рядка розділяються пропусками.
8. Нехай цифрам від 1 до 9 відповідають латинські букви від A(a) до I(i). Прочитати з клавіатури рядок символів. Скласти новий рядок з цифр, які відповідають тільки зазначеним буквам. Рядок-результат впорядкувати за неспаданням.
9. Прочитати з клавіатури рядок символів. Вибрати з нього латинські букви від A(a) до I(i), перетворити малі букви у великі, і впорядкувати їх в алфавітному порядку.
10. Кожен символ "+" у рядку замінити на символ "–", якщо перед "+" стоїть непарна цифра.
11. Прочитати з клавіатури рядок символів. Скопіювати його, замінивши водночас букви "R", "S", "T" на "K", "L", "M", відповідно.
12. Прочитати з клавіатури рядок символів. Полічити в ньому окремо символи "(" та ")". При неспівпаданні кількості повторень добавити необхідний символ наприкінці рядка, повторивши його потрібну кількість разів.
13. Перелічити у рядку усі послідовності символів "cd" і видалити ті з них, перед якими стоїть буква "b".
14. Перелічити у рядку усі послідовності символів "cd" і видалити ті з них, після яких стоїть буква "b".
15. Прочитати з клавіатури слово і побудувати всі його *анаграми* (усеможливі буквосполучення, які складаються з букв заданого слова).

Завдання 6.2.

Термінологія. *Текст* – довільна послідовність символів в одному рядку (**`string`**). *Слова* у тексті розділяються пропусками. *Речення* – послідовність слів, що завершується крапкою (знаком оклику чи запитання). Скласти підпрограми (і протестувати) розв'язання таких задач:

1. Визначити у тексті максимальну довжину послідовності символів, що не є літерами.
2. Знайти максимальну за довжиною монотонну (неспадну або незростаючу) підпослідовність натуральних чисел, заданих у тексті так, що числа відокремлюються пропусками.
3. Визначити всі слова тексту, що складаються з тих самих літер, що й перше слово цього тексту.
4. У тексті знайти пари слів, що є дзеркальними відображеннями одне одного (наприклад `"ole"` і `"elo"`).
5. У тексті знайти всі симетричні слова (наприклад, `"oko"`, `"ABBA"`).
6. Знайти всі слова, що трапляються у кожному з двох речень тексту.
7. Відредагувати деякий текст, видаливши у ньому всі непарні за порядком слова і перевертаючи парні за порядком слова (наприклад, `"How do you do? "` ⇒ `"od od? "`).
8. Для кожного символу тексту зазначити, скільки разів він зустрічається у тексті. Повідомлення про символ виводити тільки один раз.
9. Деякий текст має два речення. Знайти найкоротше слово першого рядка, якого нема у другому реченні.
10. Відредагувати деякий текст, видаливши у ньому всі слова, які налічують входження тільки двох однакових літер (наприклад, `"Akka ababa cece knopka. "` ⇒ `"knopka. "`).
11. Замінити в англомовному тексті закінчення слів `"ing"` на `"ed"`, стиснувши водночас текст.
12. В англомовному тексті знайти слово, у якому кількість голосних літер (*a*, *e*, *i*, *o, u*) є найбільшою.
13. Характеристика слова – кількість *різних* символів, що у нього входять. Впорядкувати слова у тесті за спаданням їхніх характеристик.
14. Відстань між двома словами *однакової* довжини – це кількість позицій, у яких стоять різні символи. В деякому тексті знайти пару найвіддаленіших слів заданої довжини.
15. Визначити всі слова тексту, що складаються з тих самих літер, що й останнє слово цього тексту.

7. ФАЙЛИ

План викладу матеріалу

1. Загальні положення
2. Типи файлів
3. Відкриття і закриття файлів
4. Функції роботи з файлами
5. Робота з файлами послідовного доступу
6. Робота з двійковими файлами
7. Робота з файлами довільного доступу

Ключові терміни розділу

- *Означення файлу*
- *Канал введення/виведення*
- *Файл послідовного доступу*
- *Двійковий файл*
- *Оператор* `Close`
- *Оператор* `Input #`
- *Оператор* `Seek`
- *Оператор* `Write #`
- *Оператор* `Put #`
- *Операції введення/виведення*
- *Дескриптор файлу*
- *Файл довільного доступу*
- *Оператор* `Open`
- *Функція* `Input`
- *Оператор* `Line Input #`
- *Оператор* `Print #`
- *Оператор* `Get #`
- *Функції*: `EOF`, `LOF`, `Seek`, `Loc`

7.1. Загальні положення

Файл визначає впорядковану сукупність довільного числа однотипних *елементів*. Як відомо, кількість елементів масиву визначають на етапі написання програми. Кількість елементів файлу в тексті програми не визначають (може бути довільною).

При роботі з файлами найчастіше виконують операції введення/виведення даних. Операція **введення** (або **читання**) означає пересилання даних з вхідного файлу в основну пам'ять комп'ютера; операція **виведення** (або **запису**) – пересилання даних з основної пам'яті у вихідний файл.

Способи задання назв файлів визначаються операційною системою. В операційній системі Windows у програмах VB назви файлів задають за допомогою рядків спеціального вигляду, у яких, крім назви файлу, записують ще й шлях до місцезнаходження фай-

лу на зовнішньому пристрої (зазвичай диску). Наприклад, назва файлу може мати вигляд:

```
"d:\LAB1.DAT"
"c:\ABC150\pr.bas"
```

З файловою системою зв'язане поняття **каналу** введення/виведення. Під час відкриття кожному файлу ставиться у відповідність канал з певним номером. Через цей канал і здійснюється обмін даними між оперативною пам'яттю і файлом на зовнішньому пристої.

Отже, для обміну даними між між оперативною пам'яттю і файлом програма має знати номер каналу (або *дескриптор файлу*). У Visual Basic є вбудована функція `FreeFile`, яка повертає номер вільного каналу, який можна використати для роботи з файлом. Синтаксис функції:

```
FreeFile([RangeNumber])
```

Необов'язковий параметр `RangeNumber` дозволяє визначити діапазон значень, з якого вибирається черговий вільний номер каналу. Якщо його значення дорівнює 0 (за домовленістю), то повертається номер каналу з діапазону 1 – 255, якщо 1, то з діапазону 256 – 511. Якщо вільних каналів немає, виникає помилка виконання.

7.2. Типи файлів

У Visual Basic існує поняття **типу** файла, який визначається організаційною структурою зберігання інформації в файлі і способом доступу до цієї інформації. Виділяють наступні типи файлів:

- *файл послідовного доступу* містить набір символів, який може бути з роздільниками (файл має визначену структуру) або без роздільників (структурною одиницею є рядок). Прикладами цих файлів є текстові файли і файли ініціалізації програм;
- *файл довільного доступу* містить інформацію у вигляді записів;
- *двійковий* (або *бінарний*) *файл* містить набір двійкових кодів (байтів). У принципі, це ті ж файли з послідовним доступом, але інформація у них не розбита на рядки. Особливість даних файлів – робота з байтами або блоками байтів. До таких файлів можна віднести файли виконання, файли динамічних бібліотек, файли документів Word тощо.

Подібний розподіл файлів на типи є досить умовним. Наприклад, файл послідовного доступу можна відкрити і у режимі двійкового доступу. Якщо цей файл має роздільники, то для роботи з ними доведеться написати спеціальну процедуру опрацювання роздільників і виділення даних, оскільки двійковий доступ забезпечує тільки побайтовий обмін даними з файлом. Очевидно, що це незручно. Саме тому введено умовний розподіл файлів на типи у залежності від формату файла і доступу до даних. Відповідно згруповані і засоби Visual Basic (оператори і функції) для обміну даними.

Зауваження. Ймовірно, більш коректним було би говорити не "файл послідовного доступу", а "файл, відкритий для послідовного доступу", не "двійковий файл", а "файл, відкритий для двійкового доступу". Однак, надалі будемо дотримуватися термінології, яка вказує тип файла (*файл послідовного доступу*; *файл довільного доступу*; *двійковий файл*).

Тип файла задає оптимальний набір функцій запису і читання даних з файла. Тому при роботі з файлами для написання ефективної програми завжди необхідно мати уявлення про типи файлів, з якими буде працювати програма, і про організацію зберігання даних у цих файлах. Це дає можливість забезпечити оптимальний доступ і використати відповідні цьому доступу оператори і функції.

Робота з файлами полягає у послідовному виконанні таких дій:

1. Отримання (за потребою) дескриптора файла (*handler*).
2. Відкриття файла.
3. Обмін даними (читання/запис) з файлом.
4. Закриття файла.

7.3. Відкриття і закриття файлів

Робота з кожним типом файлу має свої особливості. Однак є дві дії, загальні для всіх типів файлів – їхнє відкриття і закриття. Відкриття файла виконується оператором:

```
Open pathName For mode [Access access] [lock] _
                As [#]fileNumber [Len=recLength]
```

де:

- `PathName` – повна назва файлу (з вказівкою шляху);
- `mode` – режим *відкриття* файлу. Може приймати такі значення:

`Append` – відкриття для додавання нових даних у файл;

`Binary` – відкриття двійкових файлів;

`Input` – відкриття тільки для читання даних з файлу;

`Output` – відкриття для запису нових даних у файл;

`Random` – відкриття файлів довільного доступу. Розмір запису визначається значенням `recLength`;

- `access` – режим *доступу* до двійкових файлів і файлів довільного доступу. Може приймати такі значення:

  `Read` – тільки для читання;

  `Write` – тільки для читання;

  `Read Write` – читання/запис (за домовленістю);

- `lock` – режим *використання* файлу. Визначає можливість одночасної роботи з файлом декількох програм або декількох користувачів. Може приймати такі значення:

  `Shared` – доступний у всіх режимах (за домовленістю);

  `Lock Read` – блокування читання;

  `Lock Write` – блокування запису;

  `Lock Read Write` – блокування читання і запису.

- `fileNumber` – цілочисельний вираз, що задає ідентифікатор файла (*дескриптор*). Може мати значення від 1 до 511;

- `recLength` – число, що визначає розмір буфера даних для запису/читання у файлах послідовного доступу. Для файлів довільного доступу це число задає довжину одного запису файла. Може мати значення до 32767 байтів.

Якщо вказаний у операторі `Open` файл не знайдено або не існує, то він буде створений для усіх режимів доступу за виключенням режиму `Input`.

Оператор **закриття** файлів має наступний синтаксис:

```
Close [список_дескрипторів_файлів]
```

де `список_дескрипторів_файлів` містить номери файлів (перераховуються через кому), що закриваються. При цьому номер файлу повинен відповідати номеру файлу в функції `Open`. Якщо номери файлів не вказано, то закриваються всі файли.

7.4. Функції роботи з файлами

У Visual Basic є великий набір функцій, які призначені для опрацювання файлів. Найбільш вживані функції перераховані у таблиці 7.1, де введені такі позначення:

- `descr` – дескриптор (номер), який отримує файл при відкритті;
- `path` – повна назва файлу (зі шляхом);
- `num` – кількість символів (байтів);

Таблиця 7.1. Функції роботи з файлами

Функція	Дія
`EOF(descr)`	Повертає `True`, якщо досягнутий кінець файлу, а інакше – `False`. Використовується зазвичай при послідовному доступі. При довільному чи двійковому доступі повертає `True`, якщо оператор `Get` не зміг прочитати цілий запис, а інакше – `False`
`FileDateTime(path)`	Повертає рядок з датою і часом створення чи останньої модифікації файлу
`FileLen(path)`	Повертає розмір файлу (закритого) у байтах
`FreeFile([RangeN])`	Повертає номер вільного каналу, який можна використати для встановлення дескриптора файлу. Параметр `RangeN` визначає діапазон значень, з якого вибирається черговий вільний номер каналу: 0 (за домовленістю) – повертається номер каналу з діапазону 1 – 255; 1 – з діапазону 256 – 511
`Input(num, [#]descr)`	Повертає `num` символів з файлу (`descr`), який відкрито у режимі `Input` або `Binary`. На відміну від оператора `Input #`, функція повертає **усі** символи, включаючи коми, символи переведення каретки і нового рядка, пропуски
`LOC(descr)`	При *довільному* доступі повертає номер останнього запису, до якого здійснювався доступ. При *двійковому* доступі повертає номер останнього байту (символу), до якого здійсню-

	вався доступ. При *послідовному* доступі функція не використовується
`LOF(descr)`	Повертає розмір відкритого файлу в байтах, включаючи символ-ознаку закінчення файлу
`Seek(descr)`	При *довільному* доступі повертає номер запису, до якого буде здійснюватися доступ. При *двійковому* або *послідовному* доступі повертає номер байту (символу), до якого буде здійснюватися доступ (нумерація з одиниці)

7.5. Робота з файлами послідовного доступу

Файли послідовного доступу – це, як правило, текстові файли, тобто послідовності символів, які розбиті на рядки. У тексті може знаходитися символ переведення рядка (`vbCrLf` або `chr(13) & Chr(10)`) і сивол табуляції (`vbTab` або `chr (9)`). Ці символи використовують для форматування тексту. Спосіб відкриття файлу з послідовним доступом (для читання, запису або додавання) задається при відкритті файлу:

```
Open pathName For [Input | Output | Append] _
                  As [#] fileNumber
```

Якщо файл не існує і відкривається для читання (**Input**), то Visual Basic видає повідомлення про помилку, а якщо для запису або додавання (**Output** або **Append**), то створюється новий файл. Якщо файл з вказаною назвою існує, то у режимі **Output** його вміст видаляється, а у режимі **Append** файл відкривається для додавання нових даних. Приклади відкриття файлів послідовного доступу:

```
Open "C:\Myd.txt" For Input As #1
Open "C:\temp\data1.dat" For Output As #2
Open "C:\data2.txt" For Append As #3
```

Читання даних з файлу послідовного доступу виконуються за допомогою функції **Input** і операторів **Input #** і **Line Input #**. Функція **Input** читає з файлу задану кількість символів і використовується для читання даних, які записані у файл оператором **Print #**. Якщо у програмі потрібно прочитати дані з файлу, в якому інформація у рядках має структуру з роздільниками (**комами**), необхідно застосувати оператор:

Input #fileNumber, varlist

де:

- fileNumber – номер файлу (дескриптор);
- varlist – список змінних.

При роботі цього оператора спочатку прочитується цілий рядок, а потім підрядки, відокремлені комами (роздільниками), вміщуються у відповідні змінні списку. Для коректної роботи оператора рядки файлу повинні мати задану структуру з роздільниками. Зазвичай, цей оператор використовується у парі з оператором запису **Write** #.

Для читання рядків з файлу послідовного доступу застосовують оператор

Line Input #fileNumber, varName

де:

- fileNumber – номер файлу (дескриптор);
- varName – назва змінної типу String.

Оператор **Line Input** # читає черговий рядок файлу і передає його у змінну varName. При цьому роздільником рядків у файлі служить стандартний роздільник рядків – символ повернення каретки CHR(13) або послідовність символів повернення каретки і переведення рядка CHR(13) + CHR(10), причому в змінну varName ці роздільники не вставляються. Оператор **Line Input** # використовується зазвичай у парі з оператором **Print** #.

Для того щоб прочитати всі дані з файла за допомогою оператора **Input** # або **Line Input** # необхідно організувати цикл читання даних.

Перехід на задану позицію у файлі реалізує оператор

Seek #fileNumber, position

де position – цілочисельний вираз, що задає позицію покажчика у файлі. Позиціонування при цьому виконується посимвольно (нумерація починається з одиниці). Оператор **Seek встановлює** покажчик на необхідний символ. Якщо після цього використати оператори читання або запису, то вони будуть виконуватися, починаючи з позиції покажчика, встановленого оператором **Seek**.

При введенні/виведенні даних у файл послідовного доступу переміщення покажчика на наступну позицію (символ/символи або рядок) відбувається автоматично.

Дані у файл послідовного доступу записують операторами **Print** # і **Write** #. Оператори автоматично вставляють у файл роздільники. Кожний з операторів запису працює у парі з певним засобом читання. Для оператора **Print** # це функція **Input** або оператор **Line Input** #, а для оператора **Write** # – оператор **Input** #. Оператори запису даних у файл мають такий синтаксис:

```
Print #fileNunber, [outputlist]
Write #fileNumber, [outputlist]
```

де `outputlist` – список виразів або змінних, значення яких записуються у файл.

Оператор **Print** # функціонує майже так само, як і оператор **Print** для екрана, з тією лише різницею, що дані виводяться не на екран, а зберігаються у файлі, відкритому для запису або додавання (**Open** ... **For Output** або **Open** ... **For Append**).

Для форматування інформації, що записується в файл треба по-різному розділяти дані в операторі **Print**. Якщо в операторі дані розділяти комами (**","**), то у файлі вони будуть розділені символами табуляції. Якщо ж в операторі для розділення даних використати крапку з комою (**";"**), то дані у файл записуються без роздільників.

Оператор **Write** # має такий же синтаксис, що й **Print** #. Різниця полягає у форматуванні виведення. Якщо **Print** # зберігає усі дані у вигляді звичайного *тексту*, то **Write** # рядки символів обрамлює лапками, а числа виводяться без лапок.

Оператор **Write** # після кожного рядка автоматично вставляє символи переведення каретки `CHR(13)` і нового рядка `CHR(10)`. Елементи файлу, записані оператором **Write** #, розділяються *комами*. Пару операторів **Write** # / **Input** # зазвичай застосовують у випадках опрацювання числових даних.

Приклад 7.1. Дані табулювання квадратичної функції зберегти у текстовому файлі. Отриманий файл використати для виведення цих

даних у вікно негайного виконання і обчислення середнього арифметичного значень функції.

```
Public Sub main()
Dim descr As Byte, i As Byte, n As Byte
Dim x As Single, y As Single
Dim s As Single
Open "c:\temp\data1.dat" For Output As #1
x = -2
For i = 1 To 10
  y = x * x - 3 * x + 1
  Write #1, x, y
  x = x + 0.2
Next i
Close #1
descr = FreeFile
Open "c:\temp\data1.dat" For Input As descr
While Not EOF(descr)
  Input #descr, x, y
  Debug.Print FormatNumber(x, 2), FormatNumber(y, 4)
  s = s + y  :  n = n + 1
Wend
  s = s / n
  MsgBox "Середнє арифметичне=" & s
Close descr
End Sub
```

Приклад 7.2. Вивести у вікно негайного виконання вмістиме файлу `"c:\autoexec.bat"` і обчислити кількість символів `":"` у цьому файлі, використавши послідовний доступ.

```
Public Sub main()
Dim s As String, s1 As String * 1, n As Byte
Open "c:\autoexec.bat" For Input As #1
While Not EOF(1)
   Line Input #1, s
   Debug.Print s
Wend
Seek #1, 1 ' Установка покажчика на 1-й символ
```

```
While Not EOF(1)
   s1 = Input(1, #1)
   If s1 = ":" Then n = n + 1
Wend
MsgBox n
Close #1
End Sub
```

7.6. Робота з двійковими файлами

Нагадаємо, що функція `Input(num, [#]descr)` повертає `num` символів з файлу (`descr`), який відкрито у режимі `Input` або `Binary` (двійковий доступ). У зв'язку з цим переробимо попередній приклад, використавши двійковий доступ.

Приклад 7.3. Вивести у вікно негайного виконання вмістиме файлу `"c:\autoexec.bat"` і обчислити кількість символів `":"` у цьому файлі, використавши двійковий доступ.

Для розв'язання поставлених задач можна обійтися одним циклом. Але для кращого розуміння матеріалу, у програмі організуємо два окремі цикли:

- цикл виведення вмістимого файлу;
- цикл підрахунку кількості символів `":"`.

```
Public Sub main()
 Dim s1 As String * 1, n As Byte
 Open "c:\autoexec.bat" For Binary Access Read As #1
 While Not EOF(1)
   Debug.Print Input(1, #1);
 Wend
 Seek #1, 1 ' Перехід на початок файлу
 While Not EOF(1)
   s1 = Input(1, #1)
   If s1 = ":" Then n = n + 1
 Wend
 MsgBox n
 Close #1
 End Sub
```

Для читання/запису окремих байтів (символів) при двійковому доступі до файлу є спеціальні оператори:

Get #номер_файлу, номер_байту, змінна

–читає байт вказаного номера з файлу, що має #номер_файлу, у змінну;

Put #номер_файлу, номер_байту, змінна

–записує значення змінної у байт вказаного номера з файлу, що має #номер_файлу.

Приклад 7.4. Створити файл, який містить деяку послідовність символів (без пропусків), а потім посортувати файл згідно з алфавітом.

```
Public Sub main()
Dim x As String * 1, y As String * 1
Dim i As Integer, j As Integer
Open "c:\Temp\test.dat" For Binary Access Write As #1
For i = 1 To 15
  x = InputBox("Введіть символ")
  Put #1, i, x
Next i
Close #1
Open "c:\Temp\test.dat" For Binary As #2
Debug.Print "Початковий рядок:"
For i = 1 To LOF(2) - 1 ' Без ознаки закінчення
  Get #2, i, x
  Debug.Print x;
Next i
Debug.Print ' Перехід на наступний рядок
For i = LOF(2) - 1 To 2 Step -1
  For j = 1 To i - 1
     Get #2, j, x: Get #2, j + 1, y
     If x > y Then Put #2, j, y: Put #2, j + 1, x
  Next j
Next i
Debug.Print "Відсортований рядок:"
For i = 1 To LOF(2) - 1
  Get #2, i, x : Debug.Print x;
Next i
```

```
Debug.Print
Close #2
End Sub
```

7.7. Робота з файлами довільного доступу

Вмістиме файлу довільного доступу розглядається як набір *записів* однакового розміру, а самий запис асоціюється з типом даних користувача (**Type … End Type**).

Для читання/запису окремих записів при довільному доступі до файлу є спеціальні оператори:

Get #номер_файлу, номер_запису, змінна

–читає запис вказаного номера з файлу, що має #номер_файлу, у змінну типу **Type**, визначеного користувачем;

Put #номер_файлу, номер_запису, змінна

–записує значення змінної типу **Type**, визначеного користувачем, у запис вказаного номера з файлу, що має #номер_файлу.

Приклад 7.5. Створити файл, який містить деяку послідовність записів про робітників. Кожний запис містить індивідуальний номер та прізвище робітника. Після цього посортувати файл за зростанням індивідуальних номерів.

```
Type Person
 id As Integer
 name As String * 10
End Type

Public Sub main()
Dim x As Person, y As Person
Dim i As Integer, j As Integer, m As Integer
If FileLen("c:\Temp\test1.dat") > 0 Then Kill _
                                  "c:\Temp\test1.dat"
Open "c:\Temp\test1.dat" For Random As #1 Len = Len(x)
For i = 1 To 10
  x.id = InputBox("Введіть номер")
  x.name = InputBox("Введіть прізвище")
  Put #1, i, x
Next i
```

```
Debug.Print "Початковий список:"
m = LOF(1) \ Len(x)
For i = 1 To m
  Get #1, i, x
  Debug.Print x.id, x.name
Next i
Debug.Print
For i = m To 2 Step -1
  For j = 1 To i - 1
   Get #1, j, x: Get #1, j + 1, y
   If x.id > y.id Then Put #1, j, y : Put #1, j+1, x
  Next j
Next i
Debug.Print "Впорядкований список:"
For i = 1 To m
  Get #1, i, x
  Debug.Print x.id, x.name
Next i
Debug.Print
Close #1
End Sub
```

? Запитання для самоперевірки

1. Що таке файл послідовного доступу?
2. Що таке дійковий файл?
3. Що таке файл довільного доступу?
4. Опишіть операції введення/виведення даних для файлів.
5. Що таке дескриптор файлу?
6. Для чого використовується оператор `Open`?
7. Для чого використовується оператор `Close`?
8. Для чого використовується функція `Input`?
9. Для чого використовується функція `EOF`?
10. Для чого використовується функція `LOF`?
11. Для чого використовується функція `Seek`?
12. Для чого використовується функція `Loc`?
13. Для чого використовується оператор `Input #`?

14. Для чого використовується оператор `Line Input #`?
15. Для чого використовується оператор `Seek`?
16. Для чого використовується оператор `Print #`?
17. Для чого використовується оператор `Write #`?
18. Для чого використовується оператор `Get #`?
19. Для чого використовується оператор `Put #`?

💻 Завдання для програмування

Завдання 7.1. *Текст* – довільна *послідовність символів*, що знаходиться у деякому двійковому файлі. *Слова* у тексті розділяються пропусками. *Речення* – послідовність слів, що завершується крапкою (знаком оклику чи запитання). Написати програму, яка виконує такі дії:

1. Читає з файла три послідовні речення (порядковий номер першого речення задається з клавіатури) і виводить їх на екран у зворотньому порядку.
2. Читає текст з файла і виводить на екран тількі ті речення, що містять задане слово.
3. Читає текст з файла і виводить на екран тількі ті речення, які містять двозначні числа.
4. Читає текст з файла і виводить на екран слова, що починаються з голосної букви.
5. Читає текст з файла і виводить його на екран, змінюючи місцями кожні два сусідні слова.
6. Читає текст з файла і виводить на екран тільки ті речення, які не містять ком.
7. Читає текст з файла і визначає, скільки у ньому слів, які мають не більше чотирьох букв.
8. Читає текст з файла і виводить на екран тільки *цитати* – послідовність слів у лапках.
9. Читає текст з файла і виводить на екран тільки ті речення, які мають визначену кількість слів.
10. Читає текст з файла і виводить його на екран, замінюючи кожну першу букву слова на цю ж букву у верхньому регістрі (велику букву).
11. Читає текст з файла, знаходить найкоротше слово і визначає скільки разів воно зустрічається у тексті.

12. Читає текст з файла і виводить на екран спочатку запитання, а потім речення зі знаком оклику.
13. Читає текст з файла і виводить на екран спочатку однобуквенні слова, а потім решту слів.
14. Читає текст з файла, знаходить найдовше слово і визначає скільки разів воно зустрічається у тексті.
15. Читає текст з файла і виводить на екран тільки ті речення, які мають максимальну кількість знаків пунктуації.

Завдання 7.2. Нехай маємо деякий *текстовий файл.* Скласти і протестувати підпрограму (процедуру або функцію), яка виконує таку дію:

1. Підраховує кількість порожних рядків.
2. Знаходить максимальну довжину рядка.
3. Знаходить мінімальну довжину рядка.
4. Підраховує кількість рядків, що починаються з голосної букви.
5. Підраховує кількість рядків, що закінчуються голосною буквою.
6. Підраховує кількість рядків, що починаються зі заданої букви.
7. Підраховує кількість рядків, що закінчуються заданою буквою.
8. Підраховує кількість рядків, що починаються і закінчуються заданою буквою.
9. У тестовому файлі записана непорожна послідовність цілих чисел, які відокремлені пропусками. Записати функцію для підрахунку суми усіх елементів файла.
10. У тестовому файлі записана непорожна послідовність дійсних чисел, які відокремлені пропусками. Записати функцію для підрахунку добутку усіх елементів файла.
11. Підраховує кількість рядків, що містять двозначні числа.
12. Підраховує кількість рядків, що містять однобуквенні слова.
13. Друкує на екрані окремими рядками вмістиме файла у зворотному порядку.
14. Переписує вмістиме файла у інший текстовий файл зі збереженням ділення на рядки, але без порожних рядків.
15. Переписує вмістиме файла у інший текстовий файл зі збереженням ділення на рядки, але у новому файлі всі рядки повинні мати фіксовану довжину (лишні символи видаляються, короткі рядки доповнюються пропусками).

Завдання 7.3. У файлі довільного доступу записати деяку непорожню *послідовність дійсних чисел*. Створити програму для розв'язання таких задач:

1. Знайти серед цих чисел найменше та номер його першого входження у файл.
2. Знайти серед цих чисел найбільше та номер його останнього вхоження у файл.
3. Обчислити суму всіх від'ємних значень.
4. Визначити кількість додатних, від'ємних і нульових значень.
5. Обчислити суму чисел, що належать проміжку [–3; 5].
6. Визначити відсоток додатних чисел.
7. Знайти середнє арифметичне чисел, розташованих за порядком за першим ненульовим. Вважати, що обов'язково є хоча б одне ненульове значення, причому не на останньому місці.
8. Знайти середнє арифметичне чисел, розташованих за порядком за першим ненульовим. Вважати, вважати, що ненульових значень може не бути взагалі.
9. Знайти різницю між найбільшим і найменшим числом.
10. Обчислити суму добутків сусідніх пар чисел (кожне число при обчисленні використовується тільки один раз).
11. Знайти позицію першого за порядком числа, що дорівнює нулю.
12. Знайти порядковий номер першого ненульового числа.
13. Знайти порядковий номер останнього ненульового числа.
14. Обчислити середнє арифметичне усіх додатних чисел.
15. Знайти суму модулів тих чисел, які передують першому нульовому, або усіх чисел групи, якщо нульових немає.

8. Класи та об’єкти

План викладу матеріалу:

1. Загальні положення.
2. Модуль класу.
3. Властивості класу.
4. Методи класу.
5. Об’єкти і посилання.
6. Конструктори і деструктори.
7. Приклад класу раціональних чисел.
8. Моделювання динамічних структур даних.
9. Колекції.

Ключові терміни розділу

- *Класи та об’єкти*
- *Події*
- *Майстер побудови класів*
- *Процедура-властивість* `Let`
- *Процедура-властивість* `Get`
- *Ключове слово* `Private`
- *Ключове слово* `Friend`
- *Об’єкт і посилання на нього*
- *Стек*
- *Інкапсуляція*
- *Модуль класу*
- *Властивості класу*
- *Процедура-властивість* `Set`
- *Ключове слово* `Public`
- *Ключове слово* `Static`
- *Методи класу*
- *Конструктори і деструктори*
- *Колекції*

8.1. Загальні положення

Об’єкти і *класи* – це базові поняття при програмуванні на Visual Basic. Об’єктами є форми та елементи керування, бази даних, робочі книги тощо. Сама операційна система Windows побудована на основі класів – її головним об’єктом є вікно. Пакети MS Office цілковито побудовані на класах і роботі з об’єктами цих класів – тут усе, починаючи від застосування і закінчуючи окремим символом, розглядають як об’єкт деякого класу.

Професійний прикладний програміст, що працює у деякій проблемній області і вирішує різноманітні задачі з цієї області, зазвичай починає зі створення класів, що описують специфіку даної проблемної області. Потім уже рішення тих чи інших спеціальних задач він описує у термінах роботи з об’єктами конкретної проблемної області.

Об'єкти є *інкапсульованими* (encapsulated) – тобто вони містять і свій код, і свої дані, що призводить до значно легшої їхньої підтримки порівняно з кодом, написаним традиційним способом. Об'єкти налічують *властивості*, *методи* і *події*.

Властивості – це дані, які описують об'єкт. Множина значень властивостей визначає конкретний стан об'єкта. *Методи* – це те, що об'єкт "уміє" робити. *Подія* – це початок або завершення будь-якої дії у програмі, яка ініціюється самою програмою або середовищем (операційною системою). Для конкретного об'єкта можна написати код (*процедуру опрацювання події*), який виконуватиметься у випадку виникнення певної події.

Множину відкритих (`Public`) властивостей і методів об'єкта називають *інтерфейсом* (interface) об'єкта. Об'єкти створюються з класів; кажуть, що об'єкт є *екземпляром* класу. Клас описує *властивості*, *методи* і *події* об'єкта, їхню область дії, обставини створення і знищення об'єкта. Класи зберігаються у бібліотеці типів (type libraries) і їх можна переглянути за допомогою броузера об'єктів (object browsers).

Клас є узагальненням поняття *типу даних* і задає властивості та поведінку *об'єктів* (або екземплярів класу). Кожен об'єкт належить деякому класу. Відношення між об'єктом і його класом таке ж саме, як між змінною та її типом. Клас – це об'єднання змінних і підпрограм (процедур і функцій) їхнього опрацювання. Змінні називають *властивостями класу*, а процедури і функції – *методами класу*. Об'єднання і локалізація у рамках класу як єдиного цілого даних і функцій/процедур, які опрацьовують ці дані, у програмуванні називають *інкапсуляцією* (дослівно – "вмістиме в оболонці").

Кожен клас має визначений набір подій, які можуть виникати при роботі з об'єктами класу, найчастіше при визначених діях користувача, іноді як результат дії системи. При виникненні події, зв'язаної з тим чи іншим об'єктом, система посилає об'єкту повідомлення, яке аналізується *процедурою опрацювання цієї події*, спеціально створеною при конструюванні об'єкта.

Події забезпечують велику гнучкість при роботі з об'єктами. Методи класу виконуються однаково для усіх об'єктів класу, а на

події кожен об’єкт реагує індивідуально, оскільки має власні процедури опрацювання подій.

8.2. Модуль класу

Здебільшого при застосуванні типу, визначеного користувачем, треба також визначати й операції над даними цього типу. Природно зібрати визначення типу й операцій над ним в одному місці (*класі*).

Синтаксично клас є окремим модулем спеціального вигляду – *модулем класу*, який зберігається у файлі з розширенням **.cls**. Тому розпочинати створення класу у Visual Basic необхідно з уведення модуля класу, виконавши таку послідовність команд:

➢*Project* ➢ *Add Class Module ...* 🗀*New* ❑*Class Module* ❑*Open*

Увести модуль класу можна і за допомогою контекстно-залежного меню: ❑ПКМ {над вікном проекту} ➢ *Add* ➢ *Class Module* ❑*Class Module* ❑*Open*.

Передусім необхідно визначитися з назвою класу (`Name`), яку можна задати, змінивши стандартну назву `Class1` у вікні властивостей класу. Ця назва слугуватиме також і назвою файла з розширенням **.cls**, у якому зберігатиметься вмістиме модуля класу.

Visual Basic надає іншу можливість створення класів – за допомогою *майстра побудови класів* (утилітаVB Class Builder). Для цього на вкладці *Environment* діалогового вікна *Tools\0ptions* у групі *Show Templates For* треба активізувати опцію *Class Modules*.

За допомогою команд ➢ *File* ➢ *New* ➢ *Class* даного майстра можна додати новий модуль класу. У діалоговому вікні, що відкривається, необхідно задати назву класу і, за потреби, інші параметри.

Щоб усі зміни, виконані майстром, було внесено у проект, необхідно виконати команду *Update Project* меню *Files* майстра, або при завершенні роботи з майстром ствердно відповісти на питання щодо необхідності збереження змін. За допомогою майстра, крім класів, можна створювати властивості, методи і події класу, а також власні колекції.

Модуль класу має таку ж саму структуру, як і стандартний модуль. Він складається з двох розділів: *оголошень* і *методів*. У

першому з них описано властивості класу, а в другому – його методи. У модулі діють також специфікатори області дії **Public** і **Private**. **Public** – властивості і **Public** – методи формують *інтерфейс* класу. Тільки до цих властивостей і методів можна звертатися при роботі з об'єктами класу, оголошеними в інших модулях проекту. Зауважимо, що область дії модуля класу – весь проект.

8.3. Властивості класу

Вважають, що розпочинати код модуля класу доцільно з коментаря, що змістовно описує призначення класу, його властивості і поведінку.

Після коментаря розташовують опис змінних, що задають властивості класу. Це можуть бути *атомарні* властивості, які задаються звичайними змінними, і властивості – *учасники*, які є об'єктами інших класів. При введенні властивостей їх зазвичай оголошують закритими (**Private**). Такою є загальноприйнята практика об'єктно-орієнтованого програмування. Якщо властивості оголошують відкритими (**Public**), то під час роботи з об'єктом до них можна отримати прямий доступ як у випадку читання, так і у випадку запису. Проте така свобода у багатьох випадках є небажаною.

Доступ до закритих властивостей передбачено спеціальними *процедурами-властивостями*:

- **Property Let** дає змогу установити нове значення *атомарної* властивості, виконуючи операцію присвоювання.
- **Property Set** виконує ті ж дії, що і попередня процедура, проте застосовна до властивостей-*учасників* (у Visual Basic присвоювання значень об'єктам виконується оператором **Set**).
- **Property Get** дає змогу отримати значення властивості (як атомарної властивості, так і властивості-*учасника*).

Розрізняють п'ять стратегій застосування властивостей:

- *Читання – запис* (`Read – Write`): кожній закритій властивості відповідає пара процедур-властивостей **Let** (**Set**) – **Get**, що доповнюють одна одну.

- *Читання – запис при першому зверненні* (`Read - Write once`): значення властивості, записане при першому зверненні, надалі залишається без змін. З цією метою процедурам-властивостям **`Let`** і **`Set`** присвоюють код, що здійснює спеціальну перевірку, чи проводилося раніше присвоєння значення властивості. Для об'єктів це перевірка типу "`value Is` **`Nothing`**", для змінних типу **`Variant`** – "`value =` **`Empty`**" і т. д.
- Тільки *читання* (`Read only`): використовують виключно процедуру **`Get`**.
- Тільки *запис* (`Write only`): використовують виключно процедуру **`Let/Set`**.
- Ні читання, ні запис (`Not Read – Not Write`), — властивість закрита (використовують для внутрішніх потреб класу).

Для створення процедур-властивостей звичайною практикою є використання заготовок, створених автоматично за допомогою *майстра побудови класів*. Задавши назву процедури і клацнувши OK, отримуємо дві стандартні заготовки **`Property Let`** і **`Property Get`**. Заготовки потім наповнюються змістом і модифікуються. Процедури-властивості відіграють важливу роль у визначенні класів, а тому є сенс розглянути їхній синтаксис.

Процедуру **`Property Let`** використовують для встановлення значення атомарної властивості, вона має такий синтаксис:

```
[Public | Private | Friend] [Static] Property Let _
   назва_властивості ([список_параметрів,] значення)
      [Оператори]
      [Exit Property]
      [Оператори]
End Property
```

Процедуру **`Property Set`** використовують для встановлення значення властивості-учасника (об'єкта), вона має такий синтаксис:

```
[Public | Private | Friend] [Static] Property Set _
  назва_властивості ([список_параметрів,] посилання)
      [Оператори]
      [Exit Property]
      [Оператори]
End Property
```

Процедуру **Property Get** використовують для отримання значення властивості, вона має такий синтаксис:

```
[Public | Private | Friend] [Static] Property Get _
   назва_властивості [(список_параметрів)] [As Type]
      [Оператори]
      [Exit Property]
      [Оператори]
      [назва_властивості = вираз]
End Property
```

Розглянемо деталі синтаксису:

- Ключове слово **Public** засвідчує можливість використання властивості в усіх процедурах і модулях проекту. Крім цього, область дії може поширюватися не тільки на проект, у якому визначено клас, але й на інші проекти. Щоб керувати цим процесом, розширюючи чи звужуючи область видимості **Public** об'єктів, застосовують спеціальні специфікатори, наприклад, опцію **Option Private,** що закриває модуль від інших проектів.
- Ключове слово **Private** засвідчує можливість використання властивості тільки у межах класу.
- Ключове слово **Friend** засвідчує можливість використання властивості у процедурах і модулях тільки того проекту, у якому описано клас;
- Ключове слово **Static** означає, що значення локальних змінних процедури, якщо вони є, не змінюватимуться у проміжку між її викликами.
- Однаковою для пари процедур **Property Let** (**Set**) – **Get** має бути `назва_властивості`.
- Необов'язковий `список_параметрів` використовують здебільшого при заданні властивості, значення якої утворюють масив. Парні процедури упорядковані однаковим списком параметрів. Синтаксис `списку_параметрів` такий самий, як і у параметрів звичайних процедур. Якщо необхідно передати змінне число параметрів, використовують **ParamArray**.

- Параметр `значення` у **`Property Let`** і `посилання` у **`Property Set`** – це назва змінної (об'єкта), значення якої передається властивості. Тип значення, що повертається процедурою **`Property Get`**, повинен збігатися з відповідним типом параметра `значення/посилання`.
- Послідовність `операторів` задає обчислення значення властивості. У тілі процедури використовують оператор **`Exit Property`** для негайного виходу з процедури.

8.4. Методи класу

Будь-яка процедура (**`Sub`**) чи функція (**`Function`**), описана в розділі методів класу, є його методом. Синтаксис методів класу:

```
[Private|Public|Friend][Static]Sub Назва[(параметри)] _
    [оператори]
    [Exit Sub]
    [оператори]
End Sub
```

або

```
[Private|Public|Friend][Static] Function _
                          Назва[(параметри)] _
    [оператори]
    [Exit Function]
    [оператори]
End Function
```

Зазначимо, що майже кожен клас, незалежно від його специфіки, має деякий "джентльменський" набір методів:

- один чи декілька конструкторів класу, у тім числі конструктор за домовленістю, заданий оброблювачем події `Initialize`;
- зазвичай по парі процедур-властивостей, визначених для кожної властивості класу;
- метод, що задає друк властивостей класу;
- метод, що дає змогу в діалозі з користувачем визначати значення властивостей класу – своєрідний конструктор;
- інші методи, що визначають специфіку класу.

Методи класу, у тім числі процедури-властивості **Get**, **Let** і **Set** іноді мають службове слово **Friend** поряд з **Public** і **Private**. Кожен з них задає свою область видимості методу (область програми, де метод доступний для виклику). Нагадаємо: **Private** робить метод видимим тільки усередині класу; **Public** робить метод відкритим і область дії може поширюватися не тільки на той проект, у якому визначено певний клас, але й на інші проекти.

Щоб керувати цим процесом, розширюючи чи звужуючи область видимості **Public**-об'єктів, застосовують спеціальні специфікатори (наприклад, **Option Private**), що закривають модуль від інших проектів.

Методи зі службовим словом **Friend** називають *дружніми.* Слово **Friend**, діючи тільки на один метод, поширює його область видимості на проект, у якому описано клас з **Friend**-методом. Однак для інших проектів дружні методи не доступні, навіть у випадку доступності **Public**-методів. Отож метод "дружить" тільки зі своїм проектом. Хто програмував на C++ та інших мовах, де є успадкування класів, знайомий із дружніми методами. Там вони "дружать" із класами-нащадками батьківського класу, у якому заданий **Friend**-метод. Поза сім'єю класів дружні методи не доступні. У Visual Basic роль сім'ї відіграє проект.

8.5. Об'єкти і посилання

Нехай у проекті маємо деякий клас `MyClass`. Часто кажуть, що в оголошенні

```
Dim Y As MyClass
```

змінна `Y` є об'єктом класу `MyClass`. Щодо терміна "*об'єкт*", то це не зовсім влучно (доцільніше `Y` називати змінною, що має тип `MyClass`). Фактично `Y` є *посиланням* на об'єкт (або вказівником), що зберігає адресу пам'яті, де зберігається об'єкт (чи екземпляр) класу `MyClass`. Посилання займають у пам'яті 4 байти. Під час оголошення посилання на об'єкт пам'ять для самого об'єкта може і не виділятися, як це є у нашому прикладі. У VB є *тільки* один тип посилань – посилання на об'єкт, а тому надалі вживатимемо скорочений термін "*посилання*".

Задати посилання (пов'язати змінну-посилання з об'єктом) можна трьома способами:

- створити новий об'єкт під час оголошення, виокремивши для нього пам'ять;
- створити новий об'єкт за допомогою оператора **Set**, виокремивши для нього пам'ять;
- послатися на вже існуючий об'єкт в операторі **Set**.

Виокремлюють пам'ять для об'єкта за допомогою специфікатора **New**.

Синтаксис оголошення посилань:

```
{Dim|Private|Public|Static} посилання As [New] клас
```

Специфікатор **New** засвідчує, що в момент оголошення посилання необхідно створити новий об'єкт, тобто виокремити для нього пам'ять. У цей момент посилання отримує адресу зберігання у пам'яті цього об'єкта. Виокремлення пам'яті ще не означає ініціалізації значень властивостей об'єкта. Ініціалізацію можна задати при визначенні події `Initialize` чи у спеціально побудованому методі `Init`, який варто запускати на початку роботи з об'єктом. Тільки специфікатор **New** дає змогу створити новий об'єкт для класів, визначених програмістом. З цією метою не можна, наприклад, використати метод `CreateObject`.

Зауважимо, що специфікатор в оголошенні посилань **New** є не обов'язковим, його можна опускати. Вважають, що такі посилання отримають значення в операторі **Set**, – як адресу нового об'єкта чи завдяки посиланню на інші, раніше створені об'єкти. Задати посилання на об'єкт не можна звичайним оператором присвоєння – з цією метою у Visual Basic застосовують спеціальний оператор **Set**:

```
Set посилання = {New клас |об'єктний_вираз |Nothing }
```

Модифікатор `Nothing` розриває зв'язок посилання з об'єктом.

Приклад зв'язування посилання з об'єктом:

```
Dim X As New MyClass
```

або

```
Dim X As New MyClass
Set X = New MyClass
```

Існує два способи зв'язування: *раннє* і *пізнє*. За *пізнього* зв'язування посилання описують так:

```
Dim Y As Object
```

Це оголошення засвідчує, що змінна `Y` є посиланням на об'єкт, про клас цього об'єкта не йдеться – він може бути довільним, і з'ясується це тільки під час виконання програми, коли змінна `Y` зв'язуватиметься з тільки що створеним чи наявним об'єктом конкретного класу. Тому таке зв'язування називають пізнім, чи динамічним. При ранньому зв'язуванні в момент оголошення вказується клас об'єкта, наприклад:

```
Dim A1 As MyClass, A2 As MyClass
```

Це дає змогу ще на етапі компіляції перевіряти, чи допустимі ті чи інші операції над об'єктами `A1` і `A2`. Для програміста передусім важливо те, що за раннього зв'язування автоматично видається підказка про властивості і методи класу `MyClass`.

8.6. Конструктори і деструктори

У мовах об'єктного програмування, як, наприклад, у мові C++, при створенні об'єкта зазвичай викликається конструктор, що резервує пам'ять для об'єкта та визначає значення властивостей. Конструкторів може бути декілька, серед яких є конструктор без параметрів (конструктор за домовленістю). Інші конструктори мають параметри, що дають змогу задати властивості об'єкта в момент ініціалізації.

У Visual Basic усе простіше. Багато в чому це пояснюється тим, що тут, на відміну від багатьох інших мов програмування, є розумна стратегія початкової ініціалізації змінних – про неї уже йшлося. Тому тут діє тільки конструктор за домовленістю, проте і його часто не визначають, покладаючись на стандартну ініціалізацію. Однак ініціалізувати об'єкт "справжніми" значеннями в якийсь момент все одно доведеться. Тому для класу програмісти створюють власні конструктори, які синтаксично є методами класу. Власних конструкторів у класі трапляється декілька.

Деструктор викликається автоматично за умови знищення об'єкта. У Visual Basic немає динамічного знищення об'єкта в

момент, визначений програмістом; об'єкти знищуються при виході з області їхньої дії. Тому деструктор, зазвичай, не пишеться.

Роль конструктора у класах Visual Basic відіграє процедура опрацювання події **Initialize**. Для об'єктів – екземплярів класів подія Initialize виникає при створенні об'єкта. У процедурі опрацювання цієї події немає параметрів, тому вона відіграє роль конструктора за домовленістю.

Роль деструктора у класах Visual Basic відіграє процедура опрацювання події **Terminate**. Подія виникає, якщо усі раніше встановлені посилання на екземпляр класу отримують значення **Nothing** чи всі покажчики перестають існувати, вийшовши з області свого визначення. Процедуру опрацювання події Terminate (деструктор) пишуть дуже рідко, оскільки екземпляр класу буде знищений автоматично.

8.7. Приклад класу раціональних чисел

Як приклад класу, визначимо новий тип даних – раціональні числа та основні операції над ними: додавання, віднімання, множення і ділення (клас **Ratio**). Раціональне число задають парою цілих чисел (m, n) і зображають, зазвичай, у вигляді дробу m/n. Для кожного раціонального числа існує безліч його зображень (наприклад: 1/2, 2/4, 3/6, 4/8, ...), оскільки задають одне і те ж раціональне число. Серед усіх зображень виокремлюють те, в якому чисельник і знаменник нескоротні. Саме такі зображення зберігатимуться в нашому класі. Операції над раціональними числами визначають через відповідні операції над звичайними дробами.

```
Option Explicit
' Клас Ratio
' Властивості класу Ratio
Private m As Integer     ' чисельник
Private n As Integer     ' знаменник
' Конструктори класу Ratio
Private Sub Class_Initialize()
' Конструктор за домовленістю, ініціалізує дробом 1/1
  m = 1 : n = 1
End Sub
```

```
Public Sub CreateRational(ByVal a As Integer, _
            ByVal b As Integer) 'Власний конструктор
 Dim d As Integer ' Найбільший спільний дільник a і b
 If b = 0 Then
     MsgBox "Помилка! Знаменник = 0."
 Else
     ' Зведення знака
     If b < 0 Then b = Abs(b) : a = -a
     ' Зведення  до нескоротного дробу
     d = nsd(a, b) : m=a\d: n=b\d
 End If
End Sub
' Закрита функція обчислення НСД(m,n)
Private Function nsd(ByVal ml As Integer, _
                     ByVal nl As Integer) As Integer
 Dim p As Integer
 ml = Abs(ml): nl = Abs(nl)
 If nl > ml Then p = ml: ml = nl: nl = p
 Do Until n1 = 0
    p = ml Mod nl : ml = nl: nl = p
 Loop
 nsd= ml
End Function
Public Function Plus (a As Ratio) As Ratio
 Dim d As Integer, u As Integer, v As Integer
 Dim R As New Ratio
 u=m*a.zn+n*a.ch : v=n*a.zn
 d = nsd(u, v) : R.ch = u \ d: R.zn = v \ d
 Set Plus = R
End Function
Public Function Minus (a As Ratio) As Ratio
Dim d As Integer, u As Integer, v As Integer
Dim R As New Ratio
 u=m*a.zn-n*a.ch: v=n*a.zn
 d = nsd(u, v) : R.ch = u \ d: R.zn = v \ d
 Set Minus = R
End Function
```

```
Public Function Mult (a As Ratio) As Ratio
 Dim d As Integer, u As Integer, v As Integer
 Dim R As New Ratio
 u=m* a.ch : v=n * a.zn : d= nsd(u, v)
 R.ch = u \ d : R.zn = v \ d
 Set Mult = R
End Function

Public Function Divide(a As Ratio) As Ratio
 Dim d As Integer, u As Integer, v As Integer
 Dim R As New Ratio
 u=m*a.zn : v = n * a.ch
 If v = 0 Then
     MsgBox ("Ділення на нуль!")
 Else
  d = nsd(u, v) : R.ch = u \ d : R.zn = v \ d
  Set Divide = R
 End If
End Function

Public Sub PrintRational()
 Debug.Print (m & "/" & n)
End Sub

Public Property Get ch() As Integer
 ch = m
End Property

Public Property Get zn() As Integer
 zn = n
End Property

Public Property Let ch(ByVal NewValue As Integer)
 CreateRational NewValue, n
End Property

Public Property Let zn(ByVal NewValue As Integer)
 CreateRational m, NewValue
End Property
```

Перевіримо роботу класу `Ratio` на прикладі обчислення виразу $d = \dfrac{a + \frac{3}{5} - \frac{2}{3} \cdot b}{a + 2 \cdot b}$, де a і b – раціональні числа.

```
Public Sub main()
 Dim c As New Ratio, d As New Ratio
 Dim i As Integer, j As Integer
 i = InputBox("Уведіть чисельник")
 j = InputBox("Уведіть знаменник")
 a.CreateRational i, j
 i = InputBox("Уведіть чисельник")
 j = InputBox("Уведіть знаменник")
 b.CreateRational i, j
 c.ch = 3 : c.zn = 5 : Set c = a.Plus(c)
 d.ch = 2 : d.zn = 3 : Set d = b.Mult(d)
 Set c = c.Minus(d)  : Set d = a.Plus(b)
 Set d = d.Plus(b)   : Set d = c.Divide(d)
 d.PrintRational
End Sub
```

8.8. Моделювання динамічних структур даних

Динамічні структури даних (списки, черги, стеки, дерева тощо) відіграють надзвичайно важливу роль у програмуванні. З метою побудови таких структур використовують *посилання* (вказівники), значеннями яких є адреси розташування у пам'яті інших елементів даних (змінних, масивів, об'єктів тощо). У VB/VBA існують посилання тільки на об'єкти, а тому для емуляції динамічних структур даних використовують модулі класів. Кожен елемент структури є подібним до іншого елемента, отож ці елементи є різними екземплярами класу.

Найпростіше динамічні структури даних створювати за допомогою двох модулів класу: один з них вказує на реальну структуру даних, а другий – на елемент цієї структури. Покажемо це на основі побудови стека, структури даних, у якій елементи даних заносять і видаляють тільки через вершину стека.

Клас **StackItem** – елементи стека

```
Option Explicit
Public Value As Variant ' Інформаційне поле
Public Nextltem As Stackltem ' Вказівник на наступний
                             ' елемент стека
Private Sub Class_Initialize()
  Set Nextltem = Nothing
End Sub
Private Sub Class_Terminate()
  Set Nextltem = Nothing
End Sub
```

Клас **Stack** – стек

```
Option Explicit
' Клас Stack
Dim siTop As Stackltem
Public Function Pop() As Variant
' Отримання значення вершини стека з вилученням
 If Not StackEmpty Then
  Pop = siTop.Value : Set siTop = siTop.Nextltem
 End If
End Function
Public Sub Push(ByVal varText As Variant)
' Додавання нового елемента на вершину стека.
 Dim siNewTop As New Stackltem
 siNewTop.Value = varText
 Set siNewTop.Nextltem = siTop : Set siTop = siNewTop
End Sub
Property Get StackTop() As Variant
' Отримання значення вершини стека без вилучення
 If StackEmpty Then
    StackTop = Null
 Else
    StackTop = siTop.Value
 End If
End Property
```

```
Property Get StackEmpty() As Boolean
 StackEmpty=(siTop Is Nothing) ' Перевірка порожності
End Property
Private Sub Class_Initialize()
 Set siTop = Nothing
End Sub
Private Sub Class_Terminate()
 Set siTop = Nothing
End Sub
```

Тестування класу **Stack**

```
Option Explicit
Dim stkTest As New Stack
Sub Main()
 ' Занесення елементів і виведення з вилученням
  stkTest.Push "Привіт!"
  stkTest.Push "Маємо"
  stkTest.Push "стек"
  stkTest.Push "у"
  stkTest.Push "VB!"
  Do While Not stkTest.StackEmpty
     Debug.Print stkTest.Pop()
  Loop
 ' Занесення елементів і виведення без вилучення
  stkTest.Push 12.3
  stkTest.Push 17.5
  Debug.Print stkTest.StackTop  ' 17,5
  Debug.Print stkTest.StackTop  ' 17,5
 ' Виведення елементів з вилученням
  Debug.Print stkTest.Pop       ' 17,5
  Debug.Print stkTest.Pop       ' 12.3
  Debug.Print stkTest.Pop       ' Порожнє значення
End Sub
```

8.9. Колекції

Кажучи про об’єкти, зазначимо необхідність уміння працювати не тільки з окремими об’єктами, але й з групами цих об’єктів. Звісно, можна в багатьох випадках використати масив з метою зображення групи об’єктів. Однак для розв’язання багатьох задач необхідні дещо гнучкіші динамічні структури даних, що дають змогу організувати складні зв’язки між елементами таких структур. Приклади таких структур добре відомі. Серед них – списки лінійні та нелінійні, стеки, деки і черги, дерева бінарні і збалансовані. У сучасних мовах програмування деякі з цих структур стали такою ж частиною мови, як і масиви.

Реалізовується така динамічна структура здебільшого як *клас-колекція*. Найвдалішим прикладом є реалізація самої системи MS Office, яку розглядають як колекцію класів об’єктів. Майже для кожного класу об’єктів існує і клас, що містить колекцію цих об’єктів. Класи-колекції становлять майже половину класів MS Office.

Усі *колекції* подібні і мають деякий стандартний набір методів, що дає змогу видалити або додати новий елемент у колекцію, отримати елемент з колекції, знаючи його порядковий номер або ключ елемента. Деякі колекції мають специфічні властивості і поведінку, зумовлену специфікою самих елементів, що зберігаються в колекції. Частиною мови VB є клас **`Collection`**, який істотно полегшує роботу з динамічними структурами даних у багатьох типових ситуаціях.

Колекція у VB – впорядкована сукупність елементів, власне кажучи, різного типу. Цим колекція відрізняється від масиву, де об’єднано тільки однотипні елементи. Усі елементи проіндексовані, проте деякі можуть мати і ключ, пов’язаний з елементом. Розмір колекції заздалегідь не фіксується і може динамічно змінюватися. Будь-який окремий елемент колекції можна видалити, у цьому випадку не виникає “дір” – елементи перенумеровуються, і неперервність послідовності індексів не порушується. Однак помилку іноді спричинює саме те, що індекс, на відміну від ключа, не є постійною характеристикою елемента і може змінюватися під час роботи з

колекцією. Новий елемент можна додати у довільне місце колекції як перед, так і після будь-якого з елементів.

Клас **Collection** має властивість **Count** і 3 методи: **Add**, **Item**, **Remove**. Розглянемо їх детальніше.

- Властивість **Count** повертає число елементів колекції. Доступна тільки для читання; тип значення, що повертається, – **Long**.
- Метод **Add**(item, key, before, after) додає елементи до колекції. Перший параметр item є обов'язковим і задає елемент, що додається. Параметр key – не обов'язковий, він задається, коли елементу відповідає *ключ*. Два останні параметри уточнюють позицію вставки, задають індекс або ключ елемента, перед або після якого додається новий елемент. Можна задати тільки один з них. Якщо не задано жодного, елемент додається в кінець колекції. Усі параметри мають тип **Variant**.
- Метод **Remove**(key) видаляє елементи колекції. Видаляється елемент із заданим ключем або індексом елемента. Після видалення відбувається перенумерація елементів і зменшується лічильник **Count**.
- Метод **Item**(key) повертає значення елемента списку із заданим ключем або індексом елемента.

Зверніть увагу на потужність методів класу. Вони дають змогу реалізувати різні класичні динамічні структури. Звичайно ж, напрочуд легко реалізувати звичайний однозв'язний список з можливістю додавання елементів на початку чи наприкінці списку. Проте можна реалізувати і значно потужнішу структуру (*словник*), яка є пов'язаною сукупністю пар елементів. Перший елемент пари називають *ключем*, другий – *інформаційним полем*. Специфічністю словників є те, що, володіючи ключем елемента, можна отримати доступ до інформаційного поля. У класі **Collection** на ключ не накладається ніяких обмежень – це звичайний рядок. Зауважимо, що існує також альтернативний спосіб прямого доступу до елементів колекції за індексом, в якому ключем вважають порядковий номер елемента в колекції. Отже, колекція поєднує в собі достоїнства списків і масивів.

Наведемо тепер приклад програми, що детальніше продемонструє роботу з колекцією. Наша колекція налічуватиме дані двох типів: цілочисельні і рядки символів. Частина елементів матиме ключ, інша — тільки індекс. Елементи додаватимемо в задану позицію і вилучатимемо.

```
Sub Main()
 ' Так оголошуються об'єкти (змінні) типу Collection
 Dim MyCollection As New Collection
 Dim i As Integer, N As Long
 With MyCollection
  N =.Count : Debug.Print " Число елементів =", N
  ' Додавання елементів наприкінці списку
  ' Елементи мають індекси, але не мають ключа
   .Add(2) : .Add(4) : .Add(6)
  ' Додавання непарних елементів на свої місця.
  ' before (перед 1-им елементом)
  .Add "перший", "first", 1    ' first – ключ
  ' after (після другого)
  .Add "третій", "third",, 2   ' third – ключ
  ' after (після четвертого)
  .Add "п'ятий", "fifth",, 4   ' fifth – ключ
  N =.Count
  Debug.Print "Кількість елементів після 6-ти Add", N
  Debug.Print "Елементи колекції:"
  For i = 1 To MyCollection.Count
   Debug.Print MyCollection(i)
  Next
  ' Вилучення 4-го і 5-го елементів
  .Remove 4  :  .Remove" fifth"
  N =.Count
  Debug.Print " Кількість елементів після двох Remove=", N
  Debug.Print "Елементи колекції:"
  ' І знов друк колекції, в якій тепер 4 елементи
```

```
  For i = 1 To MyCollection.Count
   Debug.Print MyCollection(i)
  Next
 End With
End Sub
```

Наведемо тепер результати налагоджувального друку:

```
Число елементів = 0
Число елементів після 6-ти Add = 6
Елементи колекції: перший 2 третій 4 п'ятий 6
Число елементів після двох Remove = 4
Елементи колекції: перший 2 третій 6
```

1. Дайте означення класу.
2. Дайте означення об’єкта.
3. Що таке посилання на об’єкт?
4. Для чого використовують події класу?
5. Що таке інкапсуляція?
7. З якою метою використовують поля класу?
8. Що таке методи класу?
10. З якою метою використовують ключове слово **Public**?
11. З якою метою використовують ключове слово **Static**?
12. З якою метою використовують ключове слово **Friend**?
13. Дайте означення конструктора і деструктора.
14. Дайте означення властивості.
15. З якою метою використовують процедуру-властивість **Let**?
16. З якою метою використовують процедуру-властивість **Get**?
9. Що таке методи класу?
6. Для чого використовують ключове слово **Private**?

? Запитання для самоперевірки

17. Для чого використовують процедуру-властивість **`Set`**?
18. Що таке колекція?

Завдання 8.1. Утворивши відповідний клас односпрямованого списку, розв'язати одну з наведених нижче задач:

А) студенти з номерами 1 – 5;

Б) студенти з номерами 6 – 10;

В) студенти з номерами 11–15.

Додатково кожен студент має запрограмувати ще одне завдання (номер додаткового завдання вибирати відповідно до номера студента у списку студентів підгрупи).

А. Скласти програму, яка містить поточну інформацію про *замовлення* на авіаквитки (`пункт призначення, номер рейсу, прізвище та ініціали пасажира, дата вильоту`). Програма має забезпечувати: збереження усіх замовлень у вигляді списку; додавання та вилучення замовлень; виведення усіх замовлень на екран.

Додаткове завдання:

1) за номером рейсу і датою вильоту вивести список замовлень з вилученням їх зі списку;
2) упорядкувати список за зростанням дат, а кожну дату додатково впорядкувати за спаданням номерів рейсів;
3) упорядкувати список за спаданням кількості заявок на окремі рейси (для рейсу врахувати усі дати вильоту);
4) для конкретного рейсу вивести список заявок за зростанням дат без їхнього видалення зі списку;
5) упорядкувати список за зростанням номерів рейсів, а кожен

19. Коротко опишіть властивості та методи класу **`Collection`**.

💻 Завдання для програмування

номер рейсу додатково впорядкувати за спаданням дат.

Б. Скласти програму, яка містить інформацію про *анкетування* населення. Анкета містить дані про респондента (`вік: до 30/від 30 до 40/після 40, стать, освіта: початкова/середня/вища`) та відповідь на питання (`так/ні`). Програма має забезпечувати: збереження усіх анкет у вигляді списку; додавання та видалення анкет; виведення усіх анкет на екран.

Додаткове завдання:

6) встановити кількість мужчин, які мають понад 40 років і вищу освіту, що відповіли "*так*";
7) встановити кількість жінок з середньою освітою, що відповіли "*ні*" і яким менше, ніж 40 років;
8) упорядкувати список за зростанням вікової групи, а для кожної вікової групи додатково впорядкувати його за статтю;
9) встановити кількість жінок, яким виповнилося понад 30 років і які мають початкову освіту та відповіли "*так*";
10) встановити кількість мужчин, які мають менше 30-ти років, початкову освіту і відповіли "*ні*".

В. Скласти програму, яка містить записи про відправлення поїздів (`станція відправлення, станція призначення, номер поїзда, час відправлення, час прибуття`). Програма має забезпечувати: збереження усіх записів у вигляді списку; додавання та видалення записів; виведення усіх записів на екран.

Додаткове завдання:

11) упорядкувати список за кількістю поїздів, які відправляються зі станції у другій половині доби;
12) за станцією призначення вивести дані про всі поїзди, які з неї відправляються;
13) вивести дані про усі поїзди, що перебувають у дорозі понад 12 годин;
14) за станцією відправлення вивести дані про всі поїзди, які до неї прибувають;
15) упорядкувати список за кількістю поїздів, які прибувають на станцію у першій половині доби.

Завдання 8.2. Створити і протестувати заданий клас.

1. Клас, який налічує опис типу вектора цілих чисел і процедури визначення: найменшого елемента; добутку значень усіх елементів вектора; середнього арифметичного значення серед ненульових елементів.
2. Клас, який налічує опис типу вектора цілих чисел і процедури визначення: найбільшого елемента; суми значень усіх елементів вектора; середнього арифметичного значення серед додатних елементів.
3. Клас, який налічує опис типу вектора дійсних чисел і процедури: заміни від’ємних значень вектора на нульові; зменшення додатних значень на задану величину; заміни знаків усіх елементів на протилежні.
4. Клас, який налічує опис двох типів векторів цілих чисел з різною кількістю елементів і процедури: об'єднання двох заданих векторів у третій сумарної довжини; додавання векторів; множення векторів.
5. Клас, який налічує опис типу “*день_тижня*” (*пн*, *вт*, ...), типу вектора з 365-ти елементів (дні невисокосного року) з елементами типу “*день_тижня*” та процедури: заповнення вектора назвами дня тижня; визначення найпершого понеділка року; обчислення кількості певних днів тижня (*понеділків*, *вівторків* і т.д.) протягом року.
6. Клас, який налічує опис типу вектора, елементами якого є символи та процедури: обчислення кількості входжень заданого символа у вектор; заміни заданого символа на інший символ у векторі загалом; перестановки місцями двох заданих символів, якщо вони стоять поряд.
7. Клас, який налічує опис типу вектора, елементами якого є символи та процедури: утворення вектора з окремих символів, що читаються з клавіатури, доповнення новими символами (без пропусків) у зворотному порядку у тому самому векторі; друкування такого вектора.
8. Клас, який налічує опис типу вектора дійсних чисел і процедури: формування (читання) елементів вектора з клавіатури;

друкування значень вектора по п’ять у рядок; перевірити, чи утворюють елементи вектора зростаючу послідовність.

9. Клас, який налічує опис типу вектора цілих чисел і процедури: читання елементів вектора з клавіатури; друкування такого вектора; визначення кількості підряд розташованих нульових значень у векторі.
10. Клас, який налічує опис типу вектора, елементами якого є слова (групи символів) довжиною вісім букв і процедури: читання значень такого вектора з клавіатури; друкування по два слова у рядок; обчислення кількості слів, які починаються і закінчуються однією буквою.
11. Клас, який налічує опис типу “*місяць*” (*січень*, *лютий*, ...), типу “*дата*” (*число* і *місяць*) і типу “*пори_року*” та процедури: перевірки передування однієї дати іншій; утворення дати наступного дня після заданої (невисокосний рік); визначення пори року для заданої дати.
12. Клас, який налічує опис типу матриці дійсних чисел і процедури: читання значень матриці; перестановки місцями двох заданих рядків чи стовпців; заміни всіх значень заданого рядка або стовпця на однакове вказане значення.
13. Клас, який налічує опис типу матриці цілих чисел, типу матриці логічних значень такого самого розміру та процедури: заміни знаків усіх чисел дійсної матриці на протилежні; побудови логічної матриці на основі матриці дійсних чисел за формулою $B[i, j] := X[i, j] >= 1;$ визначення позиції найбільшого числа матриці.
14. Клас, який налічує опис типу множини з латинських літер і процедури: утворення множини з прочитаного тексту, що вводиться з клавіатури, до крапки; друкування у зворотному алфавітному порядку літер цієї множини; обчислення кількості входжень у множину заданої літери.
15. Клас, який налічує опис переліченого типу, – назви міст (до 10-ти), типу множини таких міст і процедури: друкування усіх міст списку, яких немає у жодній з двох заданих множин; друкування усіх міст списку, які є одночасно у кожній з двох

9. Елементи керування

План викладу матеріалу:

1. Головні властивості елементів керування (ЕК).
2. Головні події елементів керування.
3. Форма.
4. Базові елементи керування.
5. Керування проектом.
6. Приклади використання основних елементів керування.
7. Масиви елементів керування.
8. Елементи керування для роботи з дисками, каталогами та файлами.
9. Створення меню.
10. Робота з буфером обміну.

Ключові терміни розділу

- *Властивість* `Caption`
- *Кольорове оформлення ЕК*
- *Властивість* `Name`
- *Властивість* `ToolTipText`
- *Властивість* `Tag`
- *Події натискання клавіш*
- *Властивості форми*
- *Командна кнопка*
- *Списки*
- *Таймер*
- *Проект, файли проекту*
- *ЕК для керування файлами*
- *Буфер обміну*
- *Позиція елемента керування*
- *Властивості* `Enabled` і `Visible`
- *Властивість* `Appearance`
- *Властивості* `Parent` і `Container`
- *Події* `Click` і `DblClick`
- *Події, зв'язані з фокусом*
- *Події форми*
- *Позначка і текстове поле*
- *Смуги прокручування*
- *Рамки та спеціальні вікна*
- *Масиви елементів керування*
- *Меню, типи меню*
- *Методи роботи* з *буфером обміну*

9.1. Головні властивості елементів керування

Створення Windows-застосувань у Visual Basic практично неможливе без використання *елементів керування.* Зовнішній вигляд і функціонування елемента керування визначають його *властивості.* Змінювати властивості під час виконання застосування нескладно, якщо розглядати елементи керування як змінні. Наприклад, для зміни напису командної кнопки `Commandl` використовується її властивість `Caption`:

```
Commandl.Caption = "Новий напис"
```

Проект зазвичай складається з декількох форм, кожна з яких може містити елементи керування з однаковими назвами. Тому синтаксис звертання до властивостей часом розширюють:

```
[Форма.]Control.Властивість = Значення
```

Назву форми зазначати не обов'язково, якщо звертаються до елемента керування, що належить цій формі. Значення властивостей елементів керування зчитують аналогічно. Щоб довідатися про текст на командній кнопці, досить записати:

```
Напис = Commandl.Caption
```

Загалом значення властивості зчитують так:

```
Значення = [Форма.]Об'єкт.Властивість
```

Здебільшого властивості елементів керування є доступними як для зчитування, так і для зміни. Однак існують властивості, що під час виконання доступні тільки для читання (`Locked=True`); інші ж можуть бути недоступними при проектуванні. Розглянемо властивості, якими володіє більшість елементів керування.

Позицію елемента керування визначають чотири властивості: `Left`, `Top`, `Height` і `Width`. Ці значення за домовленістю використовують як одиницю вимірювання *твіп* (*twip*). Твіп — це екранно-незалежна одиниця вимірювання, яка дорівнює 1/20 точки принтера і гарантує незалежність відображення елементів застосування від дисплея (1 *дюйм*=2,54 *см*=1440 *твіпів)*. Властивості `Top` і `Left` задають координати верхнього лівого кута елемента керування на формі, властивості `Height` і `Width` — його висоту і ширину. Відлік у системі координат іде зверху вниз (`Y`) і зліва направо (`X`).

Керування кольоровим оформленням елементів відбувається за допомогою властивостей `BackColor` (колір тла), `FillColor` (відображення тексту і графіки в елементі керування) і `ForeColo`r (заповнення `shapes` – мальованих об'єктів), яким за домовленістю призначають стандартні кольори Windows. Під час проектування колір обирають у діалоговому вікні настроювання кольору, а під час роботи застосування кольори задають або за допомогою колірної схеми RGB, або константами бібліотеки VBRUN. Вид шрифту в елементах керування обирають шляхом установлення значень властивості `Font`.

Періодично при роботі застосування необхідно зробити недоступними деякі елементи керування. З цією метою використо-

вують дві властивості – Enabled і Visible. Властивість Enabled визначає, чи буде елемент керування реагувати на подію, чи ні. Якщо значення властивості дорівнює **False**, елемент керування буде недоступний і користувач не зможе його використати. Зазвичай у цьому випадку елемент підсвічується сірим кольором, так само, як елементи меню, які не можна вибрати. Властивість Visible дає змогу зробити елемент керування невидимим (значення **False**).

Властивість Name є ідентифікатором елемента керування. Отож спочатку доречно задавати назву елемента керування і лише згодом писати для нього код обробки певної події.

Значна кількість елементів керування має властивість Appearance, що відповідає за відображення елемента керування (без візуальних ефектів чи у тривимірному вигляді). Крім того, здебільшого елементам керування можна установити значення властивості ToolTipText – уведений текст відобразиться у підказці, що з'явиться, якщо користувач установить покажчик миші на елементі керування у формі.

Чимало елементів керування Visual Basic має також властивості Parent і Container. Властивість Parent указує на батьківський об'єкт. Завдяки цій властивості можливий доступ до методів чи властивостей батьківського об'єкта. У наступному прикладі рядок заголовка форми, якій належить відповідна кнопка, зберігається у змінній strCaption:

```
strCaption = Commandl.Parent.Caption
```

Властивість Container, на перший погляд, діє аналогічно. Однак на відміну від властивості Parent, яка є доступною тільки для читання, властивість Container дає змогу не тільки зчитувати, але і змінювати контейнер елемента керування. Наприклад, шляхом зміни властивості Container кнопка переходить у елемент Frame:

```
Set Commandl.Container = Framel
```

Елементи керування Form, Frame, PictureBox і ToolBar можуть слугувати контейнером для інших елементів керування.

Властивість Tag призначено для збереження будь-яких додаткових даних, необхідних програмістові, наприклад:

```
Text.Tag = "Введення назви українською мовою"
```

9.2. Головні події елементів керування

Подія `Click` викликається у випадку, коли користувач виконає клацання лівою кнопкою миші на елементі керування. Подія `DblClick` викликається подвійним клацанням кнопкою миші на елементі керування. Часовий інтервал між двома клацаннями встановлюють на панелі керування Windows.

Подія `MouseDown` викликається при натисканні кнопки миші. Щодо цього процедурі обробки події передається кілька параметрів:

```
Control_MouseDown(Button As Integer, Shift As Integer, _
    X As Single, Y As Single)
```

Передані параметри визначають стан кнопок миші (`Button`), клавіш керування (`Shift`) і позицію курсору (`X` і `Y`), як зазначено у табл. 9.1. Замість `Control` задається назва відповідного елемента керування.

Таблиця 9.1. Параметри подій `MouseUp`, `MouseDown` і `MouseMove`

Параметр	Значення
`Button`	Натиснуто кнопку миші: 1 – ліву, 2 – праву, 4 – середню
`Shift`	Натиснуто клавішу: 0 – нічого, 1 – [*Shift*], 2 – [*Ctrl*], 4 – [*Alt*]
`X`	Координата `X`
`Y`	Координата `Y`

Подія `MouseUp` викликається при відпусканні кнопки миші, а подія `MouseMove` – при пересуванні курсору миші; їхні параметри наведено у таблиці 9.1.

Події `KeyPress`, `KeyUp` і `KeyDown` зв'язані з клавіатурою і викликаються для активного елемента керування. Подія `Keypress` повертає код ASCII натиснутої клавіші. При цьому не перехоплюються спеціальні клавіші, такі як [*PrintScreen*] чи [*Alt*], а тільки [*Enter*], [*Esc*] і [*Backspace*]. Процедура передає параметр `KeyASCII`, що містить код ASCII натиснутої клавіші:

```
Control_KeyPress (KeyASCII As Integer)
```

Цей параметр передається як значення, тобто його можна змінювати. Це використовують, наприклад, для фільтрації символів, введених користувачем. Якщо символ неприпустимий, то, установивши значення `KeyASCII` рівним нулю, запобігають його надходження для подальшої обробки (відображення на екрані тощо).

Події KeyDown і KeyUp викликаються натисканням (KeyDown) чи відпусканням (KeyUp) клавіші. Вони відбуваються навіть за натискання спеціальних клавіш керування (наприклад, функціональних). У цьому випадку передаються два параметри: KeyCode і Shift. Параметр KeyCode містить клавіатурний код (а не ASCII) натиснутої клавіші (наприклад: vbKeyFl, vbKeyTab, vbKeyEscape, vbKeyLeft, vbKeyA, ..., vbKeyZ і т.п.), а параметр Shift інформує про стан клавіш [*Shift*], [*Ctrl*] і [*Alt*].

Фокус – одне з найважливіших понять при звертанні до елементів керування у Windows, адже керування одержує активний елемент, тобто елемент, що має фокус. Якщо елемент одержує фокус, то це, відповідно, відображається на екрані – текстове поле зображається з мерехтливим маркером уведення, а командна кнопка виокремлюється пунктирною рамкою навколо напису. Visual Basic дає змогу обробляти дві події, зв'язані з передачею фокуса: LostFocus і GotFocus. Якщо перейти від одного елемента керування до іншого, то для попереднього елемента викликається подія LostFocus, а для нового – GotFocus. Передача фокуса елементові керування відбувається за допомогою методу SetFocus.

9.3. Форма

Форма дає змогу спілкуватися з користувачем. Кожна форма зберігається у проекті як окремий текстовий файл (розширення .frm). Цей файл містить значення властивостей форми, характеристики робочого оточення і код, що належить до елементів керування у формі.

Крім стандартних властивостей, таких як Caption, BackColor, Font тощо, форми мають власні характерні властивості, які розглядатимемо згодом. Для перегляду властивостей форми у вікні властивостей необхідно клацнути на порожньому місці форми (але не в рядку заголовка форми), чи обрати форму зі списку об'єктів у вікні властивостей.

Стандартне вікно має рамку (*border*). Завдяки їй користувач може змінювати розміри вікна – у системному меню є відповідна команда. Вигляд рамки форми можна змінити за допомогою її властивості BorderStyle. Властивість ControlBox визначає, чи відображається системне меню, завдяки якому користувач може вийти з програми. Якщо ж системне меню вилучається, то необхідно забезпечити інший вихід з програми.

Кнопкою максимізації користувач може збільшити вікно до розміру екрана; її наявність визначається властивістю `MaxButton` форми. Якщо ж ця властивість має значення `False`, то відповідна кнопка буде відсутня, а команда `Maximize` (*Розгорнути*) вилучається зі системного меню. Якщо для властивості `MinButton` задати значення `False`, то кнопка затемнюється, а зі системного меню вилучається рядок `Minimize` (*Згорнути*).

Форми, як інші об'єкти, можуть виконувати методи і відповідати на події, що виникають. Найважливі події форми наведено у табл. 9.2.

Таблиця 9.2. Найважливі події форми

Подія	Виникнення
`Activate` (Активізація)	Коли форма стає активною, тобто відображається на екрані
`Deactivate` (Деактивізація)	Коли форма стає неактивною. Наприклад, при активізації на екрані іншої
`Initialize` (Ініціалізація)	При створенні об'єкта типу форма
`Load` (Завантаження)	У момент завантаження форми у пам'ять
`Resize` (Зміна розміру)	При зміні розміру форми
`Terminate` (Завершення)	У момент вилучення форми
`Unload` (Вивантаження)	У момент вивантаження форми з пам'яті

Зазначимо, що синтаксис процедури обробки події форми відрізняється від синтаксису процедур обробки подій елементів керування. Назва процедури обробки події форми завжди містить `Form` (при цьому не важливо, як фактично називається форма).

Найчастіше використовують подію `Load`, яка відбувається при завантаженні форми у пам'ять. Тому `Load` найкраще підходить для ініціалізації об'єктів і змінних, які належать формі. Подія `Unload` викликається, якщо форма видаляється з пам'яті. За допомогою параметра `Cancel` можна скасувати видалення форми з екрана. Покращеним варіантом події `Unload` є подія `QueryUnload`. Поряд з параметром `Cancel` у `QueryUnload` передається і параметр `UnloadMode`, що засвідчує причину виникнення події.

Подія `Resize` викликається за будь-якої зміни розмірів форми. Щодо цього варто враховувати два аспекти. По-перше, під

час обробки події Load форми ще не видно. По-друге, при завантаженні і відображенні форми завжди виникає і подія Resize, тому що при запуску розміри форми змінюються від нульових до заданих у властивостях форми. При створенні процедур для зв'язаних подій типу Activate, GotFocus, Paint і Resize небхідно переконатися, що ці дії не перебувають у протиріччі між собою і що вони не зумовлюють рекурсивних подій.

Події, зв'язані з клавіатурою, викликаються для активного елемента керування. Якщо властивість форми KeyPreview дорівнює True, то подія, зв'язана з клавіатурою, передається спочатку формі, а потім поточному елементові керування. Отже, форма може попередньо аналізувати повідомлення.

2.4. Базові елементи керування

Командну кнопку (CommandButton) використовують для того, щоб почати, перервати чи завершити будь-який процес. Головною подією для кнопки є подія Click. Інші події кнопки застосовують дуже рідко. Для виклику події Click є різні способи. Найпростіший – безпосереднє клацання на кнопці мишею. Таку подію можна викликати, якщо за допомогою клавіші [*Tab*] перемістити фокус на кнопку, а потім натиснути [*Enter*]. Можна програмно викликати подію Click, установивши рівним **True** значення властивості Value, доступної тільки під час виконання.

Властивість Default визначає, що зазначена кнопка є кнопкою, активною за домовленістю. Якщо ця властивість дорівнює **True**, то натисканням клавіші [*Enter*] автоматично генерується подія Click цієї кнопки незалежно від того, який елемент має фокус. Присвоїти значення **True** цій властивості можна тільки для однієї кнопки у формі. Варто зауважити, що у цьому випадку натискання клавіші [*Enter*] перехоплюється і передається цій кнопці. Зазвичай кнопкою за домовленістю є кнопка OK. Властивість Cancel використовується подібно Default. Вона забезпечує перехоплення клавіші [*Esc*] і виклик події Click для відповідної кнопки. Зазвичай цю властивість мають кнопки Cancel (*Скасування*).

Позначка (Label) призначена для відображення підказок користувачу або для виведення певних результатів. Текст позначки користувач змінити не може. Найважливішою властивістю позначки є властивість Caption, що містить відображуваний текст. Властивість BorderStyle задає спосіб відображення тексту – з рамкою

чи без неї. Оформляти текст можна, використовуючи усі можливості форматування тексту, доступні у вікні властивостей, – від вигляду і розміру шрифту до кольору символів.

Якщо текст довший, ніж поле позначки, то частина тексту, що залишилася, просто не відображається (усікається). Цього можна уникнути, якщо присвоїти значення `True` властивості `AutoSize`, яка узгоджує розмір позначки з довжиною тексту. У такий же спосіб можна коректувати розмір позначки і по вертикалі. З цією метою одночасно з властивістю `AutoSize` необхідно установити властивість `Wordwrap`. Тоді слова, що не поміщаються у рядку, автоматично перекидатимуться на наступний рядок.

Текстове поле (`TextBox`) є базовим елементом керування для введення даних. Найважливіші події текстового поля зумовлені клацанням миші та натисканням клавіш. Певне застосування має подія `Change`, яка відбувається щораз при введенні, видаленні чи зміні символу. Наприклад, при введенні у поле слова "*Марія*" подія `Change` викликається п'ять разів – по одному разові для кожної букви. Для аналізу введеного у поле тексту найкраще підходить подія `LostFocus`, яка викликається після того, як текстове поле стає неактивним. Однак якщо це поле є єдиним елементом керування у формі, то воно не втрачає фокус.

Найважливішою властивістю текстового поля є властивість `Text`. Ця властивість містить текст, відображений у полі. Елементи керування, що дають змогу вводити символи, мають властивість `Text`, а елементи, призначені тільки для відображення тексту, – властивість `Caption`.

Установлення властивості `MultiLine` у **`True`** дає змогу вводити у текстове поле декілька рядків. У багаторядковому полі для переходу на новий рядок можна використовувати клавішу [*Enter*]. Однак варто пам'ятати, що для деякої кнопки, можливо, установлена властивість `Default`**.** Тоді натискання клавіші [*Enter*] активізує цю кнопку. У такому випадку для переходу на новий рядок надійніше використовувати комбінацію клавіш [*Ctrl* + *Enter*] чи [*Shift* + *Enter*].

Під час роботи з багаторядковим текстовим полем варто властивості `ScrollBars` присвоювати відповідне значення для відображення смуг прокручування (горизонтальну, вертикальну чи обидві). У цьому випадку смуги прокручування функціонують самостійно (не потрібно писати код опрацювання). У текстовому

полі можна також виокремлювати текст. З цією метою у Visual Basic передбачено три властивості текстового вікна: `SelStart` визначає початкову позицію виокремленого тексту в символах; `SelLength` містить кількість виокремлених символів, а `SelText` дає змогу прочитати чи змінити виокремлений текст. У наступному прикладі виокремлюється текст із 2-го по 6-й символ і замінюється на "*новий*":

```
Textl.SelStart = 2
Textl.SelLength = 5
Textl.SelText ="новий"
```

Іноді в полі необхідно швидко видалити текст чи замінити його новим. З цією метою виокремлюють увесь текст у полі, як тільки дане поле одержує фокус.

```
Private Sub Textl_GotFocus()
    Textl.SelStart = 0
    Textl.SelLength = Len(Textl.Text)
End Sub
```

Незалежні перемикачі (`CheckBox`) – це елементи керування, які можна відзначати (ставити "галочку"), обираючи з набору опцій одну чи декілька опцій. Індикатор може мати три різних стани, які є значеннями властивості `Value`: відзначений (значення 1), не відзначений (0), відзначений і недоступний (значення 2; установити такий стан можна тільки програмно).

Залежні перемикачі (`OptionButton`) – це елементи керування, призначені для установлення тільки однієї опції з групи. Зазвичай усі перемикачі форми об'єднано в одну групу. Групу перемикачів можна сформувати, якщо розмістити їх в окремий контейнер (наприклад, `Frame`). Найважливішою властивістю перемикачів є властивість `Value`, яка визначає стан перемикача (позначено – **`True`**, не позначено – **`False`**).

Найважливішою подією для перемикачів є подія `Click`.

Список (`ListBox`) дає змогу обирати зі списку один чи декілька елементів. У будь-який час у список можна додавати нові елементи чи видаляти існуючі. Якщо не всі елементи можуть одночасно відобразитися у полі списку, то в ньому автоматично відо-

бражаються смуги прокручування.

Основна подія списку – `Click` – викликається, якщо користувач за допомогою миші чи клавіш керування курсором обирає елемент у списку.

Список – це перший з розглянутих нами елементів керування, для яких важливу роль відіграють *методи*. Для додавання нових елементів у список використовують метод `Addltem`, що має такий синтаксис:

```
Назва_списку.Addltem Елемент [, Індекс]
```

Параметр `Елемент` задає елемент списку, що додається. Параметр `Індекс` указує місце вставки у список нового елемента. Здебільшого заповнення списку виконується при завантаженні форми:

```
Private Sub Form_Load
        Listl.Addltem "яблуко"
        Listl.Addltem "груша"
        Listl.Addltem "персик"
        Listl.Addltem "абрикос"
End Sub
```

Для вилучення елемента зі списку використовують метод `RemoveItem`, якому як параметр передається індекс елемента, що вилучається. Індексація елементів списку починається з 0. Наприклад:

```
Listl.Removeltem 2
' Вилучено третій елемент списку (персик)
```

Для вилучення усіх елементів списку використовують метод `Clear`:

```
Listl.Clear
```

Значенням властивості списку `Text` є текст обраного елемента списку чи порожній рядок, якщо жоден елемент не вибрано. Властивість списку `List()` дає змогу визначити текст елемента списку за його індексом:

```
Thirdltem = Listl.List(2)
```

Властивість `Listlndex` містить індекс обраного елемента. Комбінуючи властивості `List()` і `Listlndex`, можна одержати вибраний елемент списку, наприклад:

```
Lstltem = Listl.List(Listl.Listlndex)
```

Якщо у списку не вибрано жодного елемента, то значення властивості ListIndex дорівнює –1. Поточна кількість елементів списку зберігається у властивості ListCount. Елементи списку за замовчуванням відображаються в одному стовпці. Під час проектування, за необхідності, їхнє число можна змінити з допомогою властивості Columns. Заповнення стовпців у цьому випадку відбувається послідовно – спочатку заповнюють перший, потім другий і т.д.

Властивість Sorted визначає спосіб розташування елементів у списку. Якщо установити цю властивість, то всі елементи сортуватимуться за алфавітом, навіть якщо їх додано із зазначенням індексу.

Користувач може обирати одночасно декілька елементів списку. З цією метою необхідно присвоїти властивості MultiSelect одне зі значень, які наведено у таблиці 9.3.

Таблиця 9.3. Значення властивості Multiselect

Значення	Опис
0	Множинний вибір неможливий. У списку можна вибрати тільки один елемент
1	Простий множинний вибір. Елементи списку вибираються клацанням миші чи натисканням клавіші пропуску
2	Розширений множинний вибір. Користувач може вибрати декілька елементів за допомогою миші чи клавіші керування курсором з використанням клавіш [*Shift*] і [*Ctrl*]

При множинному виборі властивість Text містить текст останнього вибраного елемента списку. Значення властивості Selected() елемента списку засвідчує: обрано даний елемент списку (значення **True**) чи ні. Наприклад:

```
If Listl.Selected(2) Then strName = Listl.List(2)
```

Поле зі списком (ComboBox) – це комбінація двох елементів керування: списку зі значеннями і поля введення тексту. Поле зі списком використовують у тому випадку, якщо не можна заздалегідь визначити значення, які варто включити до списку, чи список містить занадто багато елементів. У такому списку потрібне зна-

чення можна не тільки вибирати, але і вводити безпосередньо у поле введення. Нове значення після введення автоматично збережеться у списку.

Для поля зі списком важливу роль відіграють події: `Click` – для вибору елемента списку і `Change` — для зміни запису у полі введення тексту.

Поле зі списком має майже усі властивості текстового поля `TextBox` і списку `ListBox` (за винятком властивості `MultiLine`). Варто виокремити властивість `Style`, що визначає зовнішній вигляд і функціонування поля зі списком.

Таблиця 9.4. Властивість `Style` поля зі списком

Константа	Значення	Опис
`vbComboDropDown`	0	Значення за домовленістю. `ComboBox` – текстове поле для введення і список, що відкривається
`vbComboSimple`	1	`ComboBox` – текстове поле і постійно відкритий список
`vbCoroboDropDownList`	2	Відрізняється від списку зі значенням `VbComboDrop-Down` тільки тим, що користувач не може вводити текст у поле введення

Смуги прокручування (`ScrollBar`). Деякі елементи керування (наприклад, `TextBox`, `ListBox`) використовують смуги прокручування, причому від програміста не вимагається написання програмного коду для виконання прокручування. Смуга прокручування як окремий елемент керування хоча і призначена для виконання аналогічних функцій, однак не виконує автоматично будь-яких дій, тобто її поведінку необхідно програмувати. Існує два види смуг прокручування: горизонтальна і вертикальна. Смуги використовують дві головні події:

- `Change`, що виникає унаслідок зміни позиції бігунка чи після програмної зміни значення властивості `Value`.
- `Scroll`, що відбувається під час прокручування (коли користувач захопив і пересуває бігунок).

Перед використанням смуги прокручування необхідно установити для неї діапазон прокручування, що вказує на кількість кроків прокручування між крайніми позиціями бігунка. Поточне положення бігунка визначається значенням властивості `Value`.

Діапазон прокручування визначається властивостями `Min` і `Max` смуги прокручування. При цьому значення `Min` завжди відповідає верхньому кінцю смуги, а `Max` — нижньому (для вертикальної смуги прокручування), і при прокручуванні вмісту вікна зверху вниз значення властивості `Value` збільшується. Для горизонтальної смуги прокручуваня значення властивості `Value` збільшується, відповідно, при прокручуванні вмісту вікна зліва направо.

Клацання кнопкою миші на одній із двох кнопок зі стрілками на смузі змінює значення властивості `Value` на величину, зумовлену властивістю `SmallChange`. Якщо користувач клацне в області між бігунком і деякою з цих кнопок, то значення властивості `Value` і, відповідно, положення бігунка змінюється на величину, зумовлену властивістю `LargeChange`.

Таймер (`Timer`). За допомогою таймера можна запускати чи завершувати процеси у визначені моменти часу. Таймер використовують і у випадку, коли застосування виконується у фоновому режимі. Під час виконання програми таймер є невидимим. Таймер має єдину подію `Timer`, яка викликається при досягненні встановленого часу.

Для установлення інтервалу часу слугує властивість `Interval`, значення якої встановлюється у мілісекундах. Наприклад, значення 250 викликає подію `Timer` через кожні 250 мілісекунд незалежно від того, яке застосування є активним. Для відключення таймера треба присвоїти властивості `Interval` значення 0 чи властивості `Enabled` значення **`False`**. Максимально припустимий інтервал становить 64757 мілісекунд. Однак варто пам’ятати, що операційна система може обробляти тільки 18,2 переривання таймера в секунду, тому

точність задання інтервалу становить максимум однієї вісімнадцятої секунди. У Windows можна використати не більше 31-го таймера. Якщо обробка події `Timer` триває довше, ніж задано значенням `Interval`, то нова подія `Timer` не викликається, доки Visual Basic не обробить попередню подію.

Рамка (`Frame`) – це один з елементів-контейнерів. Його призначення – об'єднувати у групі декілька елементів керування. Об'єкти, об'єднані за допомогою рамки, можна як єдине ціле пересувати, активізувати і деактивізувати, робити видимими чи невидимими. Деякі елементи самі мають потребу в контейнері – наприклад, усі залежні перемикачі на формі завжди об'єднуються в одну групу. Щоб створити другу групу залежних перемикачів, потрібно необхідні перемикачі об'єднати в елементі-контейнері. Для об'єднання об'єктів у групу потрібно спочатку створити елемент-контейнер, а потім додати у нього необхідні елементи керування. Якщо необхідні елементи керування вже знаходяться на формі, їх досить перемістити в елемент-контейнер. Щоб перевірити, чи дійсно елемент належить контейнеру, досить перемістити контейнер. Елемент керування, що належить контейнеру, переміщується разом з ним.

Рамка не має особливих властивостей, притаманних тільки їй. Події рамки зазвичай не аналізують, тому що найчастіше проектувальник працює тільки з елементами керування, що належать рамці.

Вікно з малюнком (`PictureBox`) призначене для відображення малюнків та інших графічних об'єктів. На тлі цього елемента (як і на тлі форми) можна малювати власні графічні зображення, використовуючи елементи керування `Line` (пряма) і `Shape` (фігура). Елемент керування `PictureBox` є також елементом-контейнером, тому його можна використовувати для об'єднання інших елементів.

Події елемента `PictureBox` зазвичай не обробляють, хоча за необхідності це можна зробити. Положення `PictureBox` у формі задається властивістю `Align`, що визначає: буде його закріплено до одного з країв форми чи збережеться положення, задане програмістом. Якщо елемент керування закріплюється до одного з країв форми, то його розмір (ширина чи висота) завжди установлюється відповідно до розміру форми. Властивість `AutoSize` визначає, чи будуть авто-

матично змінюватися розміри елемента керування для відображення малюнків різного розміру.

Найважливіша властивість PictureBox – це властивість Picture, що містить відображуваний графічний об'єкт. Це може бути растрове зображення (*.bmp), піктограма (*.ico), метафайл (*.wmf) чи розширений метафайл (*.emf), а також gif- і jpeg-файли. При виконанні додатка для зміни властивості використовують функцію LoadPicture, наприклад:

```
Picture1.Picture = LoadPicture("C:\WINDOWS\AUTOS.BMP")
```

Зберегти зображення можна за допомогою функції SavePicture, наприклад:

```
SavePicture Picture1.Picture, "BUILD,BMP"
```

Методи PictureBox дають змогу програмно намалювати крапку, лінію, коло тощо. Наприклад:

```
Picture1.Line (0, 0)-(100, 500), vbRed
Picture1.Circle (300, 300), 250, vbBlue
```

Здатність елемента PictureBox відображати малюнки різних форматів використовують для перетворення піктограми (*.ico) у растрове зображення (*.bmp). З цією метою необхідно завантажити піктограму і зберегти її з розширенням .bmp. Однак растрове зображення перетворити у піктограму не можна.

Зображення (Image) також використовують для відображення малюнків. Однак на відміну від PictureBox, зображення не є елементом-контейнером, не дає змоги малювати і не припускає групування об'єктів. Однак Image використовує менше ресурсів і перемальовує швидше, ніж PictureBox. Тому для відображення малюнків Image вважають найкращим варіантом. Оскільки головне призначення Image – відображення малюнків, то його події зазвичай не аналізують.

Головною властивістю Image є властивість Picture, яка дає змогу визначити малюнок, що відображатиметься елементом керування, на стадії проектування або при виконанні програми. Властивість Stretch визначає, як відображається малюнок. Якщо значення

цієї властивості дорівнює **True**, то розміри малюнка змінюються відповідно до розмірів елемента керування `Image`, у іншому випадку розміри елемента керування пристосовуються до розміру малюнка.

9.5. Керування проектом

Усі застосування Visual Basic, навіть найпростіші, створюються у вигляді проектів. Проект – це контейнер, у якому знаходяться форми застосування та інші візуальні компоненти разом з програмним кодом. Отже, проект – це засіб інтеграції візуальних і програмних компонентів застосування. Проект Visual Basic необхідно зберігати в окремій папці. Усі компоненти зберігаються в пам'яті окремо і незалежно один від одного. Проект може налічувати такі файли:

- Файл форми (`*.frm`). Для кожної форми створюється окремий файл.
- Файл форми з елементами керування, що містять бінарну інформацію (`*.frx`). Такі файли створюються автоматично для форм, які містять елементи керування з властивостями `Picture` і `Icon`.
- Файл проекту, що містить посилання на свої компоненти (`*.vbp`). Після компіляції Visual Basic створює службовий файл доповнення до файла проекту (`*.vbw`).
- Файл кожного програмного модуля (`*.bas`).
- Файл кожного модуля класів (`*.cls`).
- Файли додаткових елементів керування (`*.ocx`).
- Максимум один файл ресурсів (`*.res`). Файл ресурсів слугує для зберігання інформації (тексту, значків, растрових зображень), модифікація якої не вимагає редагування коду.
- Кілька файлів, що залежать від типу проекту (`*.ctl` та ін.).

Отже, окремі компоненти проекту зберігаються ізольовано. Це можна побачити, зберігаючи програму. З цією метою необхідно вибрати в меню *File* команду *Save Project* чи клацнути на панелі інструментів на піктограмі з зображенням дискети. Після цього для кожної складової проекту з'являється діалогове вікно *Save As*.... Після збереження всіх компонентів проекту зберігається безпо-

середньо сам проект. У цьому випадку створюється файл типу `vbp`. У файлі проекту, окрім деяких установок, зберігається також інформація щодо його компонентів і зв'язків між ними.

Сам компонент, наприклад форма, нічого не "знає" про інші компоненти проекту. Тому їх можна включати і в інші проекти. При виконанні команди *File\0pen Project* компоненти завантажуються з урахуванням взаємозв'язків між ними. Кожен компонент проекту можна зберігати чи видаляти окремо. З цією метою необхідно виокремити його у вікні проекту і вибрати команду меню *File\Save* чи *Project\Remove*. У проект можна додавати вже існуючі компоненти за допомогою команди *Project\Add*.

Щоб програма Visual Basic функціонувала не тільки у середовищі Visual Basic, її необхідно скомпілювати і скомпонувати. З цією метою передбачено команду меню *File\Make *.EXE*.

Однією з причин успіху Visual Basic є можливість використання так званих *Custom Controls* – елементів керування, розроблених сторонніми виробниками. У назві OCX перша літера позначає абревіатуру OLE (*Object Linking and Embedding*). Для включення елемента керування у проект необхідно викликати діалогове вікно *Project\Components*. На вкладках *Controls*, *Designers* і *Insertable Objects* цього вікна вибираються елементи керування, які необхідно додати до зазначеного проекту.

У деяких випадках після завантаження проекту Visual Basic не відображає деякі вікна середовища розробки. Для активізації "зниклого" вікна необхідно скористатися однією з команд меню *View*.

9.6. Приклади використання базових елементів керування

Проект 1. *Прийом повідомлень у різних полях*. Розмістимо на формі два текстових поля для *введення повідомлень* і дві позначки для *виведення відповідного повідомлення*. При роботі застосування користувач набирає чергове повідомлення в активному полі. Після натискання клавіші *Enter* або у випадку втрати фокуса прийняте повідомлення відображається у відповідній позначці. Активізується альтернативне поле і все починається спочатку. Для виходу із

застосування можна використати стандартні засоби форми: кнопку закриття форми або системне меню форми. Виведемо вміст файла форми *Forml. frm*:

```
VERSION 5.00
Begin VB.Form Forml
  Caption          =  "Forml"
  ClientHeight     =  2595
  ClientLeft       =  60
  ClientTop        =  345
  ClientWidth      =  4680
  LinkTopic        =  "Forml"
  ScaleHeight      =  2595
  ScaleWidth       =  4680
  StartUpPosition  =  3     'Windows Default
  Begin VB.TextBox Input2
    Height   =  495
    Left     =  480
    Tablndex =  1
    Text     =  "Input2"
    Top      =  1320
    Width    =  1335
  End
  Begin VB.TextBox Inputl
    Height   =  495
    Left     =  480
    Tablndex =  0
    Text     =  "Inputl"
    Top      =  360
    Width    =  1335
  End
  Begin VB.Label  Output2
    Caption  =  "Output2"
    Height   =  495
    Left     =  2280
    Tablndex =  3
    Top      =  1320
    Width    =  1335
```

```
  End
  Begin VB.Label Output1
    Caption   =  "Output1"
    Height    =  495
    Left      =  2280
    TabIndex  =  2
    Top       =  360
    Width     =  1335
  End
End

Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Private Sub Input1_GotFocus()
  Input1.Text = ""
End Sub

Private Sub Input1_KeyUp(KeyCode As Integer, Shift As _
                                                  Integer)
  If KeyCode = vbKeyReturn Then
    Output1.Caption = Input1.Text
    Input2.SetFocus
    KeyCode = 0
  End If
End Sub

Private Sub Input1_LostFocus()
  Output1.Caption = Input1.Text
  Input2.SetFocus
End Sub

Private Sub Input2_GotFocus()
  Input2.Text = ""
End Sub
```

```
Private Sub Input2_KeyUp(KeyCode As Integer, Shift As _
                                                  Integer)
  If KeyCode = vbKeyReturn Then
    Output2.Caption = Input2.Text
    Inputl.SetFocus KeyCode = 0
  End If
End Sub

Private Sub Input2_LostFocus()
  Output2.Caption = Input2.Text
  Inputl.SetFocus
End Sub
```

Проект 2. *Виведення дільників натурального числа.* Користувач вводить у текстове поле натуральне число, а програма у багаторядковому текстовому полі виводитиме числа, на які це число ділиться без залишку, та результат ділення. На формі повинні бути такі елементи:

- Кнопка Commandl: Caption = "Дільники"
- Кнопка Command2: Caption = "Вихід"
- Текстове поле Text1: Text = "" ' Порожній рядок
- Текстове поле Text2: Text = "", MultiLine= true
- Позначка Label 1: Caption = "Ділене"
- Позначка Label2: Caption = "Результат"
- Форма Forml: Caption = "Кратні числа"

```
Option Explicit
Dim dilene As Long, dilytel As Long, resultat As Double
Private Sub Commandl_Click()
  Text2.Text = ""
  If Not IsNumeric(Textl.Text) Then
    MsgBox "Це не число", vbCritical
    Exit Sub
  Else
    dilene = Textl.Text ' Автоматичне перетворення
  End If
```

```
  For dilytel = 1 To dilene
   resultat = dilene / dilytel
   If resultat = Int(resultat) Then
      Text2 = Text2 + Str(dilene)+":"+ _
              Str(dilytel)+"="+Str(resultat)+vbCrLf
        ' Власт. Text є основна - може не зазначатися
   End If
  Next
End Sub
Private Sub Comnand2_Click()
  End
End Sub
```

Проект 3. *Табулювання квадратичної функції.* Користувач натискає кнопку "*Табулювати*", а програма у багаторядковому текстовому полі виводить результати табулювання квадратичної функції. На формі повинні бути такі елементи:

- Кнопка `Commandl: Caption = "Табулювати"`
- Кнопка `Command2: Caption = "Вийти"`
- Форма `Forml:  Caption = "Табулювання функції"`
- Текстове поле `Text1: Text = "",  MultiLine = true,`
  `Locked = True, Srollbars = 3 ' Both`
  `Text = " Аргумент  Функція`
  `----------"`

```
Private Sub Commandl_Click()
Dim x As Double, y As Double, st As String
 With Textl
  .Locked = False
  For x = 2 To 5 Step 0.2
   y = Round(x ^ 2-3 * x+1, 2)
    st = vbCrLf & Space(5) & Str(x) & Space(25) & Str(y)
   .Text = .Text & st
  Next x
  .Locked = True
 End With
End Sub
Private Sub Command2_Click()  :  End  :  End Sub
```

Проект 4. *Аналіз кодів клавіатури.* Користувач натискає довільну клавішу на клавіатурі, а програма видає клавіатурний код клавіші, код ASCII і відповідний символ (при їхній наявності), а також натискання однієї з клавіш керування. На формі з'являються такі елементи:

- Форма Forml: Caption = "Аналіз кодів клавіатури"
- Текстове поле Textl: Locked = True, Text = ""
- Текстове поле Text2: Locked = True, Text = ""
- Текстове поле Text3: Locked = True, Text = ""
- Текстове поле Text4: Locked = True, Text = ""
- Позначка Label1: Caption = "код ASCII"
- Позначка Label2: Caption = "Клавіатурний код"
- Позначка Label3: Caption = "Клавіша керування"
- Позначка Label4: Caption = "Символ"

```
Private Sub Textl_KeyDown(KeyCode As Integer, _
                                    Shift As Integer)
  Text2.Text = KeyCode  :  msg$ = ""
  If Shift And 1 Then msg$ = "SHIFT+" & msg$
  If Shift And 2 Then msg$ = "CTRL+" & msg$
  If Shift And 4 Then msg$ = "ALT+" & msg$
  If Len(msg$)>0 Then msg$ = Left(msg$, Len(msg$) - 1)
  TextS.Text = msg$
End Sub
Private Sub Textl_KeyPress(KeyAscii As Integer)
  Textl.Text = KeyAscii
  Text4.Text = Chr${KeyAscii)
End Sub
Private Sub Textl_KeyUp(KeyCode As Integer, Shift _
                                          As Integer)
  Textl.Text = ""
  Text2.Text = ""
  TextS.Text = ""
  Text4.Text = ""
End Sub
```

Проект 5. *Керування об'єктом*. Програма повинна за натисненням однієї зі стрілок пересувати об'єкт по формі у відповідному напрямі. Система координат форми має початок у лівому верхньому куті (координата (`0; 0`)). Відповідно, якщо треба, щоб об'єкт змістився вниз на *n* твіпів, необхідно додати до його вертикалі (`Top`) значення *n*, те ж і з горизонталлю (рух вправо). Для пересування вверх або вправо відповідні координати треба зменшувати. Цього разу на формі з'явиться тільки один об'єкт – фігура (`shapel` – коло).

```
Dim x As Integer ' Координата об'єкта по горизонталі
Dim y As Integer ' Координата об'єкта по вертикалі
Private Sub Form_KeyDown(KeyCode As Integer, _
                                          Shift As Integer)
  x = Shapel.Left 'Положення об'єкта по горизонталі
  y = Shapel.Top ' Положення об'єкта по вертикалі
  If KeyCode = 37 Then x = x - 100
  ' Якщо код натиснутої клавіші = 37  (стрілка вліво)
  If KeyCode = 39 Then x = x + 100
  ' Якщо код натиснутої клавіші = 39 (стрілка вправо)
  If KeyCode = 38 Then y = y - 100
  ' Якщо код натиснутої клавіші = 38 (стрілка вгору)
  If KeyCode = 40 Then y = y +100
  ' Якщо код натиснутої клавіші =40 (стрілка вниз)
  Shapel.Left = x
  Shapel.Top = y
End Sub
```

Проект 6. *Визначення дійсних коренів квадратного рівняння.* У програмі аналізуватимемо коди клавіш, які натискатиме користувач при введенні числових даних для коефіцієнтів рівняння. Символи, не сумісні з представленням чисел, погашатимемо. Події, зв'язані з клавіатурою, викликаються для активного елемента керування (`ActiveControl`). У проекті властивість форми `KeyPreview` дорівнює **`True`**, тому події, зв'язані з клавіатурою, передаються формі, а форма попередньо аналізує дані. На формі з'являються такі елементи:

- Форма `Forml: Caption = "Квадратні рівняння", KeyPreview = -1 'True`
- Командна кнопка `Command1: Caption = "Корені"`
- Командна кнопка `Command2: Caption = "Вийти"` (метод `End`)
- Текстове поле `Text1: Text = ""`
- Текстове поле `Text2: Text = ""`
- Текстове поле `Text3: Text = ""`
- Позначка `Label1: Caption = " Коефіцієнт A (<>0) "`
- Позначка `Label2: Caption = " Коефіцієнт B "`
- Позначка `Label3: Caption = " Коефіцієнт C "`
- Позначка `Label4: Caption = "", Wordwrap = -1, AutoSize = -1 ' True`

```
Private Sub Commandl_Click ()
 Dim a As Double, b As Double, c As Double
 Dim d As Double, xl As Double, x2 As Double
 If (Textl.Text="")Or(Text2.Text="")Or(Text3.Text="") Then
   MsgBox"Задайте всі коефіцієнти",vbOKOnly, "Увага"
   Exit Sub
 End If
 a = Textl.Text : b = Text2.Text : c = Text3.Text
 If a = 0 Then
   Label4.Caption = " Коефіцієнт a=0!"
   Exit Sub
 End If
 d = b*b-4*a*c
 If d < 0 Then
   Label4.Caption = "Дійсних коренів нема!"
 Else
   xl = (b+Sqr(d))/(2*a) : x2 = (-b+Sqr(d))/(2*a)
   Label4.Caption = "Дійсні корені:" + vbCrLf + _
          "xl=" + Str(xl) + vbCrLf + "x2=" + Str(x2)
 End If
End Sub
```

```
Private Sub Form_KeyPress(KeyAscii As Integer)
 Dim buf As String, ob As Object
 Set ob = ActiveControl
 If ob = Textl Then buf = Textl.Text
 If ob = Text2 Then buf = Text2.Text
 If ob = Texts Then buf = Text3.Text
 Select Case KeyAscii
   Case Asc("0") To Asc("9"), 8
     Exit Sub
   Case Asc(",")
     If InStr(buf, ",") <> 0 Then KeyAscii - 0
   Case Asc("-")
     If (Len(buf) > 0) Then KeyAscii = 0
   Case 13:
     If ob = Textl Then Text2.SetFocus
     If ob = Text2 Then TextS.SetFocus
     If ob = TextS Then Commandl.SetFocus
   Case Else
    KeyAscii = 0
 End Select
End Sub
```

Проект 7. *Перевірка незалежного перемикача*. На екран виводимо повідомлення про стан незалежного перемикача. На формі з'являються такі елементи:

- ФормаForml: `Caption = "Перевірка перемикача"`
- Командна кнопка `Commandl: Caption = "Перевірка"`
- Командна кнопка `Command2: Caption = "Вийти"` (оператор `End`)
- Незалежний перемикач `Checkl: Caption = "Перемикач", Value=l`

```
Private Sub Commandl_Click()
 If Checkl.Value = 1 Then
   MsgBox "Перемикач увімкнено",  , "Перевірка"
 Else
   MsgBox "Перемикач вимкнено", , "Перевірка"
 End If
End Sub
```

9.7. Масиви елементів керування

Масиви елементів керування подібні до звичайних масивів змінних (окремі елементи масиву розрізняють за *індексом*). Найпростіше створювати масив елементів керування за допомогою копіювання елемента керування у буфер обміну. При вставці у ту ж форму в ній знаходиться ще і первісний елемент із тією ж назвою, що й у буфері обміну. Тому Visual Basic запитує, чи доцільно створювати масив елементів, а чи варто дати нову назву елемента керування, який додають.

Якщо на запитання відповісти "`No`", то Visual Basic помістить на форму елемент керування і дасть йому назву за домовленістю. Наприклад, для кнопки `Commandl` (цю назву треба зразу ж змінити, щоб не було двох елементів з однаковими назвами). Після відповіді "`Yes`" на формі з'являться два елементи керування з однаковою назвою. Щоб Visual Basic міг розрізняти ці елементи, автоматично установлюється властивість `Index` (це можна помітити у вікні властивостей). У списку елементів відображається декілька елементів керування з однаковою назвою, після якої у дужках вказується індекс. Значення індексу збігається з зазначеним у рядку `Index`. З метою створення масиву елементів керування однакового типу можна задати елементам спільну назву і установити визначене значення властивості `Index`.

Головна перевага масиву елементів керування полягає у тому, що всі його елементи використовують одні і ті ж самі процедури опрацювання подій. Для визначення елемента, що викликав подію, Visual Basic передає у процедуру опрацювання події аргумент `Index`. Цей аргумент приймає різні значення для кожного елемента масиву. Перевіривши це значення, можна визначити, яку дію варто виконувати, наприклад:

```
Private Sub CutCopy_Click (Index As Integer)
   Call Copy
   If Index = 0 Then Call Delete
End Sub
```

У прикладі завжди викликається процедура `Copy`. Якщо елемент має індекс 0, то після неї ще виконується процедура `Delete`.

Іноді необхідно використовувати масив елементів керування, не знаючи їхньої кількості. У цьому випадку варто скористатися оператором Load, що дає змогу завантажити елемент керування під час роботи програми. Для цього необхідний тільки початковий елемент керування, визначений як масив (властивість Index містить визначене значення). За допомогою оператора Unload програмно можна видалити зазначений через властивість Index елемент керування масиву (за винятком створених при проектуванні).

Проект 8. *Програмне створення масиву елементів керування*. У цьому прикладі на стадії проектування форма має одну командну кнопку з індексом 0 і два залежні перемикачі Option1 і Option2. Якщо активним є перемикач Option1 (*Додати нову кнопку*), то під час виконання програми клацання на будь-якій командній кнопці додає нову кнопку. Змінна nCount використовується для визначення індексу нового елемента. За допомогою властивості Visible новий елемент відображається. Оскільки усі властивості збігаються з властивостями вихідного об'єкта, варто змінити також властивість Top, щоб кнопки не накладалися одна на одну. Якщо активним є перемикач Option2 (*Видалити кнопку*), то під час виконання програми клацання на будь-якій командній кнопці спричинює її видалення. На формі повинні бути такі елементи:

- Форма Form1: Caption = "Динамічне створення кнопок"
- Командна кнопка Command1: Caption = "Головна кнопка"
- Командна кнопка Command2: Caption = "Вийти" (оператор End)
- Рамка Frame1: Caption = "" (містить два залежні перемикачі)
- Залежний перемикач Option1: Value=1, Alignment = 0
  Caption = "Добавити нову кнопку"
- Залежний перемикач Option2: Value=0, Alignment = 0
  Caption ="Видалити кнопку"

```
Private Sub Command1_Click(Index As Integer)
 Static nCount As Integer
 If Option1.Value Then
  nCount = nCount + 1
  Load Command1(nCount)
  Command1(nCount).Caption = "Кнопка" & Str(nCount)
  Command1(nCount).Visible = True
  Command1(nCount).Top = nCount * 500
```

```
 Else
  Unload Commandl(Index)
 End If
End Sub
```

Досить часто командні кнопки чи інші елементи керування об'єднують у масив з метою спрощення опрацювання однотипних подій. Масиви елементів керування дають змогу значно скоротити обсяг коду, який необхідно написати для керування програмою.

Проект 9. *Набір натурального числа за допомогою кнопок*. Розглянемо панель мікрокалькулятора, на якій будуть реалізовані функції: *очищення операнда* (поля `Textl` з властивістю `Alignment`=1) і *набір натурального значення* у цьому вікні за допомогою цифрових кнопок. Спочатку розмістимо на формі рамку (`Framel`). У цій рамці розмістимо десять командних кнопок (`Commandl`, … , `Commandl0`). Вирівняємо їхні розміри і замінимо стандартні заголовки на цифри від 0 до 9. Далі замінимо назви усіх кнопок (властивість `Name`) на одну і ту ж назву `cif`. Для першої кнопки така заміна відбувається без зауважень. Для другої – Visual Basic попередить про існування об'єкта з такою назвою і запитає, чи необхідно створити масив елементів керування. Оскільки ми дійсно хочемо створити масив командних кнопок, то дамо ствердну відповідь. Наступні перейменування кнопок відбудуться без зауважень. Перейменування здійснюватимемо у порядку зростання цифр, тоді значення заголовка кнопки та властивості `Index` збігатимуться, а це даватиме змогу спростити процедуру опрацювання події `Click`, яка обслуговуватиме одночасно усі кнопки:

```
Private Sub cif_Click(Index As Integer)
 If Len(Textl."Text) = 0 And Index = 0 Then Exit Sub
 Textl.Text = Textl.Text & Chr(Asc(0) + Index)
End Sub
```

Кнопка `Commandl0` (заголовок *Скинути)* реалізує очищення операнда:

```
Private Sub Commandll_Click()
 Textl.Text = ""
End Sub
```

9.8. Елементи керування для роботи з дисками, каталогами і файлами

Елемент керування `DriveListBox` слугує для відображення списку всіх доступних дисків і пристроїв системи і забезпечує можливість їхнього вибору. Найважливішою подією цього елемента є подія `Change`, яка викликається зі зміною носія даних. Елемент `DriveListBox` має майже усі властивості звичайного поля зі списком. Однак здебільшого використовують тільки властивість `Drive`, що повертає обраний диск чи пристрій.

Елемент керування `DirectoryListBox` чи коротко `Dir-ListBox` відображає структуру обраного диска і дає змогу здійснювати вибір і зміну каталогу. Для цього елемента також головну роль відіграє подія `Change`, викликана унаслідок подвійного клацання мишею на наш каталог у вікні перегляду.

Елемент керування `DirListBox` також має деяку подібність зі списком. Однак головною його властивістю є властивість `Path`, що повертає повний шлях до обраного каталогу разом з назвою диска (наприклад, `C: \D\My`).

Після додавання у форму елементів керування `DriveList-Box` і `DirListBox` вони ще не працюють спільно. Тобто в один і той самий момент у `DriveListBox` може відображатися назва диска C, а в `DirListBox` – структура каталогів диска `D`. Тому перед використанням цих елементів їх необхідно синхронізувати у процедурі опрацювання події `Change` для `DriveListBox`:

```
Private Sub Drivel_Change()
  Dirl.Path = Drivel.Drive
End Sub
```

Зазвичай для вибору каталогу користувач натискає клавішу [*Enter*]. Однак елемент керування `DirListBox` ігнорує цю клавішу. Рішенням такої проблеми є можливість опрацювання події `KeyPress` і програмна зміна каталогу:

```
Private Sub Dirl_KeyPress(KeyAscii As Integer)
  If  KeyAscii =13 Then
```

```
    Dirl.Path = Dirl.List(Dirl.Listlndex)
  End If
End Sub
```

Елемент керування FileListBox відображає файли поточного каталогу, звідкіля їх можна вибирати. Для FileListBox головною подією є подія Click, що викликається при виборі користувачем назви файла у списку. Визначальними є також події PathChange (відбувається після зміни шляху – властивість Path) і PatternChange (відбувається після зміни маски вибору файлів – властивість Pattern).

Головною властивістю FileListBox є властивість FileName, що містить назву обраного файла (наприклад book.doc). Властивість Pattern дає змогу визначити типи тих файлів, що повинні відображатися у списку. Наприклад, для відображення файлів з розширенням *.ico і *.bmp необхідний такий код:

```
Filel.Pattern = "*.ico;*.bmp"
```

Зверніть увагу, що розширення файлів розділяються крапкою з комою. Список файлів також повинен синхронізуватися з обраним пристроєм і каталогом. Це відбувається унаслідок обробки події Change для DirListBox. Щодо цього використовується властивість Path елемента FileListBox:

```
Private Sub Dirl_Change ()
   Filel.Path = Dirl.Path
End Sub
```

Оскільки елемент DirListBox уже синхронізований з вибором диска, усі три елементи тепер працюють разом. Для відображення вичерпної назви файла, включаючи шлях, необхідно просто скласти відповідні рядки, що містять значення назви диска, шляху і назви файла. Помістити символ "\" між шляхом і назвою файла досить просто. Це виконується таким оператором:

```
IblPath.Caption = Filel.Path & "\" & Filel.Filename
```

Щоб уникнути відображення у шляху зайвої кількості символів "\" (наприклад, у випадку вибору файла кореневого каталогу), не-

обхідно дещо змінити код:

```
Private Sub Filel_Click()
 If Right(Filel.Path, I) = "\" Then
  IblPath.Caption = Filel.Path & Filel.Filename
 Else
  IblPath.Caption = Filel.Path & "\" & Filel.Filename
End If
End Sub
```

9.9. Створення меню

Menu Editor (Редактор меню) дає змогу створити нове меню і рядки меню, додавати в меню нові команди, замінювати команди меню власними командами, а також змінювати і видаляти наявне меню і рядки меню. Menu Editor можна викликати командою *Menu Editor* з меню *Tools* (Інструменти) або натисненням кнопки *Menu Editor* на панелі інструментів.

Працюючи з Menu Editor, можна встановити значну кількість властивостей меню, тоді як усі властивості меню доступні у вікні *Properties* (Властивості). Дві найважливіші властивості меню:

- `Name` (*назва*) – значення цієї властивості використовують для посилання на меню з програми.
- `Caption` (*напис*) – це текст, який з'являється на елементі меню.

У списку в нижній частині вікна Menu Editor перелічено елементи керування меню поточної форми. При наборі на клавіатурі назви пункту меню у текстовому полі `Caption` (Напис) цей пункт також з'являється у списку елементів керування меню. Вибравши наявний елемент керування меню у цьому списку, можна відредагувати його властивості.

Положення елемента керування меню у списку елементів керування меню визначає, чи є певний елемент керування заголовком меню (*menu title*), пунктом меню (*menu item*), заголовком підменю (*submenu title*) чи пунктом підменю (*submenu item*):

- крайній лівий елемент керування меню відображається на смузі меню як *заголовок меню*;

- елемент керування меню, розташований під заголовком меню і зміщений праворуч на одну позицію, є *пунктом меню*;
- пункт меню, услід за яким розташовані елементи керування, зміщені праворуч ще на одну позицію, називають *заголовком підменю*;
- елементи керування меню, зміщені праворуч і розташовані нижче за заголовок підменю, є *пунктами* цього *підменю*.

Елемент керування, властивість `Caption` якого має значення `"-"` (дефіс), з'являється як роздільник (*separator bar*). Роздільник слугує для відособлення логічних груп пунктів меню.

Примітка. Елемент керування не може бути роздільником, якщо він є заголовком меню, має підменю, позначений чи вимкнений або має призначену клавішу швидкого доступу. Хоч роздільники і створюються як елементи управління меню, вони не відповідають на подію `Click` і не можуть обиратися.

Щоб створити елементи керування меню в Menu Editor, необхідно:

1. Вибрати форму.
2. Вибрати команду **Menu Editor** з меню **Tools** або натиснути кнопку на панелі інструментів.
3. У текстовому полі `Caption` набрати текст для першого заголовка меню, що з'являється в рядку меню. Вмістити амперсанд (&) перед літерою, яка слугуватиме клавішею доступу для цього меню. Цю літеру буде підкреслено при відображенні напису меню. Текст заголовка меню відображається у списку елементів меню.
4. У текстовому полі `Name` набрати назву, за допомогою якої можна буде посилатися на елемент керування меню у програмі.
5. Натиснути кнопку зі стрілкою ліворуч або праворуч для зміни рівня зміщення елемента керування.
6. Встановити за бажанням інші властивості елемента керування. Це можна зробити в Menu Editor або згодом у вікні *Properties*.
7. Натиснути кнопку *Next* (Наступний) для створення іншого елемента керування або натиснути кнопку *Insert* (Вставити) для додання елемента керування до існуючих. Кнопками зі стрілка-

ми вгору і вниз можна пересувати елемент керування між існуючими елементами.

8. При завершенні створення елементів керування меню щодо цієї форми натиснути кнопку OK, щоб закрити Menu Editor.
9. Створені заголовки меню відображаються на формі. Під час розробки при натисненні кнопкою миші на заголовку меню розкривається список команд цього меню.

Для вибору команд меню за допомогою клавіатури можна визначити клавіші доступу і швидкі клавіші. За допомогою клавіш доступу (*access keys*) користувач може відкрити меню, втримуючи клавішу <*Alt*> і набираючи на клавіатурі призначений символ для меню. У відкритому меню користувач може обрати команду, натиснувши на клавіатурі клавішу з літерою, призначеною цій команді (клавішу доступу). Наприклад, комбінація клавіш <*Alt*>+<*E*> може відкривати меню *Edit*, а натиснувши клавішу <*P*>, можна вибрати команду *Paste* (Вставити) цього меню. Літера назви команди, що відповідає клавіші доступу, підкреслена.

Примітка. У меню не можна призначати однакові клавіші доступу різним командам, вони просто не працюватимуть.

Швидкі клавіші (*shortcut keys*) дають змогу виконати команду меню, не відкриваючи його. Швидкі клавіші – це комбінація клавіші керування (*Ctrl*) і функціональних клавіш чи клавіш символів, наприклад, <*Ctrl*> + <*F1*>або <*Ctrl*> + <*A*>. Вони з'являються в меню праворуч від відповідної команди. Вибрати функціональну клавішу або комбінацію клавіш можна у комбінованому вікні *Shortcut* (Швидкі клавіші).

Коли користувач обирає елемент керування меню, відбувається подія `Click`. Для кожного елемента меню у програму необхідно вмістити процедуру обробки події `Click`. Усі елементи меню, за винятком роздільників і недоступних або невидимих елементів меню, розпізнають подію `Click`.

Код процедури обробки подій для елементів керування меню нічим не відрізняється від коду процедури обробки події для інших елементів керування. Наприклад, код процедури опрацювання події `Click` для команди *Close* (Закрити) меню *File* може виглядати так:

```
Sub mnuFileClose_Click(): UnLoad Me : End Sub
```

Контекстне меню (pop-up menu, context menu) – це меню, яке відображається у будь-якому місці форми і не прив'язане до рядка. Набір його команд залежить від положення покажчика миші у момент натиснення правої кнопки миші, тому таке меню і називають контекстно-залежним. У Microsoft Windows його активізують натисненням правої кнопки миші.

Будь-яке меню, що має принаймні одну команду, можна відобразити під час виконання як контекстне меню. Щоб відобразити контекстне меню, використовують метод `PopupMenu`. Синтаксис його такий:

```
[object.]PopupMenu menuname [, flags [, x  [, y [,
                                  boldcommand]]]]
```

Параметр `flags` необхідний для подальшого визначення положення і поведінки контекстного меню. Параметр `boldcommand` задає назву елемента у контекстному меню, який буде виокремлено напівжирним шрифтом. Тільки один елемент у контекстному меню можна відобразити напівжирним шрифтом.

У наступному прикладі відображається меню з назвою `mnuFile`, коли користувач клацає на формі правою кнопкою миші. Можна використати поділ `MouseUp` або `MouseDown`, щоб визначити, коли користувач натискає праву кнопку миші:

```
Private Sub orm_MouseUp(Button As Integer, Shift As _
                    Integer, X As Single, Y As Single)
 If Button = 2 Then ' Натиснення правою кнопкою миші
  PopupMenu mnuFile 'Показати меню File як конт.меню
 End If
End Sub
```

Будь-який код, наступний за викликом методу `PopupMenu`, не виконується, доки користувач не обере будь-яку команду меню або не вийде з меню. Часто контекстне меню використовують для доступу до опцій, недоступних у рядку меню (у цих меню прапорець `Visible` у Menu Editor скинутий). Коли Visual Basic відображає контекстне меню, він ігнорує його властивість `Visible`.

9.10. Робота з буфером обміну

Текстові поля і комбіновані вікна мають кілька властивостей, що стосуються виокремленого тексту. Ці властивості (табл. 9.5), які посилаються на блок тексту, виокремленого в елементі управління, дають змогу створювати функції вирізування і вставлення для користувача. Властивості, перелічені в таблиці 9.5, можна використовувати й під час виконання програми.

Таблиця 9.5. Властивості виокремленого тексту

Властивість	Опис
`SelStart`	Довге ціле, яке задає початкове положення виокремленого блоку тексту. Якщо текст не виокремлено, ця властивість задає положення точки вставки. Значення 0 задає положення перед першим символом у елементі керування. Значення, яке дорівнює довжині тексту в елементі керування, зазначає положення відразу за останніми символами в елементі керування
`SelLength`	Довге ціле, яке задає число виокремлених символів
`SelText`	Рядок, що містить виокремлені символи (або порожній рядок, якщо не виокремлено жодного символу)

Керувати виокремленням тексту можна установкою значень властивостей `SelStart` і `SelLength`. Наприклад, такі оператори виокремлюють увесь текст у текстовому полі:

```
Textl.SetFocus
 ' Початок виокремлення перед першим символом
Textl.SelStart = 0
 ' Виокремлення до кінця тексту
Textl.SelLength = Len(Textl.Text)
```

Якщо привласнити властивості `SelText` новий рядок, то він замінить виокремлений текст і поточна точка вставки заміниться на точку одразу ж за останнім символом введеного тексту. Наприклад, наступний оператор замінює виокремлений текст рядком "*Мене щойно вставили*!":

```
Textl.SelText = "Мене щойно вставили!"
```

Якщо не виокремлено жодного тексту, рядок просто вставляється в текстове поле в поточній позиції.

Об'єкт `Clipboard` (буфер обміну – БО) не має ні властивостей, ні подій, однак він має декілька методів, які поділяють на три категорії:

- методи `GetText` (отримати текст з БО) і `SetText` (розмістити текст у БО) використовуються для переміщення тексту;
- методи `GetData` і `SetData` переміщують графіку;
- методи `GetFormat` і `Clear` працюють і з текстовими, і з графічними форматами.

Метод `SetText` копіює текст у Буфер обміну, замінюючи будь-який текст, що зберігався там раніше. Цей метод використовують як *оператор*. Його синтаксис:

```
Clipboard. SetText data[, format]
```

Метод GetText повертає текст, що зберігається у буфері обміну. Його використовують як *функцію*:

```
destination = Clipboard.GetText()
```

Комбінуючи ці два методи з властивостями виокремлення, наведеними вище, можна легко написати програмні коди для текстового поля: `Copy` (Копіювати), `Cut` (Вирізати) і `Paste` (Вставити). Узагальнені процедури (зберігаються у загальному модулі) реалізовують ці команди:

```
Sub EditCopyProcO
 ' Copy the selected text onto the Clipboard
 Clipboard.SetText Textl.SelText
End Sub
Sub EditCutProc()
 ' Copy the selected text onto the Clipboard
 Clipboard.SetText Textl.SelText
 ' Delete the selected text.
Textl.SelText = ""
End Sub
Sub EditPasteProc{)
 ' Place the text from the Clipboard
```

```
  Textl.SelText = Clipboard.GetText()
End Sub
```

Процедури опрацювання подій для відповідних елементів меню:

```
Private Sub mnuEditCopy_Click()
  EditCopyProc
End Sub
Private Sub mnuEditCut_Click()
  EditCutProc
End Sub
Private Sub mnuEditPaste_Click ()
  EditPasteProc
End Sub
```

У наведених кодах передбачено, що весь текст переміщується між конкретним текстовим полем `Text1` і буфером обміну. Однак операції копіювання, вирізування і вставлення тексту через БО можливі також між текстовим полем `Textl` та елементами керування на інших формах. Можна також переміщувати текст між текстовим полем і будь-яким додатком, що використовує буфер обміну.

Щоб команди `Copy`, `Cut` і `Paste` працювали з будь-яким текстовим полем, яке має фокус, необхідно використати властивість `ActiveControl` об'єкта `Screen`. Наступний код передбачає, що активний елемент управління є довільним текстовим полем:

```
Sub EditCopyProc()
 Clipboard.SetText Screen.ActiveControl.SelText
End Sub
Sub EditCutProc()
 Clipboard.SetText Screen.ActiveControl.SelText
 Screen.ActiveControl.SelText = ""
End Sub
Sub EditPasteProc()
Screen.ActiveControl.SelText = Clipboard.GetText()
End Sub
```

Наступний код надає узагальнені процедури `Copy` і `Paste`, які працюють з будь-яким стандартним елементом управління:

```
Sub EditCopyProc()
 If TypeOf Screen.ActiveControl Is TextBox Then
  Clipboard.SetText Screen.ActiveControl.SelText
 Elself TypeOf Screen.ActiveControl Is ComboBox Then
  Clipboard.SetText Screen.ActiveControl.Text
 Elself TypeOf Screen.ActiveControl Is PictureBox Then
  Clipboard.SetData Screen.ActiveControl.Picture
 Elself TypeOf Screen.ActiveControl Is ListBox Then
  Clipboard.SetText Screen.ActiveControl.Text
 Else
 'Ніяка дія не має значення для інших ЕК
 End If
End Sub
Sub EditPasteProc()
 If TypeOf Screen.ActiveControl Is TextBox Then
  Screen.ActiveControl.SelText = Clipboard.GetText()
 Elself TypeOf Screen.ActiveControl Is ComboBox Then
  Screen.ActiveControl.Text = Clipboard.GetText()
 Elself TypeOf Screen.ActiveControl Is PictureBox Then
  Screen.ActiveControl.Picture = Clipboard.GetData()
 Elself TypeOf Screen.ActiveControl Is ListBox Then
  Screen.ActiveControl.Addltem Clipboard.GetText()
 Else
 'Ніяка дія не має значення для інших ЕК
 End If
End Sub
```

Можна використати метод `GetFormat`, щоб визначити формат даних у буфері обміну даними. Наприклад, можна заблокувати команду `Paste` залежно від того, чи сумісні дані в буфері обміну з поточним активним елементом управління:

```
Private Sub imuPasteMod_Click()
 mnuCopy.Enabled = True : mnuCut.Enabled = True
 mnuPaste.Enabled = False
 If TypeOf Screen.ActiveControl Is TextBox Then
```

```
  If Clipboard.GetFormat(vbCFText) Then _
                                  mnuPaste.Enabled = True
 End If
End Sub
```

4. Як реалізовується мультивибір у `ListBox`?
6. Опишіть головні події роботи з клавіатурою.
10. Що таке таймер? З якою метою його використовують?
11. Що таке елемент керування `Image`? З якою метою його використовують?
12. Що таке елемент керування `Shape`? З якою метою його використовують?
13. Що таке рядок меню? Опишіть його складові.
14. Що таке спливаюче меню?

Завдання 9.1. Створити віконне застосування для табулювання функції, яка вибирається з *завдання 2.1* (с. 40). Значення функції обчислювати у вузлах сітки, що утворюється розбиттям відрізка $[a, b]$ на n рівних частин, і зберігати на формі у вигляді списку `ListBox`, що допускає мультивибір. Після цього реалізувати обчислення агрегованих величин:

1. Найменшого значення функції.
2. Найбільшого значення функції.
3. Середнього арифметичного значення функції.
4. Добуток від'ємних значень функції.
5. Кількість невід'ємних значень функції.

Вимоги до виконання індивідуального завдання:

- обчислювати агреговані величини серед обраних значень у

7. Опишіть команди контекстного меню. З якою метою їх використовують?
8. Що таке залежний перемикач? З якою метою його використовують?
9. Що таке незалежний перемикач? З якою метою його використовують?

Завдання для програмування

1. Що таке позначка? З якою метою її використовують?
2. Що таке панель? З якою метою її використовують?
3. Опишіть базові компоненти введення і відображення даних.

списку або в усіх точках – за домовленістю;
- при введенні користувачем числових значень попередньо перевіряти символи;
- передбачити опрацювання можливих помилок при виконанні математичних операцій та обчисленні стандартних математичних функцій;
- вибирати агреговані величини незалежними перемикачами;
- на формі навпроти обраних перемикачів мають бути відображені відповідні значення агрегованих величин; поля, розміщені навпроти невибраних перемикачів, мають бути невидимими.

Завдання 9.2. Завершити *приклад 9* (с. 188), створивши власний варіант електронного мікрокалькулятора. Передбачити також можливість уведення чисел (з попередньою перевіркою символів) з клавіатури.

Завдання 9.3. Увівши відповідні класи, створити віконне застосування для реалізації операцій над комплексними числами (студенти з непарними номерами у списку студентів групи) або над *раціональними* числами (студенти з парними номерами).

10. Програмування на VBA

План викладу матеріалу:

1. Загальна характеристика об'єктної моделі MS Office.
2. Властивості та методи головних об'єктів MS Excel.
3. Властивості та методи головних об'єктів MS Word.

Ключові терміни розділу

- *Об'єкт* `Application`
- *Колекція* `Workbooks`
- *Колекція* `Worksheets`
- *Колекція* `Documents`
- *Колекція* `Sentences`
- *Колекція* `Characters`
- *Об'єкт* `Range` *у MS Word*
- *Властивості* `Workbooks`
- *Властивості* `Range` *у Excel*
- *Властивості* `Documents`
- *Властивості* `Sentences`
- *Властивості* `Characters`
- *Властивості* `Selection`
- *Методи* `Worksheets`
- *Загальні методи колекцій*
- *Методи* `Range` *у Word*
- *Колекція* `Windows`
- *Колекція* `Sheets`
- *Об'єкт* `Range` *у MS Excel*
- *Колекція* `Paragraphs`
- *Колекція* `Words`
- *Об'єкт* `Selection`
- *Звертання до елемента колекції*
- *Властивості* `Worksheets`
- *Способи адресування комірок*
- *Властивості* `Paragraphs`
- *Властивості* `Words`
- *Властивості* `Range` *у Word*
- *Методи* `Workbooks`
- *Методи* `Range` *у Excel*
- *Методи* `Documents`
- *Методи роботи з курсором*

10.1. Загальна характеристика об'єктної моделі MS Office

Головна відмінність програм мовою Visual Basic for Applications (надалі — просто *VBA*) від програм на Visual Basic полягає у тому, що поряд зі звичайними змінними та константами, ці програми маніпулюють готовими об'єктами застосувань MS Office, такими, наприклад, як документи, абзаци, рядки й слова Word; або робочі книги, робочі аркуші та діапазони комірок Excel. Щоб писати програми на VBA, необхідно досить добре уявляти собі функціональні можливості таких об'єктів; властивості, якими вони володіють, методи впливу на ці об'єкти. Об'єктна модель Microsoft Office утворює досить складну ієрархію, і в навчальному посібнику подати її вичерпний опис неможливо: за цим краще звернутися до вбудованої довідки Microsoft Office або до додаткової довідкової

літератури. У посібнику зосередимося лише на головних об'єктах двох центральних застосувань Microsoft Office – Excel і Word, їхніх властивостях і методах.

Програмуючи на VBA, найчастіше необхідно впливати на дані, що зберігаються в документі, робочій книзі чи базі даних; змінювати текст, уміст комірок чи графічні об'єкти; відкривати чи зберігати документи, приховувати або показувати робочі аркуші в книзі Excel тощо. Такі змінювані елементи застосувань (*документи* Word: слова, абзаци, виноски, колонтитули тощо; *робочої книги* Excel: комірки, робочі аркуші, діаграми тощо), а також і самі застосування MS Office називають у VBA *об'єктами* (`Objects`).

У VBA є сотні найрізноманітніших об'єктів, чимало з яких поєднуються в *колекції* об'єктів. Перелічимо лише найважливіші об'єкти та їхні колекції в MS Excel і MS Word (табл. 10.1 і 10.2).

Таблиця 10.1. Об'єкти MS Excel

Об'єкт	Опис
1	2
`Application` (об'єкт "Додаток")	Представляє застосування (додаток) MS Excel загалом. Містить у собі глобальні встановлювані параметри, убудовані функції тощо
`Windows` (колекція "Вікна")	Використовують при згорненні/розгорненні вікна, розбитті його на частини й фіксуванні підвікон. Приналежний до цієї колекції об'єкт `ActiveWindow` представляє активне вікно
`Workbooks` (колекція "Робочі книги")	Містить у собі всі відкриті робочі книги. За допомогою методу `Open` можна відкрити ще одну робочу книгу. Метод `Add` створює нову робочу книгу. Метод `Close` закриває робочу книгу, а `Save` – зберігає. Об'єкт `ActiveWorkbook` цієї колекції представляє *активну* робочу книгу

Закінчення табл.10.1

1	2
`Sheets` (колекція “Аркуші”)	Містить у собі всі аркуші робочої книги – як звичайні робочі аркуші, так і аркуші діаграм. Найчастіше використовувані методи: `Add`, `Copy`, `Delete`, `Select`
`Worksheets` (колекція “Робочі аркуші”)	Містить у собі всі робочі аркуші робочої книги. Використовуються ті ж методи, що й для колекції `Sheets`. Об’єкти цієї колекції використовують при копіюванні або видаленні робочих аркушів, їхньому приховуванні чи показі, проведенні обчислень для формул робочого аркуша. Приналежний до цієї колекції об’єкт `ActiveWorksheet` представляє активний робочий аркуш
`Range` (об’єкт “Діапазон”)	Дає змогу змінювати властивості діапазону комірок. Приналежний до цього класу об’єкт `ActiveCell` представляє активну комірку. Зверніть увагу на те, що не існує такого об’єкта, як *комірка* (`Cell`). Комірка – це частковий випадок об’єкта `Range`

Таблиця 10.2. Об’єкти MS Word

Об’єкт	Опис
1	2
`Application` (об’єкт “Додаток”)	Представляє застосування MS Word загалом
`Windows` (колекція “Вікна”)	Колекція об’єктів схожа на однойменну колекцію об’єктів у MS Excel
`Documents` (колекція “Документи”)	Усі відкриті документи у застосуванні MS Word. До колекції `Documents` застосовують методи `Add`, `Open`, `Close`, `Save`. Приналежний до колекції об’єкт `ActiveDocument` представляє активний документ

Закінчення табл. 10.2

1	2
`Paragraphs` (колекція “Абзаци”)	Об’єкти колекції є специфічними об’єктами Word – *абзацами* документа. Можна з’ясувати загальну кількість абзаців, змінити їхнє форматування, додати нові абзаци тощо
`Range` (об’єкт “Фрагмент”)	Представляє собою *неперервний* фрагмент тексту в документі й визначається номерами початкового та кінцевого символів
`Selection` (об’єкт "Виділення")	Представляє собою *виокремлений* фрагмент тексту в активному документі або положення текстового курсору за відсутності виокремлення
`Sentences` (колекція “Речення”)	Колекція об’єктів представляє речення у документі, абзаці чи іншому фрагменті документа. Звернемо увагу, що не існує поодинокого об’єкта `Sentence`. Кожний елемент цієї колекції є об’єктом типу `Range`, що представляє одне речення. Щоб звернутися до конкретного речення, необхідно зазначити його номер. Наприклад, отримання першого речення активного документа: `ActiveDocument.Sentences(1)`
`Words` (колекція “Слова”)	Колекція об’єктів представляє слова в документі, абзаці чи іншому фрагменті документа. Як і для колекції `Sentences`, для колекції `Words` не існує поодинокого об’єкта `Word`. Кожен елемент цієї колекції є об’єктом типу `Range`, що представляє одне слово
`Characters` (колекція “Символи’)	Колекція об’єктів представляє символи в документі, абзаці чи іншому фрагменті документа. Як і для колекції `Sentences`, для колекції `Characters` не існує поодинокого об’єкта `Character`. Кожен елемент цієї колекції є об’єктом типу `Range`, що представляє один символ

Численні бібліотеки об'єктів MS Office задають каркас усіх документів, які можна побудувати у цьому середовищі. Коли створюється новий документ, наприклад, робоча книга MS Excel, то за домовленістю зі всієї сукупності бібліотек вибирається декілька, об'єкти яких і становлять *каркас документа*. Каркас, що створюється за домовленістю у той момент, коли відкривається нова робоча книга, складається з об'єктів, зачислених до таких бібліотек:

- *Excel* – бібліотека, яка задає основу документів MS Excel. Тут зберігається клас, що визначає кореневий об'єкт `Excel.Application`, і всі класи об'єктів, у нього вкладених.
- *Office* – бібліотека класів, спільних для всіх додатків MS Office (інструментальні панелі, об'єкти `Assistant` і `Answer Wizard` тощо).
- *Stdole* – бібліотека класів, що дає змогу працювати з OLE-об'єктами.
- *VBA* – бібліотека класів, пов'язаних з мовою VBA. Тут зберігаються всі стандартні функції і константи, класи `Collection` і `ErrObject`.
- *VBAProject* – проект за домовленістю, зв'язаний з документом.

Якщо порівняти каркас робочої книги MS Excel з каркасом документа MS Word, то вони відрізняються тим, що в основі одного лежить бібліотека ***Excel***, а в основі іншого – бібліотека ***Word.*** Ці бібліотеки містять специфічні для даних додатків об'єкти. Бібліотеки *Office*, *Stdole*, *VBA* – це спільні для всіх додатків MS Office бібліотеки.

В об'єктній моделі MS Excel та інших додатках MS Office об'єкти пов'язані між собою відношенням вбудування. На нульовому рівні ієрархії існує деякий центральний об'єкт, в який вбудовано інші об'єкти, що становлять перший рівень ієрархії. У кожний з об'єктів першого і наступних рівнів можуть бути вбудовані об'єкти наступного рівня. Вбудування об'єктів реалізують за допомогою *властивостей-учасників* відповідних об'єктів.

При посиланні на об'єкт у програмі іноді доводиться для уточнення задавати всі об'єкти, що лежать на ієрархічному шляху до необхідного об'єкта. У цьому випадку назви об'єктів відокремлюють один від одного крапкою. Наприклад, щоб присвоїти значення 3

першій комірці другої сторінки четвертої відкритої робочої книги Excel, можна використати такий оператор VBA:

```
Application.Workbooks(4).Worksheets(2).Range("A1").Value = 3
```

У цьому прикладі використовують не тільки прості об'єкти, а також об'єкти, що є колекціями (`Collections`) – **`Workbooks`** й **`Worksheets`** (у вбудованій довідці на ієрархічному дереві об'єктів прості об'єкти пофарбовані бірюзовим кольором, а об'єкти, що є також і колекціями – жовтим). У такому випадку в дужках після назви колекції зазначають назву або індекс конкретного елемента колекції.

Окремі об'єкти, що входять у колекцію, називають *елементами* колекції. На окремі елементи колекції можна посилатися тільки за допомогою одного з двох способів:

- указуючи в дужках *назву* конкретного об'єкта колекції;
- указуючи в дужках *індекс* (порядковий номер) елемента в колекції (нумерація елементів у колекції починається з одиниці).

Отже, об'єкти, які утворюють колекцію, окремо поза колекцією не існують; їх можна ідентифікувати тільки через елемент колекції. Коли кажуть/пишуть "об'єкт `Workbook`", то мають на увазі елемент колекції `Workbooks`.

Наприклад, оператор

```
Workbooks("Моякнига.xls").Close
```

закриває робочу книгу під назвою `Моякнига`. А от приклад використання індексу елемента колекції. Оператор

```
ActiveSheet.Lines(1).Select
```

виокремлює першу з наявних ліній на активній робочій сторінці.

Хоча колекції – це *групи об'єктів*, самі колекції також є одиночними об'єктами. Такий збірний об'єкт-колекція представляє всю сукупність об'єктів, і має власні властивості й методи, за допомогою яких можна змінювати разом стан усіх об'єктів колекції. Наприклад, оператор

```
ActiveSheet.Lines.Delete
```

вилучає усі намальовані на активному робочому аркуші прямі лінії.

Зазвичай індивідуальні об'єкти, що є елементами колекції,

мають набагато більше властивостей і методів, ніж відповідний збірний об'єкт-колекція. Наприклад, об'єкт-колекція `Workbooks` у Excel має всього п'ять властивостей (`Application, Count, Creator, Item, Parent`) і чотири методи (`Add, Close, Open, OpenText`), водночас об'єкт `Workbook` має 59 властивостей і 42 методи.

Не всі об'єкти додатків можуть групуватися у колекції – для деяких індивідуальних об'єктів не існує відповідних колекцій.

Зазвичай при звертанні до об'єкта не доцільно вичерпно задавати весь ієрархічний шлях, часто достатньо вибрати шлях за домовленістю. Можна використовувати й спеціальний оператор **`With`** щодо уточнення шляху відразу для кількох операторів.

10.2. Властивості та методи головних об'єктів MS Excel

10.2.1. Властивості-учасники об'єкта Excel.Application. Розглянемо властивості-учасники об'єкта `Excel.Application`, що повертають головні об'єкти, специфічні для MS Excel:

- **`WorkBooks`** – колекція відкритих у MS Excel робочих книг.
- **`Windows`** – колекція відкритих вікон в усіх робочих книгах. Одну і ту ж робочу книгу часто відкривають у декількох вікнах, щоб бачити її різні ділянки. Така колекція дає змогу отримати доступ до кожного такого вікна.
- **`WorkSheetPunction`** – об'єкт-контейнер з функціями MS Excel, які застосовують при обчисленнях на робочих сторінках.
- **`Addlns`** – колекція компонент, які розширюють можливості розв'язання спеціальних задач у MS Excel.
- **`Dialogs`** – колекція стандартних діалогових вікон.
- **`Names`** – колекція назв у відкритих робочих книгах MS Excel.
- **`RecentFiles`** – колекція назв файлів робочих книг, використаних недавно.

Далі коротко оглянемо властивості-учасники, які повертають вкладені об'єкти `Excel.Application`, які є доступними на верхньому рівні:

- **`ActiveWorkbook`**, **`ActiveWindow`**, **`ActiveSheet`**, **`ActiveChart`**, **`ActiveCell`** – повертають, відповідно, активну робочу

книгу, активне вікно, активну сторінку, активну діаграму та активну комірку, якщо такі існують у момент виклику відповідної властивості. За відсутності активного об'єкта, що запитується, виникне помилка.

- **Sheets**, **Charts** – повертають, відповідно, колекції робочих сторінок і сторінок діаграм активної робочої книги. Якщо активної робочої книги немає, то виникає помилка.
- **Rows**, **Columns**, **Cells**, **Range** – повертають колекції рядків, стовпців, комірок або задану область активної робочої сторінки. Якщо активної робочої сторінки немає, то виникає помилка.
- **Selection** – повертає виокремлений об'єкт в активному вікні. Тип об'єкта, що повертається, залежить від поточного виокремлення. Якщо в активному вікні немає виокремленого об'єкта, то повертається **Nothing**.
- **ThisWorkbook** – повертає поточну робочу книгу, що містить макрос, один з операторів якого і викликав цю властивість.

10.2.2. Атомарні властивості об'єкта `Excel.Application`. Додаток MS Excel, як і інші додатки MS Office, можна налаштувати користувачем на свій розсуд. Це налаштування можна виконати вручну (команда Параметри з меню Сервіс), а можна і програмно. Для програмного налаштування використовують атомарні властивості `Excel.Application`, – у цьому їхнє головне призначення. Наведемо вибірковий опис деяких груп атомарних властивостей:

- **DefaultFilePath**, **DefaultSaveFormat**, **DefaultSheetDirection** – задають, відповідно, шлях за домовленістю, формат за домовленістю, напрям перегляду тексту.
- D**isplayAlerts**, **DisplayCommentlndicator**, **DisplayFormulaBar**, **DisplaystatusBar** – булеві властивості, що дають змогу увімкнути/вимкнути видачу на екран, відповідно, повідомлень у процесі роботи макросів, спеціальний індикатор при показі коментарів, показ панелі формул і рядка статусу.
- **EnableAnimations**, **EnableAutoComplete**, **EnableEvents**, **EnableSound** – булеві властивості, що вмикають/вимикають, відповідно, анімацію при додаванні/видаленні рядків чи стовпців робочої сторінки, автозаповнення таблиці, звук тощо.

- **Height**, **Width**, **Left**, **Top** – задають висоту, ширину вікна і координати верхнього лівого кута вікна.
- Інші властивості, які дають змогу настроювати вигляд і поведінку курсору, скролінг, характеристики користувача і безліч інших параметрів.

10.2.3. Властивості та методи колекції Workbooks і об'єкта Workbook. Колекція **Workbooks**, що містить усі відкриті робочі книги, має стандартні для вбудованих колекцій властивості **Application**, **Count**, **Parent** та **Item**. Розглянемо найважливіші методи:

- **Add**([**Template**]) **As Workbook**

  – додає нову книгу в колекцію. Нову книгу створюють на основі шаблону, заданого параметром **Template**. Якщо його опущено, то використовують шаблон за домовленістю. Метод повертає створену робочу книгу.

- **Open**(**Filename As String**, [**Format**], ...) **As Workbook**

  – відкриває робочу книгу і додає її у колекцію. При відкритті можна задавати параметри, які визначають властивості книги. Обов'язковим параметром є тільки перший, що задає назву файла книги.

- **Close**()

  – закриває усі книги колекції. При закритті тієї чи іншої книги може з'явитися діалогове вікно з пропозицією зберегти зроблені зміни.

Книга складається зі сторінок. У термінах об'єктів це означає, що об'єкт **Workbook** має властивість **Sheets**, що повертає об'єкт **Sheets** - колекцію сторінок робочої книги. Оскільки робочі книги Excel містять сторінки різного типу, то нарівні з колекцією **Sheets** у об'єкта **Workbook** є властивості, що повертають колекції сторінок певного типу:

- **Worksheets** – робочих сторінок.
- **Charts** – сторінок, що містять діаграми.

Колекції **Worksheets** і **Charts** становлять у сукупності колекцію **Sheets**. здебільшого доводиться працювати з кожною із цих колекцій зокрема, проте іноді корисно мати змогу виконувати операції над усіма сторінками, незалежно від їхнього типу. Розглянемо інші властивості-учасники об'єкта **Workbook**:

- **ActiveSheet** – повертає активну сторінку робочої книги;
- **ActiveChart** – повертає об'єкт класу **Chart**; це може бути діаграма на сторінці діаграм або активна діаграма на активній робочій сторінці;
- **Names** – колекція назв робочої книги;
- **Container** – об'єкт, який містить робочу книгу (робоча книга може бути вбудована в інший об'єкт чи бути його частиною).

Атомарних властивостей об'єкта **Workbook** є значно більше. Серед них чимало булевих властивостей, які дають змогу вмикати/вимикати ту чи іншу властивість робочої книги.

Приклад 10.1. Виведення на екран назв робочої книги і шляху до неї:

```
Public Sub AllNames() ' Друк назв документа
 With Activeworkbook
   Debug.Print  " Властивість Name - ", Name
   Debug.Print  " Властивість CodeName - ", CodeName
   Debug.Print  " Властивість FullName - ", FullName
   Debug.Print  " Властивість Path - ", Path
 End With
End Sub
```

Ось як виглядають результати у вікні негайного виконання:

```
Властивість Name - BookTwo.xls
Властивість CodeName - ThisWorkbook
Властивість FullName - D:\Книга VisBasic\BookTwo.xls
Властивість Path - D:\Книга VisBasic\
```

Коротко охарактеризуємо деякі методи об'єкта **Workbook**. Створюються і відкриваються робочі книги методами **Add** і **Open** колекції **Workbooks**. А ось закриваються і зберігаються, використовуючи власні методи:

- **Save**, **SaveAs**, **SaveCopyAs** – дають змогу зберегти робочу книгу без видалення її з відповідної колекції.

- **Close** – виконує ті ж функції, що і **Save**, проте водночас закриває книгу і вилучає її з колекції.
- **Activate** – активізує робочу книгу.
- **Protect**, **ProtectSharing**, **Unprotect**, **UnprotectSharing** – дають змогу вмикати/вимикати паролі.
- **RunAutoMacros** – запускає на виконання автомакроси книги.

10.2.4. Властивості та методи колекції Worksheet. Колекція **Worksheets** містить об'єкти класу **Worksheet** – робочі сторінки електронних таблиць. За домовленістю при створенні кожної нової робочої книги до її складу зачислюють три такі сторінки. Об'єктно це означає, що унаслідок створення нової книги автоматично створюється колекція **Worksheets**, яка містить три елементи. Колекція **Worksheets** має типовий набір властивостей: **Application**, **Count**, **Parent**, **Item**. Крім цих властивостей є менш типова властивість для колекцій – властивість **Visible**, яка дає змогу зробити видимими або невидимими робочі сторінки книги. Найважливіші методи колекції **Worksheets**:

- **Function Add**([**Before**], [**After**], [**Count**]) **As Object**

  – дає змогу додати нову робочу сторінку до книги, повертаючи відповідний об'єкт як результат. Додана сторінка стає активною. Параметри **Before** і **After** зазначають, куди помістити сторінку, – перед чи після сторінки, яка до виконання методу була активною. Параметр **Count** дає змогу одночасно додавати декілька сторінок.

- **Sub Copy**([**Before**], [**After**]) – копіювання робочої сторінки.
- **Sub Delete**() – видалення колекції робочих сторінок.
- **Sub FillAcrossSheets**(**Range As Range**, [**Type As _**
  xlFillWith = xlFillWithAll])

  – область, задана параметром **Range**, копіюється у відповідне місце всіх робочих листів. Тип копіювання задається другим параметром (можна, наприклад, копіювати тільки формули чи форматування). За домовленістю копіюється весь вміст області, заданої параметром **Range**. Об'єкт, що копіюється, повинен бути частиною одного з робочих листів колекції.

Приклад 10.2. Копіювання діапазону комірок:

```
Public Sub CopyRange()
  ' Копіювання об'єкта Range першої сторінки
  ' на всі сторінки робочої книги
 With ThisWorkbook
  .Worksheets.FillAcrossSheets(.Worksheets(1) _
                               .Range("B9:E23"))
 End With
End Sub
```

- **Sub Move**([**Before**], [**After**]) – використовуються з метою переміщення листів. До колекції його краще не застосовувати.
- **Sub PrintPreview**, **Sub Printout** – використовують з метою попереднього перегляду колекції робочих листів перед друком і для друку колекції.
- **Sub Select** [**Replace**] – використовують з метою виділення сторінок колекції.

10.2.5. **Властивості та методи об'єкта Worksheet**. Об'єкт **Worksheet** (робоча сторінка) – головний тип сторінок робочої книги. Саме на цих сторінках здійснюються основні дії в комірках електронної таблиці. Головна особливість електронної таблиці полягає в тому, що в її комірки можна вводити не тільки дані, але й формули. При зміні даних електронної таблиці, ініційованих користувачем, зовнішнім посиланням або макросом програмного проекту, перераховуються усі формули.

З об'єктної точки зору окремі комірки електронної таблиці та області, що містять сукупності цих комірок, є об'єктами типу **Range**. Ці об'єкти є головними об'єктами робочої сторінки. Серед властивостей **Worksheet** передусім розглянемо властивості, які повертають деякий окремий об'єкт або колекцію об'єктів (властивості-учасники):

- **Range**(**Cell1**, [**Ce112**]) **As Range** – повертає об'єкт **Range**, що визначається кутовими комірками **Cell1** і **Ce112**.

- **Cells As Range** – повертає колекцію комірок електронної таблиці. Оскільки **Cells** одночасно є об'єктом **Range** і колекцією комірок, то можна використати індекси, щоб добратися до окремого елемента колекції (комірки електронної таблиці).
- **Rows As Range** і **Columns As Range** – повертають, відповідно, колекції рядків і стовпців таблиці. За індексом можна добратися до окремого рядка або стовпця таблиці. Водночас ці колекції є об'єктами **Range**, оскільки задають деяку область робочої сторінки.
- **UsedRange As Range** – повертає область робочої сторінки, що використовується. Зазвичай, лише незначна частина робочої сторінки зайнята даними, формулами, малюнками чи діаграмами. Властивість **UsedRange** дає змогу отримати мінімальну прямокутну область, що використовується.
- **Circular Reference As Range** – повертає об'єкт **Range**, що містить перше циклічне посилання на робочій сторінці. При відсутності такого посилання повертається значення **Nothing**.

Отже, одну і ту ж область таблиці (об'єкт **Range**) можна отримати різними способами. Наведемо приклад, що демонструє два способи отримання комірки "C5":

```
Debug.Print ActiveSheet.Range("C5")
Debug.Print ActiveSheet.Cells(5, 3)
```

У наступному прикладі робота йде над окремим стовпцем і рядком, але, фактично, й тут діє той самий об'єкт **Range**:

```
ActiveSheet.Columns(2).Value = "Так"
ActiveSheet.Rows( 1 ) .Value = "Hi"
ActiveSheet.Rows(1).Font.Bold = True
```

- **Shapes** – повертає колекцію, елементами якої є об'єкти класу **Shape**. Цю колекцію сформовано з об'єктів різних типів (малюнків, діаграм, графіків, вбудованих і пов'язаних об'єктів OLE, елементів керування тощо).
- **Names** – повертає колекцію назв робочої сторінки.
- **Comments** – повертає колекцію коментарів робочої сторінки.
- **QueryTables** – повертає колекцію, елементами якої є об'єкти

класу **QueryTable** (таблиці, отримані на основі запиту до зовнішнього джерела даних – бази даних, Web-сторінки тощо).

- **AutoFilter** – повертає об'єкт, який здійснює фільтрацію даних.
- **Next** і **Previous** – повертають, відповідно наступну і попередню сторінки робочої книги.
- **PageSetup** – задає параметри сторінки.

Розглянемо деякі атомарні властивості об'єкта **Worksheet**:

- **CodeName** – містить кодову назву робочої сторінки. Властивість має статус *тільки для читання;*
- **Name** – містить назву робочої сторінки.
- **EnableAutoFilter** – вмикає/вимикає стрілки автофільтрації на захищеній сторінці.
- **EnableCalculation** – вмикає/вимикає автоматичне переобчислення формул при зміні даних.
- **EnablePivotTable** – вмикає/вимикає елементи керування зведеною таблицею на захищеній сторінці.
- **EnableSelection** – вмикає/вимикає доступ комірок захищеної сторінки. Має три можливі значення: **xlNoRestrictions** (для вибору доступні всі комірки), **xlNoSelection** (всі комірки є недоступними для вибору) і **xlUnlockedCells** (відкритими є комірки, в яких властивість **Locked** має значення **False**).
- **StandardHeight** – повертає стандартну (за домовленістю) висоту всіх рядків. Властивість має статус *тільки для читання* (для всіх рядків одночасно змінити висоту не можна). Для кожного окремо взятого рядка можна змінити висоту, використовуючи властивість **RowHeight**.
- **StandardWidth** – повертає стандартну (за домовленістю) ширину всіх рядків. Користувач може змінювати цю властивість.
- **Type** – повертає тип робочої сторінки (**xlWorksheet**, **xlExcel4MacroSheet**, **xlExcel4IntlMacroSheet**). Властивість має статус *тільки для читання*.
- **Visible** – вмикає/вимикає видимість робочої сторінки. За-

галом – це булева властивість, що має значення **True**, якщо об'єкт видимий. Але програмно можна задати третє значення **xlSheetVeryHidden** – користувач не зуміє вручну добратися до прихованої робочої сторінки. Це здійснюють тільки програмно, змінивши значення цієї властивості на **True** або **False**.

Об'єкт **Worksheet** має чимало методів. Розглянемо спочатку методи, які є спільними для багатьох об'єктів і вже траплялися або ще будуть траплятися при описі інших об'єктів (це даватиме змогу нам уникнути зайвих подробиць):

- **Activate** – активізує робочу сторінку.
- **Delete** – видаляє робочу сторінку. Цей метод, як і нижченаведені методи **Copy**, **Move**, **Select** та інші, розглянуто при описі колекції робочих сторінок. Зазвичай саме при роботі з окремою сторінкою ці методи найчастіше застосовують.
- **Copy** – має два варіанти. У першому варіанті використовується без параметрів, копіюючи вміст робочої сторінки в буфер. У другому варіанті **Copy**(**Before**, **After**) створює копію сторінки, розміщуючи її перед або після сторінки, що викликала метод. Зрозуміло, що тільки один з двох параметрів – **Before** або **After** можна задати.
- **Move**(**Before**, **After**) – переміщує робочу сторінку, змінюючи порядок проходження сторінок у робочій книзі.
- **Paste**([**Destination**] [, **Link**]) – розташовує вміст буфера на робочу сторінку. Можливий параметр **Destination** типу **Range** – діапазон, в який буде розташовано вміст буфера. Другий можливий параметр **Link** набуває значення **True** у випадку, коли встановлюється зв'язок з джерелом даних (за домовленістю має значення **False**). Одночасно можна задати тільки один з цих параметрів.
- **PasteSpecial**(**Format**, **Link**, **DisplayAsIcon**, **IconFileName**, **Iconlndex**, **IconLabel**) – також розташовує вміст буфера в область виділення робочої сторінки. Різниця полягає у тому, що метод застосовується тоді, коли вміст буфера зберігається у спеціальному форматі, відмінному від форма-

ту Excel. Найчастіше метод застосовують для розміщення об'єктів інших додатків. Параметр **Format** типу рядка задає формат об'єкта, що зберігається у буфері. Параметр **Link** має те ж значення, що і в попередньому випадку. Іншу групу параметрів використовують тоді, коли об'єкт "приклеюється" у вигляді позначки: **DisplayAsIcon** має значення **True**, **IconFileName** задає назву файлу, що містить значки, **IconIndex** – індекс позначки у файлі, **IconLabel** – текст, пов'язаний з позначкою. Оскільки об'єкт або значок належить до області виокремлення, то така виокремлена область повинна бути встановлена заздалегідь.

- **Select**([**Replace**]) – створює об'єкт **Selection**. Можливий параметр **Replace** має значення **True**, якщо новий об'єкт замінює виокремлення, що раніше існувало, і **False**, коли відбувається розширення області виокремлення так, щоб вона охоплювала і новий об'єкт.
- **CheckSpelling** – перевіряє правопис робочої сторінки.
- **Protect** – захищає робочу сторінку від змін.
- **Unprotect** – відміняє захист робочої сторінки.

Розглянемо тепер методи, які раніше не траплялися. Здебільшого ці методи відображають специфіку MS Excel:

- **Calculate** – здійснює обчислення формул робочої сторінки. Зазвичай властивість **EnableCalculation** увімкнена і обчислення відбуваються автоматично. Проте при вимкненій властивості **EnableCalculation** необхідно застосовувати цей метод для ініціювання обчислень.
- **ClearArrows** – видаляє стрілки трасування, встановлені для перегляду залежності при обчисленнях. Для програмного задання трасування використовуються методи **ShowDependents** і **ShowPrecedents**, які є методами об'єкта **Range**.
- **Evaluate**(**Name**) – перетворює назву в об'єкт або значення. Цей метод зручно застосовувати, коли користувач вводить назву під час діалогу. Наприклад, користувач вводить назву комірки, а йому повертається її значення, або користувач задає деякий

вираз, що містить звернення до стандартних функцій, і отримує значення цього виразу.

- **PivotTableWizard** – створює зведену таблицю.
- **ResetAllPageBreaks** – відновлює початкове розбиття робочої сторінки на сторінки виведення.
- **SetBackgroundPicture**(**Filename**) – встановлює графічний фон для робочої сторінки або сторінки діаграм. Картинку для фону обирають з файла, назву якого задає F**ileName**.
- **ShowDataForm** – вказує на форму даних, пов'язану з даною робочою сторінкою.

10.2.6. Об'єкти Range і Selection. Об'єкти **Range** і **Selection** належать до групи схожих об'єктів, що трапляються у різних додатках MS Office. Це головні об'єкти, з якими доводиться працювати програмісту. У додатку MS Word є чітка логіка в тому, як створюються об'єкти **Range**. Об'єкти верхнього рівня, наприклад, **Document**, мають метод **Range**, що дає змогу створити новий діапазон. Об'єкти дещо нижчого рівня, наприклад **Paragraph**, мають властивість **Range**, що повертає діапазон, пов'язаний з об'єктом. У MS Excel ситуація інша. Об'єкти **Application** і **Worksheet** мають тільки властивість **Range**. Цю ж властивість має і сам об'єкт **Range**, який представляє об'єкти нижнього рівня аж до комірки. Синтаксис цієї властивості такий:

```
Property Range(Cell1 [, Ce112]) As Range
```

Об'єкт **Selection** у MS Excel виникає двояко: внаслідок роботи методу **Select**, або внаслідок виклику властивості **Selection**. Тип отриманого об'єкта може бути різним і визначається типом виокремленого об'єкта. Найчастіше об'єкт **Selection** належить класу **Range** і тоді при роботі з ним можна використати всі властивості та методи об'єктів класу **Range**.

Зазначимо, що **Range** є унікальним об'єктом – він може представляти як єдиний елемент таблиці, так і стовпець або рядок, деяку зв'язну і незв'язну прямокутну область, а також об'єднання і перетин усіх подібних елементів. Це ж саме стосується і об'єкта **Selection**. Параметри **Cell1** і **Ce112** мають складний синтаксис,

який дає змогу, відповідно, повернути об'єкт **Range** складної конфігурації. Якщо при виклику властивості **Range** використовується тільки один параметр, то **Cell1** може бути:

- назвою комірки, наприклад, – **"A1"**;
- діапазоном комірок, наприклад, – **"A1:B5"**;
- виразом над діапазонами, що містить операції об'єднання (кома) або перетину (пропуск), наприклад, – **"A1:B5, F1:G8"** або **"A1:B5 A3:G8"**.

Якщо задаються обидва параметри **Cell1** і **Ce112**, то вони визначають прямокутну область, задану найменшим лівим верхнім кутом і максимальним правим кутом діапазонів, що визначаються параметрами. У цьому випадку параметри можуть бути і змінними класу **Range**. Розглянемо декілька простих прикладів.

Приклад 10.3. Реалізація простих обчислень у комірках:

```
Public Sub Work3()
 ThisWorkbook.Worksheets("Sheetl").Activate
 Range("C4").Value = 5
 Range("C5").Formula = "=C4+2"
 Range("C6:C8").Formula = "=C4+C5-1"
End Sub
```

Властивості **Value** і **Formula** об'єкта **Range** у цьому контексті є властивостями за домовленістю, отож оператори присвоєння можна ще записати так:

```
Range("C4") = 5
Range("C5") = "=C4+2"
Range("C6:C8") = "=C4+C5-1"
```

Коли формула присвоюється діапазону комірок, то адреси комірок у формулі є відносними і змінюються при переході до наступної комірки діапазону. Отож формула, приписана комірці `C7`, матиме вигляд: `"=C5+C6-1"`.

Приклад 10.4. Відносність адрес комірок щодо *діапазону*:

```
Public Sub Work4()
  Dim myRange As Range
  Set myRange = Range("C1:C4")
  myRange.Range("A1") = 7            ' C1=7
  myRange.Range("B1") = 8            ' D1=8
  myRange.Range("A2") = "=A1+2"      ' C2=2
  myRange.Range("A3:A5") = "=A1+A2" ' C3:C5=0
End Sub
```

Спочатку створюється об'єкт `myRange` (діапазон `"C1:C4"`). Виклик `myRange.Range("A1")` визначає комірку з адресою, яка обчислюється відносно об'єкта `myRange` (комірка `C1`). У формулах правої частини `A1` і `A2` прив'язані до адрес робочої сторінки.

Приклад 10.5. Відносність адрес комірок щодо *виокремлення*:

```
Public Sub Work5()
 Range("D1").Select
 Selection.Range("A1") = 7            ' D1=7
 Selection.Range("A2") = "=C1+2"      ' D2=2
 Selection.Range("A3:A4") = "=C1+C2"  ' D3:D4=0
End Sub
```

Приклад 10.6. Виклик **`Range`** з двома параметрами:

```
Public Sub Work6()
 Dim myRange1 As Range
 Set myRange1 = Range("E1", "E6")
 Debug.Print myRange1.Count          ' 6
 myRange1.Range("A1") = 27           ' E1=27
 myRange1.Range("A2") = "=D1+2"      ' E2=2
 myRange1.Range("A3:A6") = "=D1+D2"  ' E3:E6=0
End Sub
```

10.2.7. **Способи адресування комірок**. У попередніх прикладах використано *відносне адресування* комірок у форматі `A1`. Адреса комірки у цьому форматі формується з назви стовпця (`A`, `B`, ..., `Z`,

AB, ...AZ, ..., HZ, IA, ...IV) і номера рядка (1. .65536). Адреса комірки є відносною (змінюється при копіюванні). Для задання абсолютної адреси комірки, яка не змінюєься при копіюванні, використовують символ "$". Наприклад: Z$10 – номер рядка 10 не змінюється при копіюванні; $Z10 – назва стовпця Z не змінюється при копіюванні; $Z$10 – абсолютна адреса комірки Z10.

Адреса комірки на робочій сторінці є лише частиною вичерпної адреси, яка налічує назву сторінки і назву книги. При заданні вичерпної адреси комірки назву сторінки завершує символ "!", а назву книги беруть у квадратні дужки.

Приклад 10.7. Використання вичерпної адреси комірки:

```
Public Sub Work7()
  Debug.Print Range("$A$3").Value
  Debug.Print Range("Sheet1!$A$3").Value
  Debug.Print Range("[BookOne.xls]Sheet1!$A$3").Value
End Sub
```

Для адресування комірок ще використовують формат **R1C1** – адреса задається індексом рядка (**Row**) та індексом стовпця (**Column**). І тут адреси комірок бувають абсолютними (задають індекси) і відносними (задають зміщення щодо активної комірки). Зміщення записують у квадратних дужках зі знаком, що показує напрям зміщення.

Приклад 10.8. Використання формату R1C1:

```
Public Sub Work8()
  Range("A3") = 11
  Range("A4") = "=R3Cl+5"            ' 16
  Range("A5:A6") = "=R[-2]+R[-1]"    ' 27; 43
End Sub
```

Звернемо увагу на дві обставини:

- При виклику **Range** його параметри можна задавати тільки у форматі **A1**. Тому в лівій частині збережено “старий спосіб” адресування комірок. У формулах застосоване адресування у форматі **R1C1**, щоб явно підкреслити відносний характер адрес.

- Якщо обчислення у формулах розповсюджуються на діапазон, пов'язаний з одним стовпцем чи одним рядком, то можна задавати адреси, використовуючи тільки один індекс. У прикладі задане зміщення за рядками, оскільки стовпець залишається незмінним і його можна не вказувати.

При створенні об'єктів **Range** не можна користуватися зміщенням – доступний тільки формат **A1**. Проте можна використати зміщення, щоб перейти від одного об'єкта **Range** до іншого. Досягається це методом **Offset**(**RowOffset**, **ColumnOffset**) об'єкта **Range**. Параметри методу задають зміщення нового об'єкта **Range** за рядками і стовпцями щодо заданого об'єкта **Range**.

Приклад 10.9. Створення діапазону, зміщеного відносно заданого:

```
Public Sub Work9()
 Set myRange = Range("Al:A4")
 Set myRangel = myRange.Offset(2, 3)
 myRange1.Select   ' Виділено D3:D6
End Sub
```

При звертанні до окремих комірок зручно використовувати властивість **Cells** робочої сторінки, яка повертає колекцію комірок.

Приклад 10.10. Табулювання функції:

```
Public Sub TabFun()
 Dim i As Integer, x As Single
 x = -1
 For i = 1 To 11
  ActiveSheet.Cells(1, i).Value = x
  ActiveSheet.Cells(2, i).Value = x * x - 3 * x + 2
  x = x + 0.5
 Next i
End Sub
```

10.2.8. Властивості та методи об'єкта Range. Про об'єкт **Range** можна говорити дуже довго – це основа Excel. Він має чималу кіль-

кість властивостей і методів, однак немає подій, оскільки події пов'язані з об'єктами, що стоять на дещо вищих рівнях ієрархії.

Зауважимо, що багато властивостей об'єкта **Range** збігаються з властивостями об'єкта **Worksheet**. У цьому немає нічого дивного, оскільки **Range** задає частину робочої сторінки, а властивості частини і цілого багато в чому збігаються. Зокрема, це властивості, що повертають об'єкт **Range**: **Range**, **Cells**, **Columns**, **Rows**. Зрозуміло, що за допомогою властивості **Range** можна виділити деякий діапазон не тільки на робочій сторінці, але і в будь-якій області, визначеній об'єктом **Range**. Це стосується й усіх інших властивостей, що повертають об'єкт **Range**.

Безліч властивостей об'єкта **Range** повертають єдиний об'єкт, водночас батьківський об'єкт **Worksheet** повертає всю колекцію (**Name**, **Comment**, **PivotTable**, **QueryTable**). На об'єкт **Range**, що повертає єдиний об'єкт, накладаються певні вимоги. Наприклад, щоб повернути коментар, необхідно, щоб об'єкт **Range** містив єдину комірку з коментарем. Зведена таблиця повинна містити верхній лівий кут об'єкта **Range**. Розглянемо деякі головні властивості, специфічні для об'єкта **Range**:

- **Address, AddressLocals** – рядок, що містить адресу об'єкта **Range**. Другий випадок – для локалізованих версій. Цю адресу можна задавати у форматах **A1** або **R1C1** як абсолютну чи відносну.
- **Borders** – повертає колекцію з чотирьох меж об'єкта **Range**. Дає змогу виділити кольором і/або товщиною лінії межі об'єкта.
- **Text** – повертає рядок тексту, пов'язаного з об'єктом **Range**. Використовують тільки для читання.
- **Characters** – повертає частину тексту, пов'язаного з об'єктом **Range**. Виділення частини тексту здійснюється за допомогою параметрів **Start** і **Length**. Отриманий текст можна змінювати.
- **Column**, **Row** – повертають, відповідно, номер першого стовпця чи першого рядка в області об'єкта **Range**.
- **Font** – повертає об'єкт **Font**, що використовується при написанні тексту в області об'єкта **Range**.

- **Formula**, **FormulaR1C1**, **FormulaArray**, **FormulaLocal**, **FormulaHidden**, **FormulaLabel**, **FormulaR1C1Local** – дають змогу прочитати або задати формулу в форматі **A1**, **R1C1**, формулу над масивами, формулу для локалізованих версій і т.д.
- **Locked** – повертає **True**, якщо об'єкт **Range** закритий для модифікацій.
- **Value** – значення комірки. Якщо вона порожня, то повертається значення **Empty**, що можна перевірити, викликавши функцію **IsEmpty**. Якщо об'єкт Range містить понад одну комірку, то повертається масив значень, що можна перевірити, викликавши функцію **IsArray**. Функції **IsNumber**, **IsText** дають змогу визначити тип значення, що зберігається у комірці.

Об'єкт **Range** має понад 80 різних методів. Спільних методів з об'єктом **Worksheet** є приблизно 10%. До таких методів зачислено методи загального призначення: **Activate**, **PrintOut**, **Calculate**, **CheckSpelling**, **Copy**, **Delete**, **PasteSpecial** та **Select**. Об'єкт **Range** має спільні методи не тільки з об'єктом **Worksheet**, але й зі старшим в ієрархії об'єктом **Workbook**. Метод **Run**, що дає змогу запускати макроси, є і у об'єкта **Range**. Перелічимо деякі групи методів, не наводячи їхнього докладного опису:

- **Clear** – методи групи реалізують різноманітні очищення змісту, коментарів та інших деталей в області об'єкта Range.
- **Copy** – методи групи виконують копіювання діапазону.
- **Fill** – методи групи виконують заповнення діапазону.
- **Find** – методи групи дають змогу здійснити різноманітний пошук в області об'єкта **Range**.
- **Show** – методи групи призначені для відображення даних на екрані дисплея.
- **Sort** – два методи групи виконують сортування даних в області об'єкта **Range**.

Залишається понад 40 методів, описати які неможливо за обсягом книги. Здебільшого вони відомі читачеві з досвіду роботи в MS Excel у "ручному" режимі (методи заповнення діапазону, форма-

тування комірок, робота з даними, робота зі списками, підбір параметрів тощо).

Наведемо короткий опис трьох методів:

- **Sub AutoFill**(**Destination As Range**, [**Type As _**
  XIAutoFillType = xlFillDefault])

  – заповнює діапазон, заданий параметром **Destination**, використовуючи значення об'єкта **Range** і тип заповнення, визначений параметром **Type**. Діапазон призначення **Destination** повинен містити у собі початковий об'єкт **Range**, що викликав метод.

- **Sub AutoFormat**([**Format As XlRangeAutoFormat = _**
  xlRangeAutoFormatClassicl], **[Number]**, [**Font**], **_**
  [**Alignment**], [**Border**], [**Pattern**], [**Width**])

  – вмикає автоматичне форматування діапазону. Тип форматування визначає перший параметр, інші параметри вмикають або вимикають ті чи інші можливості форматування. За домовленістю вони включені.

- **Sub DataSeries**([**Rowcol**], [**Type As XLDataSeriesType _**
  = xlDataSeriesLinear], [**Date As XlDataSeriesDate _**
  [xlDay], [**Step**], [**Stop**], [**Trend**])

  – автоматично створює послідовність даних у зазначеному діапазоні, що задовольняє певному закону побудови. Перший параметр має два можливих значення: **xlRows** і **xlColumns**, які задають спосіб заповнення даних – по рядках або стовпцях. Другий параметр задає тип заповнення послідовності даних. Інші параметри дають змогу керувати процесом заповнення даних.

Приклад 10.11. Використання методу **DataSeries**:

```
Public Sub Work11()
 Dim Sh As Worksheet, myr As Range
 Set Sh = ThisWorkbook.Worksheets(1)
 Set myr = Sh.Range("C21:C32")
 myr.Cells(1, 1) = "31-Jan-2004"
```

```
 myr.DataSeries Type:=xlChronological, Date:=xlMonth
 Set myr = Sh.Range("D21:D32")
 myr.Cells(1, 1) = 320
 myr.DataSeries Type:=xlDataSeriesLinear, Step:=20
End Sub
```

10.3. Властивості та методи головних об'єктів MS Word

Коротко наведемо найважливіші властивості головних об'єктів MS Word.

Об'єкт **Application**:

- **ActiveWindow** – повертає активне вікно;
- **ActiveDocument** – повертає активний документ;
- **AutoCorrect** – вмикає/вимикає автозаміну та автоформатування при уведенні тексту;
- **ScreenUpdating** – вмикає/вимикає режим оновлення екрана;
- **FontNames** – повертає назви всіх доступних шрифтів.

Об'єкт **Document** (елемент з колекції **Documents**):

- **ActiveWindow** – повертає активне вікно;
- **Comments** – повертає колекцію об'єктів "Примітки" для цього документа;
- **FullName** – повертає повну назву документа, включаючи шлях;
- **Name** – повертає назву документа;
- **Saved** – ознака того, що документ збережений на диску;
- **GrammarChecked** – ознака того, що треба виконувати перевірку граматики для цього документа.

Об'єкт **Paragraph** (абзац – елемент з колекції **Paragraphs**):

- **Alignment** – задає вирівнювання абзацу: за лівим краєм (**wdAlignParagraphLeft**), за правим краєм (**wdAlignParagraphRight**), за центром (**wdAlignParagraphCenter**), за шириною (**wdAlignParagraphJustify**);

- **Count** – повертає величину кількості абзаців у даній колекції;
- **First** – повертає перший з виділених абзаців;
- **FirstLineIndent** – задає відступ для першого рядка абзацу: додатнє число задає відступ вправо, від'ємне – уліво;
- **Last** – повертає останній з виділених абзаців;
- **Hypenation** – ознака виконання автоматичного розділення слів на склади при перенесенні на наступний рядок;
- **PageBreakBefore** – задає ознаку того, що абзац має бути першим абзацом на новій сторінці.

Конспективно розглянемо найважливіші методи головних об'єктів MS Word (табл. 10.3).

Таблиця 10.3. Методи об'єктів Word

Метод	Опис
Об'єкт **Application** (Додаток)	
CheckSpelling	Запускає перевірку правопису
Help	Відкриває убудовану довідку
Quit	Завершує роботу Word
Undo	Скасовує останню виконану дію
Об'єкт **Document** (Документ)	
Activate	Активізує документ
Close	Закриває документ
PrintPreview	Вмикає/вимикає режим попереднього перегляду
Save	Зберігає документ
Save As	Зберігає документ під іншою назвою
Об'єкт **Paragraph** (Абзац)	
Add	Додає новий, порожній абзац у документ
Indent	Збільшує відступ абзацу на один рівень
Outdent	Зменшує відступ на один рівень
Reset	Скасовує "ручне" форматування абзацу

Приклад 10.12. Збереження новоствореного документа у поточному каталозі з одночасним встановленням шрифту для його вмісту (об'єкт **Content**):

```
Public Sub work12()
 Dim doc As Document
 Set doc = Documents.Add
 With doc
  .Content.Font.Name = "Arial"
  .SaveAs FileName:="Uch11"
 End With
End Sub
```

При роботі з текстом найчастіше використовують об'єкт **Selection** (об'єкт "Виокремлення"), який представляє собою *виокремлений* фрагмент тексту в активному документі або положення текстового курсора при відсутності виокремлення.

Об'єкт **Range** подібний до об'єкта **Selection** з погляду його властивостей і методів. Відмінність полягає у тому, що **Range** – це неперервний діапазон тексту, що *не залежить* від поточного виокремлення і має визначену початкову (**Start**) та кінцеву (**End**) точки. Об'єкт **Range** має властивість **Text**, що повертає текст діапазону, і метод **Select** для виокремлення діапазону.

Об'єкт **Document** має метод **Range**, який створює діапазон за допомогою початкової (**Start**) і кінцевої точки (**End**) діапазону.

Приклад 10.13. Виділення перших 10 –ти символів тексту:

```
 Public Sub work13()
  Set mr = ActiveDocument.Range(Start:=0, End:=10)
  mr.Bold = True ' Виділення перших 10 –ти символів
 End Sub
```

Нагадаємо, що конкретні елементи колекцій **Words**, **Sentences** і **Characters** є об'єктами **Range**. Властивість **Range**, яку мають об'єкти **Paragraph** (абзац), **Bookmark** (позначка) і **Cell** (комірка таблиці), повертає діапазон на базі конкретного абзацу, позначки чи комірки таблиці.

Приклад 10.14. Виокремити діапазон на основі абзаців з другого по п'ятий включно:

```
Public Sub work14()
 Dim mr As Range, md As Document
 Set md = ActiveDocument
 Set mr = md.Range(Start:=md.Paragraphs(2).Range.Start, _
          End:=md.Paragraphs(5).Range.End)
 mr.Select
End Sub
```

Приклад 10.15. Виокремити діапазон на основі трьох слів:

```
Public Sub work15()
 Dim mr As Range, md As Document
 Set md = ActiveDocument
 Set mr = md.Range(Start:=md.Words(3).Start, _
          End:=md.Words(6).End)
  mr.Select
End Sub
```

Об'єкти **Range** і **Selection** мають чимало подібних власивостей і методів, оскільки виокремлений фрагмент тексту і є власне *діапазоном*. Ці об'єкти мають методи, які дають змогу:

- скопіювати/вирізати виокремлений фрагмент тексту (чи діапазон) у буфер обміну (методи **Copy**/**Cut**);
- вставити вмістиме буфера обміну в текст/діапазон (метод **Paste**);
- знаходити входження деякого фрагмента тексту в документі (метод **Find**) з можливістю його заміни на інший фрагмент;
- переміщувати курсор у документі тощо.

При застосуванні методу **Paste** виокремлений фрагмент тексту *замінюється* на вміст буфера обміну. Щоб не замінювати виокремлений фрагмент, виокремлення треба попередньо *скасувати* методом **Collapse**, задавши напрям переміщення курсору (параметр **Direction**) *перед* (значення **wdCollapseStart** – за домовленістю) або *після* виокремленого (значення **wdCollapseEnd**).

Приклад 10.16. Скопіювати перший абзац активного документа:

```
Public Sub work16()
  ActiveDocument.Paragraphs(1).Range.Copy
  Selection.Collapse
  Selection.Paste
 End Sub
```

Приклад 10.17. Скопіювати виокремлений фрагмент тексту активного документа в кінець тексту цього ж документа:

```
Public Sub work17()
  Dim mr As Range
  Selection.Copy
  Set mr = ActiveDocument.Content
  mr.Collapse direction:=wdCollapseEnd
  mr.Paste
End Sub
```

Властивість **Text** (властивість за домовленістю) об'єктів **Range** і **Selection** містить текст виокремленого фрагмента/діапазону. Присвоєння значення цій властивості дає змогу замінити виокремлений фрагмент/діапазон на новий рядок символів. Якщо **Selection** представляє курсор, то у документ просто вставляється новий рядок символів.

Приклад 10.18. Знаходження у виділеному фрагменті тексту активного документа всіх входжень слова "*мама*" і заміни його на слово "*тато*":

```
Public Sub work18()
 With Selection.Find
  .Text = "мама"
  .ClearFormatting
  .Replacement.Text = "тато"
  .Replacement.ClearFormatting
  .Execute Forward:=True, Replace:=wdReplaceAll
 End With : End Sub
```

Методи **InsertBefore**/**InsertAfter** дають змогу вставляти новий текст перед/після виділеного фрагмента тексту без його заміни. Метод **TypeText** вставляє новий текст на місці курсору. Якщо є виокремлення, то метод **TypeText** буде його замінювати (або не замінювати) залежно від глобальної ознаки *Заміняти виокремлений фрагмент* (закладка *Редагування* діалогового вікна команди *Параметри* у меню *Сервіс*).

Приклад 10.19. Вставити перед виокремленим фрагментом тексту активного документа рядок символів “"Новий текст" ” (з лапками) і перевести після цього курсор на кінець виокремленого фрагмента:

```
Public Sub work19()
 With Selection
  .InsertBefore Chr(34) & "Новий текст" & Chr(34) & Chr(32)
  .Collapse Direction:=wdCollapseEnd
 End With
End Sub
```

Методи **InsertParagraphBefore/InsertParagraphAfter** дають змогу вставляти новий порожній абзац перед/після виокремленого фрагмента тексту без його заміни. Метод **TypeParagraph** вставляє новий порожній абзац на місці курсору. Якщо є виокремлення, то метод **TypeParagraph** буде його замінювати.

Приклад 10.20. Вставити на початок активного документа рядок символів “Вступ ” (без лапок) як окремий абзац:

```
Public Sub work20()
 With ActiveDocument.Content
  .InsertParagraphBefore
  .InsertBefore "Вступ"
 End With : End Sub
```

Метод

Move([**Unit**], [**Count**]) **As Long**

об’єкта **Selection** пересуває курсор на задану кількість (**Count**) одиниць (**Unit**). Одиницями можуть бути: символи (wdCharac-

ter), слова (wdWord), речення (wdSentence), абзаци (wdParagrph) тощо. При виконанні цього методу об'єкт **Selection** звужується ("*колапсує*") до точки вставки, яка пересувається у напрямі до кінця документа (додатне значення **Count**) на задану величину одиниць **Count**. Для від'ємного значення **Count** – у напрямі до початку документа на задану величину одиниць |**Count**|. За домовленістю Unit **=** wdCharacter; Count=1. Метод **Move** оформлений у вигляді функції, що повертає кількість одиниць, на яку вдалося перемістити курсор (точку вставки).

Методи **EndOf**, **MoveLeft**, **MoveRight**, **MoveUp**, **MoveDown** використовують для розширення виділеного фрагмента (об'єкт **Selection**) або діапазону (об'єкт **Range**).

Приклад 10.21. Визначити кількість абзаців від поточного положення курсору до кінця документа:

```
Public Sub work18()
Dim n As Integer, ok As Integer
 Do
   ok = Selection.Move(wdParagraph, 1)
   If ok <> 0 Then n = n + 1
 Loop While ok = 1
 Debug.Print n
End Sub
```

Приклад 10.22. Виокремити напівтовстим накресленням перше слово кожного абзацу від поточного положення курсора до кінця документа:

```
Public Sub work22()
 Dim ok As Integer
 Do
  Selection.Words(1).Font.Bold = True
  ok = Selection.Move(wdParagraph, 1)
 Loop Until ok = 0
End Sub
```

Приклад 10.23. Підкреслити латинські слова у виокремленому фрагменті документа:

```
Public Sub latyn()
 Dim mr As Range
 If Selection.Start = Selection.End Then
  MsgBox "Не виділено фрагмент!", vbOKOnly, _
                        "виділення латинських слів"
  GoTo bye
 End If
 Set mr = Selection.Range
 ActiveDocument.Bookmarks("\StartOfSel").Select
 While ActiveDocument.Bookmarks("\Sel").Start < mr.End
  If Selection.Text >= "A" And Selection.Text <= "z" Then
    Selection.Words(1).Underline = True
  End If
  Selection.Move unit:=wdWord
 Wend
bye:

End Sub
```

Зауваження. У прикладі 10.23 використано стантартні позначки: `"\StartOfSel"` –початок виокремлення; `"\Sel"`– виокремлення або точка вставки.

? Запитання для самоперевірки

1. Коротко опишіть каркас робочої книги MS Excel.
2. Коротко опишіть каркас документа у MS Word.
3. Коротко опишіть головні властивості колекції `Windows`.
4. Коротко опишіть головні властивості колекції `Workbooks`.
5. Коротко опишіть головні властивості об'єкта `Workbook`.
6. Коротко опишіть головні властивості колекції `Worksheets`.
7. Коротко опишіть головні властивості об'єкта `Worksheet`.
8. Коротко опишіть головні властивості об'єкта `Range` у MS Excel.
9. Коротко опишіть головні методи колекції `Workbooks`.
10. Коротко опишіть головні методи `Workbook`.
11. Коротко опишіть головні методи об'єкта `Worksheet`.

12. Коротко опишіть головні методи об'єкта `Range` у MS Excel.
13. Що таке відносне/ абсолютне адресування комірок?
14. Коротко опишіть головні властивості об'єкта `Document`.
15. Коротко опишіть головні властивості об'єкта `Range` у Word.
16. Коротко опишіть головні властивості об'єкта `Selection`.
17. Коротко опишіть головні методи об'єкта `Document`.
18. Коротко опишіть головні методи об'єкта `Range` у Word.
19. Коротко опишіть головні методи об'єкта `Selection`.

💻 Завдання для програмування

Завдання 10.1. Створити макрос (підпрограму), який даватиме змогу сформувати на робочому аркуші MS Excel арифметичну прогресію. Відредагувати макрос для введення у діалоговому режимі першого члена та різниці прогресії. Додати на власну панель інструментів кнопку виклику цього макроса. Протестувати макрос.

Завдання 10.2. Створити макрос, який даватиме змогу сформувати на робочому аркуші MS Excel графік певної функції. Діапазон комірок для побудови графіка необхідно сформувати до початку побудови графіка функції. Додати на власну панель інструментів кнопку виклику цього макроса. Протестувати макрос.

Завдання 10.3. На основі аналізу макросів завдань 1 і 2 створити макрос, який даватиме змогу сформувати на робочому аркуші MS Excel графік певної функції. Діапазон комірок для побудови графіка має формуватися самим макросом, тобто у діалоговому режимі макроса треба передати: вигляд функції, початкове і кінцеве значення, крок (або вигляд функції, початкове значення, крок і кількість точок). Додати на власну панель інструментів кнопку виклику цього макроса. Протестувати макрос.

Завдання 10.4. Створити макрос, який даватиме змогу вставляти у текст спеціальний символ-позначку із *Wingdings* (наприклад, кнопку) з обох боків обрамлену пропуском. Додати на власну панель інструментів кнопку виклику цього макроса. Протестувати модуль. Відредагувати макрос для виведення підказок користувачу.

Завдання 10.5. Створити два макроси, які даватимуть змогу виділити напівжирним накресленням частину рядка тексту від текстового курсора до початку (кінця) рядка. Додати на власну панель інструментів кнопки виклику цих макросів. Протестувати модулі. Відредагувати макрос для виведення підказок користувачу.

Завдання 10.6. На основі макросів, створених у завданні 2, записати макрос, який об'єднує в собі можливості обох цих макросів. Напрям виділення має задавати користувач за допомогою діалогового вікна. Додати на власну панель інструментів кнопку виклику цього макроса. Протестувати модуль.

Завдання 10.7. Програмування процедур обробки тексту.

1. Визначити у тексті максимальну довжину послідовності символів, що не є літерами.
2. Знайти максимальну за довжиною монотонну (неспадну або незростаючу) підпослідовність натуральних чисел, заданих у тексті так, що числа відокремлюються пропусками.
3. Визначити всі слова тексту, що складаються з тих самих літер, що і перше слово цього тексту.
4. З клавіатури вводиться речення з довільних символів. Виберіть з неї всі літери від J(j) до S(s) (великі перетворіть на малі) і впорядкуйте їх у алфавітному порядку.
5. У тексті знайти пари слів, що є дзеркальними відображеннями (типу "Оля" і "яло").
6. У тексті знайти всі симетричні слова (типу "око", "АББА" і т.п.).
7. Знайти всі слова, що повторюються у кожному з двох речень тексту.
8. Маємо текст з цифр і літер. Перетворіть текст так, щоб спочатку йшли всі цифри, а потім літери.
9. Відредагуйте деякий текст, видаливши у ньому всі слова з непарними номерами і перевертаючи слова з парними номерами (типу "How do you do?" $\Rightarrow$ "od od?").
10. Для кожного символу тексту зазначити, скільки разів він повторюється у тексті. Повідомлення про символ виводити тільки один раз.
11. Деякий текст має два речення. Знайти найкоротше слово першого рядка, якого нема у другому реченні.
12. Нехай цифрам від 1 до 9 відповідають літери від A(a) до I(i). З клавіатури вводиться речення з довільних символів. Скласти слово, що складається з цифр, що відповідають тільки даним літерам (малим або великим), впорядкованих за незростанням.
13. З клавіатури вводиться речення з довільних символів. Виберіть з неї всі літери від A(a) до I(i). Малі перетворіть на великі і впорядкуйте їх усіх у алфавітному порядку.

14. Відредагуйте деякий текст, видаливши у ньому всі слова, що мають у своєму складі входження тільки двох літер (типу “Акка абаба сесе кнопка.” ⇒ “кнопка.”).
15. Замінити в англомовному тексті закінчення слів “*ing*” на “*ed*”, стиснувши при цьому текст.
16. В англомовному тексті знайти слово, у якому кількість голосних літер (*a*, *e*, *i*, *o, u*) є найбільшою.
17. Характеристика слова – кількість *різних* символів, що у нього входять. Впорядкувати слова у тексті за спаданням їхніх характеристик.
18. Визначити у тексті пари слів, у яких перше слово закінчується тією літерою, якою починається друге слово цієї пари.
19. Маємо текст з цифр і літер. Перетворіть текст так, щоб спочатку йшли всі літери, а потім цифри.
20. Нехай цифрам від 1 до 9 відповідають літери від A(a) до I(i). З клавіатури вводиться речення з довільних символів. Скласти слово, що складається з цифр, що відповідають тільки даним літерам (малим або великим), впорядкованих за неспаданням.
21. З клавіатури вводиться речення з довільних символів. Виберіть з неї всі літери від A(a) до I(i). Великі перетворіть на малі і впорядкуйте їх усіх за незростанням.
22. Деякий текст має два речення. Знайти найдовше слово першого рядка, якого нема у другому реченні.
23. З клавіатури вводиться речення. Отримати всі речення, що отримуються при перестановці слів заданого речення.
24. З клавіатури вводиться слово. Побудувати всі анаграми слова, тобто слова (можливо, беззмістовні), що складаються з усіх літер вихідного слова, розміщених у довільному порядку.
25. Відредагуйте деякий текст, видаливши у ньому всі слова з парними номерами і перевертаючи слова з непарними номерами (типу “How do you do?” ⇒ “woH uoy?”).

Список літератури

1. *Буч Г.*, *Рамбо Дж.*, *Джекобсон А.* Язык UML. Руководство пользователя. – М.: ДМК Пресс; СПб.: Питер, 2004.
2. *Вирт Н.* Алгоритмы и структуры данных. – М.: Мир, 1989.
3. *Каррано Ф.М.*, *Причард Дж.Дж.* Абстракция данных и решения задач на C++. Стены и зеркала. – М.: Издательский дом "Вильямс", 2003.
4. *Костів О. В.*, *Ярошко С. А.* Методи розробки алгоритмів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002.
5. *Ларман К.* Применение UML и шаблонов проектирования. – М.: Издательский дом "Вильямс", 2004.
6. *Леоненков А.В.* Самоучитель UML. – СПб.: БХВ – Санкт-Петербург, 2004.
7. *Матросов А. В.* и др. MS Office XP: разработка приложений. – СПб.: БХВ – Санкт-Петербург, 2003.
8. *Петров В. Н.* Информационные системы. – СПб.: Питер, 2002.
9. *Скотт К.* Унифицированный процесс. Основные концепции. – М.: Издательский дом "Вильямс", 2003.
10. *Трофимов С.А.* Case-технологии: практическая работа в Rational Rose. – М.: ЗАО "Издательство БИНОМ", 2001.

Зміст

Навчальне видання

Дудзяний Ігор Михайлович

Програмування мовою Visual Basic/VBA

Редактор *І. Лоїк*
Технічний редактор *С. Сеник*
Комп'ютерне макетування *І. Дудзяний*

Підп. до друку . .2004. Формат 60×84/16. Папір друк.
Друк на ризогр. Умовн. друк. арк. 14,0. Обл.-вид. арк. 14.5.
Тираж 150 прим. Зам.

Видавничий центр Львівського національного університету
імені Івана Франка. 79000 Львів, вул. Дорошенка, 41