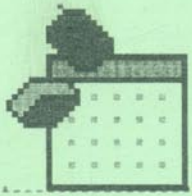
Камінський А. В.

Боцула М. П.



Практичний посібник з основ мови програмування VBA

Міністерство освіти і науки України
Вінницький національний технічний університет

А. В. Камінський, М. П. Боцула

Практичний посібник з основ мови програмування VBA

Затверджено Вченою радою Вінницького національного технічного університету як навчальний посібник для студентів спеціальностей "Електричні системи і мережі", "Електричні станції", "Електротехнічні системи електроспоживання", "Комп'ютерний еколого-економічний моніторинг". Протокол № 11 від 2 липня 2007 р.

Вінниця ВНТУ 2008

УДК 681.3.06
К18

Рецензенти:

Петрук В. Г., д.т.н., проф., директор Інституту екології та екологічної кібернетики, зав. кафедри хімії та екологічної безпеки ВНТУ

Мокін В. Б., д.т.н., доц., зав. кафедри моделювання та моніторингу складних систем ВНТУ

Комісаренко Д. Ю., к.т.н., провідний інженер ТОВ "Арісент Технолоджис Україна"

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України

Камінський А. В., Боцула М. П.

К18 **Практичний посібник з основ мови програмування VBA.** Навчальний посібник. – Вінниця: ВНТУ, 2008. – 80 с.

В навчальному посібнику розглянуті фундаментальні основи мови програмування VBA. Викладення теоретичного матеріалу супроводжується наведенням прикладів, що демонструють ту чи іншу особливість мови. Посібник розроблений відповідно до плану кафедри та програм дисциплін "Інформатика і системологія", "Робота з базами даних", "Алгоритмічні мови та програмне забезпечення", "Обчислювальна техніка і програмування", "Автоматизація обробки інформації засобами Microsoft Excel".

УДК 681.3.06

ЗМІСТ

ВСТУП.
МІСЦЕ МОВИ VBA СЕРЕД ІНШИХ МОВ ПРОГРАМУВАННЯ

На відміну від інших мов програмування мова програмування Visual Basic for Applications (скорочено VBA) не дозволяє створювати автономних додатків у вигляді файлів з розширенням exe, які б самостійно виконувалися в середовищі операційної системи Windows. Сама мова програмування та програми, написані цією мовою, невід'ємні від так званої host-програми (host-application), яка утворює середовище або оболонку, в межах якої функціонують VBA-програми. Як host-програми для мови програмування VBA виступають практично всі програми, які входять до складу відомого пакету програм Microsoft Office, зокрема Microsoft Word та Microsoft Excel. Але незважаючи на те, що сама мова програмування VBA є розробкою фірми Microsoft, вона використовується також в програмних продуктах сторонніх компаній, наприклад, в програмі Autocad, яка є розробкою фірми Autodesk, або в ArcGIS, що є розробкою фірми ESRI.

Сучасна мова програмування VBA є цілком об'єктно-орієнтованою мовою програмування, що надає потужні засоби для реалізації свого призначення – автоматизації операцій, які виконує користувач в середовищі host-програм, та розширення можливостей, що надаються такими середовищами. За допомогою мови VBA створюються складні і потужні промислові інформаційні середовища на базі можливостей Microsoft Office та інших host-програм.

Даний посібник в жодному разі не претендує на повне викладення усіх особливостей та можливостей мови програмування VBA. В ньому розглядаються базові основи мови програмування VBA, і головною метою посібника є роз'яснення принципів функціонування VBA, саме розуміння яких дозволяє ефективно створювати власні програмні додатки до host-програм. Ця робота має за мету навчити читача активно використовувати стандартну документацію VBA, що є найефективнішим шляхом отримання повної, точної та достовірної інформації з будь-якого аспекту програмування.

Основним джерелом інформації, використаним при написанні даного посібника, була стандартна документація з мови програмування VBA. Ця документація постачається у вигляді розділу довідки "Microsoft Visual Ba-

sic Documentation" електронної допомоги користувача, яка викликається із меню "Help" host-програми мови програмування. В зв'язку з цим посилання на цю документацію оформлено у списку літератури в кінці даного посібника під номером [2] дещо нестандартно.

Матеріал посібника рекомендується вивчати паралельно з опрацюванням його за персональним комп'ютером. Всі приклади, які слід спробувати реалізувати самостійно, позначено у посібнику значком  . Синтаксис запису операторів і функцій позначено значком  . Посилання на додаткову інформацію, яку слід переглянути у стандартній документації VBA [2], подано у вигляді назв відповідних розділів документації та позначено значком  . Також у посібнику наведено зауваження, які вказують на особливості і можливості застосування поточних відомостей. Такі зауваження позначено значком  .

Для полегшення роботи читача з документацією з мови програмування VBA, яка постачається англійською мовою навіть у російській локалізованій версії продуктів пакету Microsoft Office, в додатку 1 наведено таблицю відповідності найчастіше вживаних англомовних термінів українським.

Необхідно також зазначити, що мова програмування VBA в даному посібнику розглядається безвідносно до тієї чи іншої host-програми, а тому припускається, що читач даного посібника вже знайомий з інтерфейсом того середовища, в межах якого буде використовуватися мова програмування VBA.

Короткий опис базових можливостей текстового редактора та налагоджувача мови програмування VBA, які використовуються в host-додатках Microsoft Office, наведено у третьому розділі даного посібника.

Посібник може стати у пригоді як читачеві, що вперше знайомиться з цією мовою програмування, так і читачеві, який вже має певний досвід роботи з цією мовою.

Додаткову інформацію з програмування мовою VBA завжди можна знайти в документації з мови програмування [2], багатьох літературних виданнях [3 - 8] та web-ресурсах, посилання на які наведено у додатку 2.

1. ОСНОВНІ ЕЛЕМЕНТИ МОВИ ПРОГРАМУВАННЯ VBA

1.1. Лексичні елементи мови програмування VBA

Першим етапом лексичного аналізу тексту програми, написаної будь-якою мовою програмування, є розбиття його на найменші складові частини, які не підлягають подальшому діленню. Такі складові частини програми називаються ***лексемами***. ***Лексеми*** є базовою одиницею для побудови складніших конструкцій мови, таких як вирази та оператори.

Компілятор аналізує текст програми послідовно символ за символом зліва направо та розбиває його на окремі лексеми максимально можливої довжини. Таким чином запис "DimA" розглядається як одна лексема "DimA", а не дві окремі лексеми "**Dim**" та "A".

Для відокремлення однієї лексеми від іншої необхідно використовувати символи пропуску, табуляції та нового рядка. Ці символи називаються ***символами-роздільниками***. Окрім відокремлення однієї лексеми від іншої вони можуть використовуватися для більш зручного (для людини) оформлення тексту програми.

Коментар – спеціальним чином позначена частина тексту програми, яка ігнорується компілятором. Зазвичай застосовуються для пояснення роботи частин програми для людини. Може використовуватися для тимчасового виключення рядку програми із процесу інтерпретації. Коментар розпочинається символом апострофу (') і закінчується в тому самому текстовому рядку, в якому він розташований (якщо його не продовжено на наступний рядок символом _). Може містити будь-які символи.

Наступна програма містить три коментарі, причому останній коментар розташований у двох текстових рядках:

```
' Цей рядок програми містить тільки коментар
Dim a As Integer ' коментар до змінної a
Dim b As Single ' цей коментар до змінної b _
      розташовується в два рядки
```

Компілятор VBA розпізнає такі види лексем:

- ***ключові слова***;
- ***ідентифікатори***;
- ***літеральні константи***;

- ***пунктуатори*** (знаки пунктуації).

Ідентифікатор (ім'я, назва) – послідовність символів, що використовується для позначення змінних, типів користувача, функцій і т.д. Може складатися тільки із цифр, літер алфавіту і символу підкреслення (_). Перший символ ідентифікатора повинен бути літерою алфавіту. Ідентифікатори нечутливі до регістра літер, тобто рядкові та прописні літери не розрізняються (таким чином, SomeName та soMEnaME вважається одним і тим самим ідентифікатором). Ідентифікатор не може збігатися з ключовими словами. Максимальна довжина ідентифікатора – 255 символів.

Таким чином, ідентифікаторами є лексеми: a, b, abc12, buffer_len. Не є ідентифікаторами такі лексеми: 12abc, _name, black!white.

Ключові слова – лексеми, які мають спеціальне значення в рамках мови VBA і не можуть бути використані як ідентифікатори. Зокрема ключовими словами є назви всіх операторів та типів даних.

Приклади ключових слів: **For**, **Integer**, **ByRef**, **Null**.

Літеральна константа – деяке незмінне значення. Бувають таких видів: ***цілі***, ***з плаваючою крапкою***, ***рядкові*** та ***літеральні константи дати***.

Про літеральні константи дати:
- Date Data Type

Кожна літеральна константа має певний тип даних (типи даних розглядаються в розділі 1.4), який обирається автоматично залежно від виду та значень літеральної константи.

Цілі константи задають цілі числа. Для цілої константи, як правило, обирається один з двох типів даних: **Integer** або **Long**, який має найменший достатній розмір для утримання значення відповідної константи. Для натуральних чисел може бути використано тип **Byte**.

Наприклад, ціла літеральна константа -123456 буде мати тип даних **Long**.

Літеральні константи ***з плаваючою крапкою*** задають дійсні числа. Такі константи мають цілу частину, дробову частину та показник степені, який може бути відсутнім. Дробова частина числа відокремлюється від цілої символом крапки. Показник степені відокремлюється символом e або E.

Для константи ***з плаваючою крапкою*** обирається найменший із двох типів даних: **Single** та **Double**, який має достатній розмір для значення відповідної константи.

Наприклад, літеральна константа з плаваючою крапкою 123.45E-3

задає число $123{,}45 \cdot 10^{-3}$. Ця константа буде мати тип даних **Single**.

Рядкова літеральна константа складається із нуля (напр., "") або більше символів (напр., "рядкова"), взятих у подвійні лапки. Рядкова літеральна константа має тип даних **String**.

Приклад рядкової літеральної константи: "String literal constant". Вона має тип даних **String**.

Пунктуатори – символи, що мають спеціальне значення. Наприклад, плюс (+), зірка (*), двокрапка (:), кома (,). Деякі ***пунктуатори*** можуть бути операціями, наприклад, плюс (+) є операцією додавання, зірка (*) – операцією множення.

Розглянемо таку програму:

```
Sub test()
    Dim a As Single, m(10) As Integer
    a = 10.2+7
End Sub
```

Перший рядок даної програми містить чотири лексеми: ключове слово **Sub**, ідентифікатор test та два пунктуатори: відкриваючу круглу дужку та закриваючу круглу дужку.

Другий рядок даної програми містить одинадцять лексем: по одному ключовому слову **Dim**, **Single**, **Integer**, два ключових слова **As**, ідентифікатори a та m, цілочислову константу 10, а також три пунктуатори: кому (,), відкриваючу круглу дужку та закриваючу круглу дужку.

Третій рядок даної програми утримує три лексеми: ідентифікатор a, числову константу з плаваючою крапкою 10.2, цілочислову контсанту 7, а також два пунктуатори: знак рівності (=) та знак додавання (+).

Четвертий рядок даної програми містить дві лексеми, які є ключовими словами **End** та **Sub**.

Очевидно, що не всяка послідовність лексем складає правильну програму. Допустимі комбінації лексем та їх тлумачення визначаються набором правил, який називається ***синтаксисом*** мови програмування.

1.2. Синтаксична нотація

Синтаксична нотація є формальною системою позначень, яка дозволяє в компактній формі однозначно описати синтаксис (тобто правила запису тексту програм) тієї чи іншої мови програмування. Перша синтак-

сична нотація, що згодом стала відомою під назвою BNF (Backus-Naur Form), була створена і застосована у 1960 році для опису синтаксису мови програмування Algol 60. В наш час стало традицією застосовувати ту чи іншу форму синтаксичної нотації для опису синтаксису мов програмування. Без використання BNF нотації або одного із її різновидів не обходиться жодна документація з мови програмування, в тому числі документація з мови програмування VBA.

Після засвоєння основ тієї чи іншої мови програмування документація з цієї мови стає основним джерелом довідникової інформації з даної мови як для початківців, так і для професійних програмістів. Виходячи з цього, в даному посібнику для опису синтаксису мови програмування використовується синтаксична нотація, прийнята в документації з мови програмування VBA.

Синтаксична нотація описує правила, за якими складаються окремі елементи мови програмування. Такі правила в синтаксичній нотації описуються за допомогою таких позначень:

- *терміналів* – слів та символів, які безпосередньо відповідають лексемам мови програмування, тобто повинні вводитись в текст програми в тому вигляді, в якому вони задані в описі даного правила. У використовуваній в даному посібнику синтаксичній нотації термінали виділяються жирним шрифтом;
- *нетерміналів* – слів, які в свою чергу описуються іншими правилами синтаксичної нотації або неформально українською мовою. В прийнятій в даному посібнику синтаксичній нотації нетермінали виділяються курсивом;
- *квадратних дужок*, які позначають необов'язкову складову елемента мови програмування, який описується даним правилом;
- *фігурних дужок* та *вертикальної риски*. Дані символи використовуються для позначення того, що в даному місці програми повинен використовуватися один із елементів, взятих у фігурні дужки та розділених вертикальною рискою;
- *трьох крапок*, розташованих після квадратних дужок. Три крапки вказують на те, що частина елемента мови програмування, взята у квадратні дужки, може бути повторена декілька разів.

Розглянемо декілька прикладів опису синтаксису мови VBA за допомогою прийнятої синтаксичної нотації.

Наприклад, правило написання оператора **Do** в програмі на VBA,

представлене за допомогою розглянутої синтаксичної нотації, буде мати вигляд:

Do [{**While** | **Until**} *exp*]
    [*statement*]
Loop

Написані жирним шрифтом слова **Do**, **While**, **Until** та **Loop** є терміналами і тому повинні задаватися в програмі у такій самій формі, як вони задані у синтаксичній нотації. Написані курсивом слова *exp* та *statement* є нетерміналами і потребують додаткового роз'яснення. В даному випадку *exp* відповідає будь-якому виразу мови програмування VBA, а *statement* – одному або декільком довільним операторам VBA.

Термінали **While** та **Until** взято у фігурні дужки та розділено вертикальною рискою, що означає, що у відповідному місці програми необхідно записати або слово **While,** або **Until**. Нетермінал *statement* взято у квадратні дужки, тому він може бути опущений у тексті програми. Те саме стосується всієї частини першого рядка, розташованої після термінала **Do**. Таким чином, заданому синтаксису оператора Do будуть відповідати такі записи:

```
Do Until a>10
      MsgBox a
      a=a+1
Loop
Do While b
Loop
Do
Loop
```

Символи [ ] { } | є частиною синтаксичної нотації і не є елементами мови програмування VBA, тому повинні бути опущені при записі програми згідно з заданою синтаксичною нотацією.

Елементи програми, які відповідають елементам синтаксичної нотації, розташованим в окремих текстових рядках, теж повинні розташовуватися в окремих текстових рядках.

1.3. Структура програми

Весь текст програми на VBA розташовується в ***модулях*** різних типів. Набір модулів, розташованих в одному і тому самому документі, прийнято називати ***проектом***.

В різних host-програмах можна використовувати модулі різних типів. При цьому модуль конкретного типу може виступати не тільки як контейнер для тексту програми VBA, а також відігравати додаткову роль, наприклад, модуль форми також містить графічний опис форми.

Але в кожній host-програмі можна використовувати модулі двох таких типів: ***стандартний модуль*** та ***модуль класу***. Модулі цих двох типів призначені тільки для того, щоб містити текст програми VBA. Ще одна їх особливість порівняно із модулями інших типів полягає в тому, що ***змінні***, ***процедури*** та ***функції***, які описані в стандартному модулі або у модулі класу, можуть бути доступні в модулях інших проектів.

Розгляд модулів класу та модулів інших типів виходить за межі даного навчального посібника. Більш детальну інформацію з даного приводу можна отримати в документації на мову програмування VBA [2].

У мові VBA діє таке правило: оператори опису змінних та констант, які не знаходяться всередині функції чи процедури, а також оператори Option можуть розташовуватись тільки на початку модуля перед всіма описами процедур та функцій. Виходячи із цього, модуль умовно ділять на дві частини: секцію описів та секцію процедур.

Таким чином, секція опису є текстом програми на початку модуля, який містить тільки оператори опису змінних та оператори Option. Секція процедур розташовується після секції означень та містить тільки оператори опису процедур та функцій.

```
' Це секція декларацій
Option Explicit
Dim a As Integer
Option Base 1

' Це секція процедур
Sub test()
End Sub
```

1.4. Типи даних

Тип визначає зміст, розмір та діапазон значень змінної.

Змінна – іменована область пам'яті, що містить дані певного типу, вміст якої може бути змінений під час виконання програми.

У мові програмування VBA є такі типи даних:

1) числові типи даних:
   - цілочислові типи: **Byte**, **Integer**, **Long**;
   - типи даних з плаваючою крапкою: **Single**, **Double**, **Currency**;
2) тип даних **Boolean**;
3) тип даних **Date**;
4) тип даних **String**;
5) об'єктні типи даних:
   - тип даних **Object**;
   - конкретні об'єктні типи даних (наприклад, **Application**, **Collection** і т.д.);
6) типи даних користувача (структури, створюються за допомогою оператора **Type**, та переліки, створюються за допомогою оператора **Enum**).

Змінні числових типів даних призначені для зберігання чисел, причому змінні цілочислового типу можуть мати тільки цілі числа, а змінні, які мають тип даних з плаваючою крапкою, можуть мати також дійсні числа.

Змінна типу **Byte** має розмір 1 байт і може зберігати тільки беззнакові (тобто невід'ємні) цілі числа з діапазону від 0 до 255.

Змінна типу **Integer** має розмір 2 байти і може зберігати цілі числа із знаком (тобто додатні, від'ємні та нуль) із діапазону від -32768 до 32767.

Змінна типу **Long** має розмір 4 байти і може зберігати цілі числа із знаком із діапазону від -2147483648 до 2147483647.

Всі змінні, які мають тип даних з плаваючою крапкою (дійсні числа), можуть зберігати як додатні, так і від'ємні числа.

Змінна типу **Single** має розмір 4 байти і може зберігати дійсні числа із діапазону від $-3{,}4 \cdot 10^{-38}$ до $3{,}4 \cdot 10^{38}$.

Змінна типу **Double** має розмір 8 байт і може зберігати дійсні числа із діапазону від $-1{,}7 \cdot 10^{-308}$ до $1{,}7 \cdot 10^{308}$.

Про перечислення:
- **Enum Statement**

Змінна типу **Currency** має розмір 8 байт і може зберігати дійсні чис-

ла із діапазону від -922337203685477,5808 до 922337203685477,5807. На відміну від типів даних з плаваючою крапкою, тип **Currency** зберігає тільки 4 цифри після десяткової коми.

Змінна типу даних **Boolean** має розмір 2 байти і може зберігати тільки одне із двох логічних значень – True ("істина") або False ("хибність").

Змінна типу даних **String** може зберігати послідовність довільних символів, яку називають текстовим рядком. Розмір змінної типу **String** залежить від довжини текстового рядка (тобто кількості символів, які зберігає змінна). Змінна типу **String** може також зберігати пустий текстовий рядок (тобто текстовий рядок, який не утримує жодного символу).

Змінна типу **Date** має розмір 8 байт і може зберігати дату із діапазону від 1.1.100 до 31.12.9999 та час із діапазону від 0:00:00 до 23:59:59.

Змінні об'єктних типів даних можуть зберігати адреси об'єктів. Змінна типу **Object** може містити адресу будь-якого об'єкта (наприклад, **Collection**, **Application** і т.д.), а змінна конкретного об'єктного типу може містити адресу тільки такого самого об'єкта (наприклад, змінна типу **Application** може зберігати тільки адресу об'єкта **Application**).

Змінні об'єктних типів можуть також утримувати спеціальне значення **Nothing**, яке вказує на те, що в даний момент ця змінна не містить адреси ніякого об'єкта. Декілька різних об'єктних змінних можуть зберігати адресу одного і того самого об'єкта, наприклад:

```
Dim a As Object, b As Collection, c As Collection
Set a = New Collection
Set b = a
Set c = a
```

В наведеному прикладі створюється всього один об'єкт **Collection** та три об'єктні змінні, в які заноситься адреса створеного об'єкта **Collection**. Тобто в оперативній пам'яті комп'ютера існує лише один об'єкт, а не три.

Variant – це спеціальний тип даних, змінні якого можуть зберігати значення будь-якого іншого типу даних. Не описані змінні мають тип даних **Variant**. Змінні типу **Variant** ініціалізуються спеціальним значенням **Empty**, яке вказує на те, що змінній не було присвоєно ніякого іншого значення. Змінним типу **Variant** можна присвоювати спеціальне значення **Null**, яке вказує на те, що змінна не містить ніякого значення іншого типу. Змінна типу **Variant** займає більше пам'яті, ніж змінна із спеціалізованим

типом даних, який підходить для зберігання того самого значення, через необхідність зберігати ознаку типу значення, яке міститься в даний час у змінній типу **Variant**.

Всі розглянуті вище типи даних називаються також базовими або вбудованими. Використовуючи ці типи даних користувач має змогу створювати власні складніші типи даних за допомогою спеціального оператора **Type**. Тип даних користувача, створений за допомогою оператора **Type**, також називається структурою. Фактично структура є набором окремих змінних інших типів даних (як базових, так і масивів та інших структур), які називаються полями або членами структури та розташовуються в пам'яті послідовно в тому порядку, в якому вони були перераховані в операторі **Type**.

Оператор **Type** має такий синтаксис:

[Private | Public] Type *ident-typename*
ident [(*subscripts*)] **As** *type*
[*ident* [(*subscripts*)] **As** *type*]
...
End Type

Private – при використанні даного ключового слова тип даних користувача буде мати область дії – модуль (див. п. 1.8), в якому знаходиться даний оператор **Type**.

Public – при використанні даного ключового слова тип даних користувача буде мати область дії – проект.

ident-typename – ідентифікатор, який задає назву для типу даних користувача.

ident, *subscripts*, *type* – мають те саме значення, що і в операторі **Dim** (див. п. 1,5).

Оператор **Type** може розташовуватися тільки в секції визначень модуля. Якщо в операторі **Type** не було використано жодне з ключових слів **Private** та **Public**, тип даних користувача буде мати область дії – проект.

Якщо поле структури є масивом фіксованого розміру (див. п. 1.7), то розмірності даного масиву повинні задаватися константними виразами.

Нижче наведено приклад опису структури та використання її полів:

```
Type Student
        first_name As String
        last_name As String
        age As Integer
        course As Byte
End Type

Sub Example
        Dim s1 As Student
        s1.first_name = "Вася"
        s1.last_name = "Пупкін"
        s1.age = 20
        s1.course = 1
        MsgBox "В наступному році " & s1.first_name & _
                " буде на " & s1.course+1 & _
                " курсі"
End Sub
```

1.5. Опис та ініціалізація змінних

Під час опису змінної задається її тип, виділяється та ініціалізується пам'ять для змінної. В одній і тій самій області дії (див. п. 1.4) не може бути описано дві змінні з однаковим ім'ям.

Під ***ініціалізацією*** змінної розуміють занесення у змінну початкового значення одразу після її створення. Кожна змінна будь-якого типу в мові VBA автоматично ініціалізується строго визначеними значеннями.

Змінні описуються за допомогою операторів **Dim**, **Public**, **Private**, **Static**.

Оператор опису змінних **Dim** має такий синтаксис:

Dim *ident* [**As** [**New**] *type*] [, *ident* [**As** [**New**] *type*]]...

ident – назва змінної.

type – тип змінної; якщо його не вказано – використовується тип **Variant**.

Ключове слово **New** дозволяє створити новий об'єкт типу *type*, адреса якого заноситься у змінну *ident*. При використанні ключового слова **New** тип *type* повинен бути об'єктним типом.

В момент створення числові змінні ініціалізуються значенням 0, рядкові змінні змінної довжини – пустим рядком (""), а рядкові змінні фіксованої довжини заповнюються нулями.

Змінні типу **Variant** ініціалізуються значенням **Empty**. Об'єктні змінні ініціалізуються значенням **Nothing**, якщо ключове слово **New** не було використано.

Кожне поле змінної, що має тип даних користувача, створений за допомогою оператора **Type**, ініціалізується так само, як і окремі змінні відповідного типу.

Змінні, описані за допомогою оператора **Dim**, будуть мати область дії процедура чи функція, якщо опис розташовано всередині процедури чи функції, та область дії модуль, якщо опис розташовано в секції визначень модуля.

Оператори опису змінних **Public**, **Private** та **Static** мають такий самий синтаксис, як і оператор **Dim**, тільки замість ключового слова **Dim** використовуються відповідно ключові слова **Public**, **Private** та **Static**.

Все сказане вище для оператора **Dim** справедливо також для операторів **Public**, **Private** та **Static** за винятком наступного.

Оператори **Public** та **Private** відрізняються від оператора **Dim** тільки областю дії, яку будуть мати змінні, що описуються за допомогою даних операторів. Змінні, описані за допомогою оператора **Public**, будуть мати область дії проект. Змінні, описані за допомогою оператора **Private**, будуть мати область дії модуль.

Оператори **Public** та **Private** можуть застосовуватися тільки в секції опису модуля (тобто не можуть використовуватися для опису змінних всередині процедур та функцій).

Оператор **Static** відрізняється від оператора **Dim** тільки часом життя, який будуть мати змінні, що описуються за допомогою даного оператора. Змінні, описані за допомогою оператора **Static**, будуть мати статичний час життя.

Оператор **Static** може застосовуватися тільки всередині процедур та функцій (тобто не може використовуватися для опису змінних в секції опису змінних модуля).

```
Dim a As Integer, b, str As String
Private x As Single
Public y As Application
Sub Test()
    Dim c As New Collection
    Static z
End Sub
```

В наведеному вище фрагменті програми змінні b та z мають тип даних **Variant**. Змінна c має автоматичний час життя, всі інші змінні мають статичний час життя. Змінні c та z мають область дії процедура Test(), змінна y має область дії проект, всі інші змінні мають область дії модуль. Змінні a та x ініціалізуються нулем, змінна str – пустим рядком, змінні b та z – значенням **Empty**, змінна y – значенням **Nothing**, а змінна c – адресою нового об'єкта типу **Collection**.

У програмі на VBA описувати змінні перед їх використанням за допомогою розглянутих вище операторів необов'язково. Але при цьому всі неописані змінні будуть мати тип **Variant**, що в багатьох випадках небажано. У VBA є спеціальний оператор **Option Explicit**, який забороняє використання змінних без їх попереднього опису за допомогою розглянутих операторів опису змінних.

Оператор **Option Explicit** має такий синтаксис:

Option Explicit

Оператор **Option Explicit** може розташовуватися тільки в секції описів модуля. При наявності у модулі даного оператора використання змінної, яка не є явно описаною, призведе до виникнення помилки.

Використання оператора **Option Explicit** дозволяє прискорити роботу програми за рахунок "економії" часу на розпізнання типу змінної під час виконання програмного проекту.

1.6. Опис констант

Окрім ***літеральних констант*** (див. п. 1.1) в мові програмування VBA існують також ***іменовані константи***, які надалі будуть називатися просто ***константи***.

Іменовану константу можна розглядати як змінну, значення якої не може бути змінено протягом виконання програми. Опис константи задає її тип та значення. Іменовані константи описуються за допомогою оператора **Const**.

Синтаксис оператора **Const**:

[{**Public** | **Private**}] **Const** *ident* [**As** *type*] = *exp* _
[, *ident* [**As** *type*] = *exp*]...

ident – ***ідентифікатор***, який задає назву константи.
type – ***тип*** константи (не може бути об'єктним та типом, що визначається користувачем). Якщо тип не вказано, використовується тип даних, який має результат виразу *exp*.
exp – ***константний вираз***, тобто вираз, який може містити тільки ***іменовані*** та ***літеральні*** константи, а також ***арифметичні*** та ***логічні операції*** (див. п. 1.11, 1.12).

Ключові слова **Public** та **Private** можуть застосовуватися при описі констант тільки в секції опису модуля (див. п. 1.3). Ключове слово **Public** вказує на те, що всі константи, описані за допомогою оператора **Const**, будуть мати область дії проект, а ключове слово **Private** вказує на те, що вони будуть мати область дії модуль.

Якщо жодне з ключових слів **Public** та **Private** не вказано, константи, описані в секції опису модуля, будуть мати область дії модуль.

```
Public Const NutsCount = 10, ChildCount As Byte = 3
Const NutsPerChild = NutsCount / ChildCount
```

Константа ChildCount буде мати тип даних **Byte**, оскільки цей тип даних явно вказано під час опису цієї константи. Оскільки для константи NutsCount тип даних явно не вказано, вона буде мати такий саме тип даних, як і результат того виразу, який присвоюється цій константі. В даному випадку цей вираз складається тільки із літеральної константи 10, яка має тип даних **Integer** (див. п. 1.4). Отже, константа NutsCount теж буде мати тип даних **Integer**.

Константі NutsPerChild присвоюється значення виразу NutsCount / ChildCount. Результатом операції ділення над операндами типу **Integer** та **Byte** є значення типу **Double** (див. п. 1.11), отже константа NutsPerChild теж буде мати тип **Double**.

Оскільки константи NutsCount та ChildCount описані з використанням ключового слова **Public**, вони будуть мати область дії проект. Константа NutsPerChild буде мати область дії модуль, оскільки під час її опису не використовувалося жодне з ключових слів **Public** та **Private**.

У тих випадках, коли те чи інше конкретне значення (тобто літеральну константу) необхідно використовувати більш ніж в одному місці програми, доцільно створити іменовану константу та присвоїти їй те конкретне значення, яке повинно використовуватися. Наприклад, у програмі, яка запрошує користувача задати пароль, що буде використовуватися для доступу до особистої інформації, мінімальна дозволена довжина паролю може бути задана за допомогою такої іменованої константи:

Public Const MinPasswordLength **As Byte** = 5

По-перше, це робить програму зрозумілішою для програміста, оскільки в усіх місцях програми, в яких використовується ця довжина, замість конкретного числового значення 5 програміст побачить більш інформативну назву MinPasswordLength. По-друге, якщо у випадку виникнення необхідності підсилення безпеки буде прийнято рішення про збільшення мінімальної допустимої довжини паролів, достатньо буде замінити значення 5 на нове тільки в одному місці програми (у рядку опису константи MinPasswordLength). А якби в усіх місцях програми, в яких виконується перевірка довжини пароля, заданого користувачем, замість іменованої константи MinPasswordLength використовувалася літеральна константа 5, необхідно було б виконати пошук всіх таких місць та заміну значення 5 в них.

1.7. Масиви. Оператори та функції для роботи з масивами

Масив – набір послідовно проіндексованих елементів однакового типу даних. Кожен елемент масиву має унікальний номер (***індекс***), який ідентифікує даний елемент.

Масиви бувають ***одновимірні*** та ***багатовимірні***. Кожен вимір масиву має ***верхню*** та ***нижню границю***.

Розмір (кількість елементів) одновимірного масиву дорівнює різниці між значеннями верхньої та нижньої границь масиву плюс 1. Розмір багатовимірного масиву дорівнює добутку таких різниць для кожного виміру масиву.

Масиви бувають ***динамічні*** та ***фіксовані***. ***Фіксований масив*** – це масив, розмір якого вказується під час опису і більше не може бути змінений. ***Динамічний масив*** – це масив, розмір якого не вказується під час опису і може бути змінений у програмі у будь-який час.

Масиви описуються за допомогою операторів **Dim**, **Public**, **Private**

та **Static**, які мають однаковий синтаксис. Відмінності між даними операторами такі самі, як і для випадку опису змінних (див. п. 1.5).

Оператор опису змінних **Dim** для випадку опису масивів має такий синтаксис:

Dim *ident*([*subscripts*]) [**As** [**New**] *type*] _
[, *ident*([*subscripts*]) [**As** [**New**] *type*]] ...

ident – назва масиву.
subscripts – має такий синтаксис:

[*lower* **To**] *upper* [, [*lower* **To**] *upper*] ...

lower, *upper* – вирази, результати яких задають нижню та верхню границі масиву, відповідно. Якщо нижня границя масиву не задана, то її значення у даному модулі визначається оператором **Option Base**. Якщо оператор **Option Base** відсутній, то значення нижньої границі масиву дорівнює 0.
type – тип, який буде мати кожен елемент масиву, якщо не вказано, використовується тип **Variant**.

Якщо ключове слово **New** вказано, то створюється стільки нових об'єктів *type*, скільки елементів містить масив. Адреси цих об'єктів заносяться в елементи масиву. При використанні ключового слова **New** тип *type* повинен бути об'єктним типом.

Масив може мати не більше 60 вимірів.

Якщо жодної границі масиву не вказано, то масив буде ***динамічним***. Після створення розмір такого масиву буде дорівнювати 0. Розмір масиву можна змінювати далі у програмі будь-яку кількість разів за допомогою оператора **ReDim**.

Якщо хоча б одна границя масиву вказана, то масив буде ***фіксованим***. Змінити розмір такого масиву за допомогою оператора **ReDim** неможливо.

Оператор **Option Base** використовується для задання нижньої межі масивів за замовчуванням і має такий синтаксис:

Option Base {**0** | **1**}

Оператор **Option Base** може бути заданий тільки на рівні модуля і тільки один раз перед всіма процедурами та функціями (тобто в секції опи-

сів модуля), а також перед описами масивів фіксованих розмірів. Якщо даний оператор не заданий, нижня межа масивів за замовчуванням буде дорівнювати 0. Оператор **Option Base** впливає на опис масивів тільки того модуля, в якому він сам знаходиться.

```
Option Base 1
Dim a(3 To 5, 2 To 7) As Integer, _
    b() As Long, c(2+3)
```

Наведений вище оператор **Dim** описує три масиви. Масив a є двовимірним ***фіксованим*** масивом, кожен елемент якого має тип **Integer**. Загальна кількість елементів масиву a складає (5–3+1)*(7–2+1)=18.

Масив b є ***динамічним*** масивом, кожен елемент якого буде мати тип **Long**. Одразу після створення кількість елементів масиву b дорівнює нулю. Кількість елементів масиву b можна змінити надалі за допомогою оператора **ReDim**.

Масив c є одновимірним ***фіксованим*** масивом, кожен елемент якого має тип **Variant**. Нижня границя першого (і єдиного) виміру масиву c складає 1 згідно з оператором **Option Base**. Верхня границя першого виміру масиву c задається виразом 2+3 і складає 5. Загальна кількість елементів масиву c складає 5–1+1=5.

Багатовимірні масиви можна розглядати як масиви масивів. Наприклад, наведений вище двовимірний масив a – це одновимірний масив із 3-х елементів, кожний елемент якого є одновимірним масивом із 6-ти елементів типу **Integer**.

Оператор **ReDim** використовується для зміни розміру динамічних масивів та має такий синтаксис:

ReDim [**Preserve**] *ident* **(** [*lower* **To**] *upper* [**,** [*lower* **To**] *upper*] **)** _
[**,** [**Preserve**] *ident* **(** [*lower* **To**] *upper* [**,** [*lower* **To**] *upper*] **)**]...

ident – назва масиву, розмір якого буде змінюватися. Якщо масиву із такою назвою не існує, буде створено новий динамічний масив.

lower, *upper* – вирази, результати яких задають нові значення для нижньої та верхньої границь масиву, відповідно. Якщо нижня границя масиву не задана, то її значення у даному модулі визначається оператором **Option Base**. Якщо оператор **Option Base** відсутній, то значення нижньої границі масиву дорівнює 0.

При використанні ключового слова **Preserve** не можна змінювати кількість вимірів масиву, можна змінювати розмір тільки останнього виміру масиву, а також можна змінювати значення тільки верхньої границі масиву.

Якщо ключове слово **Preserve** вказано, існуючі значення елементів масиву буде збережено при зміні його розміру, інакше значення всіх елементів масиву втрачаються. При зменшенні розміру масиву незалежно від наявності ключового слова **Preserve** останні елементи масиву, які перестають вміщатися у масив внаслідок зменшення його розміру, знищуються.

Розмір динамічного масиву можна змінювати будь-яку кількість разів. Однак оператор **ReDim** неможна використовувати для масивів, які передані у процедуру чи функцію за посиланням (див. п. 2.6).

Всі елементи масиву або тільки нові елементи масиву (при використанні ключового слова **Preserve** та збільшенні розміру масиву) ініціалізуються значеннями так само, як і при використанні оператора **Dim**.

```
Sub test()
Dim m() As Integer
      ReDim m(2 To 5, 3 To 10)
      m(2,3) = 123
      ReDim Preserve m(2 To 5, 3 To 20)
      MsgBox m(2,3) ' на екрані відобразиться _
                      значення 123
      ReDim m(50, 50)
      MsgBox m(2,3) ' на екрані відобразиться _
                      значення 0
      ReDim m(10)
End Sub
```

На початку наведеної вище процедури створюється динамічний масив m. Одразу після створення масив m не містить жодного елемента. Далі за допомогою оператора **ReDim** масив m перетворюється у двовимірний масив із 32 елементів. Після цього в елемент масиву з індексами (2,3) заноситься значення 123.

Далі за допомогою оператора **ReDim** змінюється верхня границя другого виміру масиву m. Оскільки при цьому було використано ключове слово **Preserve**, всі елементи масиву, які існували до зміни верхньої границі, зберігають свої значення. Зокрема, елемент масиву з індексами (2,3) зберігає своє значення 123, яке виводиться на екран за допомогою функції

MsgBox (див. п. 2.9). Після цього верхня границя другого виміру масиву m ще раз збільшується за допомогою оператора **ReDim**, але на цей раз без використання ключового слова **Preserve**. В результаті всі елементи масиву втрачають свої значення та ініціалізуються значенням 0, яким ініціалізуються всі змінні типу **Integer**. Тому другий оператор **MsgBox**, який виводить на екран значення елемента масиву з індексами (2,3), виведе на екран число 0. Останній оператор **ReDim** змінює кількість вимірів масиву m, який перетворюється у одновимірний.

В деяких випадках границі того чи іншого виміру масиву можуть бути невідомі (наприклад, якщо масив передано як аргумент процедурі або функції). Для визначення нижньої границі заданого виміру будь-якого масиву можна скористатися стандартною функцією **LBound**, а для визначення верхньої границі – функцією **UBound**.

Функція **LBound** має такий синтаксис:

```
Function LBound(arrayname() As type, dimension As Long = 1) _
        As Long
```

type – тип даних.

Функція **UBound** має такий самий синтаксис, як і функція **LBound**.

Наступна процедура відображає на екрані нижню та верхню границі першого виміру будь-якого масиву, переданого як аргумент:

```
Sub DisplayBounds(a() As Integer)
    MsgBox LBound(a)
    MsgBox UBound(a)
End Sub
```

1.8. Області дії змінних та констант

Імена змінних та констант в програмі VBA можуть використовуватися (тобто бути доступними або видимими) тільки в певних частинах програми, які називаються ***областями дії*** або ***областями видимості*** цих імен.

Є такі види області дії:

- область дії процедура або функція. Змінна або константа з такою областю дії доступна після точки свого опису тільки всередині тієї процедури або функції, в якій ця змінна описана;

- область дії модуль. Змінна або константа з такою областю дії доступна в усіх процедурах та функціях тільки того модуля, всередині якого ця змінна або константа описана;
- область дії проект. Змінна або константа з такою областю дії доступна в усіх процедурах та функціях всіх модулів проекту.

Опис змінної або константи та його місце розташування у тексті програми визначають область дії змінної або константи. Так, змінні та константи, що описані всередині процедури чи функції, мають область дії процедура чи функція.

Змінні, описані в секції визначень модуля за допомогою операторів **Dim** та **Private** мають область дії модуль. Константи, описані в секції визначень модуля як **Const** та **Private Const** мають область дії модуль. Змінні, описані в секції визначень модуля за допомогою оператора **Public**, мають область дії проект. Константи, описані в секції визначень модуля як **Public Const** мають область дії проект.

```
Dim m1
Private m2
Public p
Const cm1=1
Private Const cm2=2
Public Const cp=cm1*cm2

Sub s1()
Dim a
    a = 10+cm1
    Const c1 = cm2+cp
    p = m1+m2*c1
End Sub

Sub s2()
    MsgBox m1+cm1
    m2 = p+3*cm2
    Dim a
    a = cp
    Dim x
    x = 20
End Sub
```

У наведеній вище програмі змінні m1, m2 та константи cm1, cm2 мають область дії модуль. Це означає, що ці змінні та константи можна використовувати у всіх процедурах та функціях, розташованих в цьому са-

мому модулі, але не можна використовувати в усіх інших модулях.

Змінна p та константа cp мають область дії проект. Це означає, що їх можна використовувати у всіх процедурах та функціях, розташованих не тільки в цьому самому модулі, але і в будь-якому іншому модулі даного проекту.

Змінна a, розташована всередині процедури s1(), має область дії процедура s1(). Змінна a, розташована всередині процедури s2(), має область дії процедура s2(). Це дві різні, ніяким чином не пов'язані між собою змінні. Кожна з них може використовуватися тільки в межах відповідної процедури після місця свого опису.

Константа c1 має область дії процедура s1(). Цю константу можна використовувати тільки в межах процедури s1() після місця опису цієї константи.

Змінна x має область дії процедура s2(). Цю змінну можна використовувати тільки в межах процедури s2() після місця опису цієї змінної.

1.9. Час життя змінних

Час життя змінної – період часу виконання програми, протягом якого змінна існує. Може бути ***статичним*** (***глобальним***) та ***автоматичним*** (***локальним***).

Статичні змінні існують протягом всього часу виконання програми (будь-якого коду в будь-якому модулі).

Автоматичні змінні створюються (тобто під них відводиться пам'ять) кожен раз, коли починає виконуватися процедура або функція, в якій вони були описані, та знищуються (пам'ять вивільняється) кожен раз, коли ця процедура чи функція завершується.

Опис змінної та його місце розташування у тексті програми визначають час життя даної змінної.

Змінні, описані в секції визначень модуля, мають ***статичний*** час життя.

Змінні, описані всередині процедури або функції за допомогою оператора опису змінних **Static**, мають ***статичний*** час життя.

Всі змінні, описані за допомогою будь-якого оператора опису всередині функції або процедури, яка в свою чергу описана з ключовим словом **Static**, мають ***статичний*** час життя.

В усіх інших випадках змінні, описані всередині процедури чи функ-

ції, мають ***автоматичний*** час життя.

```
Dim a As Integer
Public b As Integer

Sub test()
    Dim s1 As Integer
    Static s2 As Integer
    s1 = s1 + 1
    s2 = s2 + 1
    a = a + 1
    MsgBox s1
    MsgBox s2
    MsgBox a
End Sub

Static Sub test2()
    Dim x As Integer
    Static y As Integer
    a = 10
    Call test
    Call test
End Sub
```

У наведеній вище програмі змінні a, b, s2, x та y мають статичний час життя. Це означає, що дані змінні створюються в момент запуску програми та існують доти, доки програма не завершиться. Так, якщо запустити наведену вище програму з процедури test2(), ці змінні будуть існувати протягом виконання цієї процедури, а також протягом виконання процедури test1(), яка викликається із процедури test2().

На відміну від статичних змінних, автоматична змінна s1 буде створюватися в момент виклику процедури test1() та знищуватися в момент завершення цієї процедури.

З врахуванням сказаного вище, розглянемо які значення буде відображено на екрані даною програмою, якщо її запустити з процедури test2().

В момент запуску програми створюються всі ***статичні*** змінні та ініціалізуються значенням 0, яким ініціалізуються всі змінні типу **Integer**. Після цього в змінну a заноситься значення 10. Далі викликається процедура test(), що приводить до створення та ініціалізації нулем автоматичної змінної s1. Після цього значення змінних s1, s2 та a збільшуються на оди-

ницю. Оскільки в цей момент часу всі змінні, окрім a, зберігають значення 0, то після збільшення на одиницю змінні s1 та s2 набувають значення 1, а змінна a, яка містила значення 10, набуває значення 11. Тому в результаті виконання трьох розташованих далі у програмі функцій **MsgBox** на екрані відобразяться значення 1, 1 та 11.

Після цього процедура test() завершується, що приводить до знищення автоматичної змінної s1. Всі статичні змінні продовжують існувати та містити свої поточні значення. Далі із процедури test2() ще раз викликається процедура test. Як і попередній раз, це приводить до створення та ініціалізації нулем змінної s1. Після цього значення цієї змінної збільшується на одиницю і стає рівним 1. В статичних змінних s2 та a на цей час збереглися значення 1 та 11, відповідно. Тому подальше їх збільшення на одиницю приводить до того, що їх значення стають рівними 2 та 12, відповідно, і в результаті виконання розташованих далі викликів функцій **MsgBox** на екрані відобразяться значення 1, 2 та 12.

1.10. Автоматичне перетворення типів даних

В деяких місцях програми відбувається перетворення значення одного типу даних у значення іншого типу даних без явного вказання на те з боку програміста. Такі перетворення називаються ***автоматичними перетвореннями типів даних***.

Зокрема, результат виразу одного типу автоматично перетворюється до іншого типу в таких випадках:

- якщо вираз виступає ***операндом*** операції (див. п. 1.11); тип, до якого перетворюється результат виразу, визначається вимогами операції;
- якщо вираз використовується як умова в операторі вибору If (див. п. 2.3) або операторах циклу (див. п. 2.4); результат виразу перетворюється до типу **Boolean**;
- при використанні виразу в операторі присвоєння **Let** (див. п. 2.2); тип, до якого перетворюється результат виразу, визначається типом змінної, в яку заноситься значення виразу.

Про функції перетворення типів даних:
- Type Conversion Functions

Можливі автоматичні перетворення типів даних, які наведені нижче.

1. Із числового типу даних у тип даних **Boolean**. Числове значення 0 стає значенням **False**, а всі інші числові значення – значенням **True**.
2. Із типу **Boolean** у числовий тип. Значення **False** перетворюєть-

ся у числове значення 0, а значення **True** перетворюється у числове значення -1 для всіх числових типів, окрім **Byte**, для якого значення **True** перетворюється у числове значення 255.

3. Із числового типу даних у тип даних **Date** (дата/час). При такому перетворенні ціла частина числа задає дату (-1 відповідає 29.12.1899, 0 – 30.12.1899, 1 – 31.12.1899, 2 – 1.1.1900 і т.п.), дробова частина задає час (як долю від доби, наприклад, 0.5 відповідає 12:00:00).

4. Із одного числового типу даних у інший числовий тип даних. При перетворенні значення, яке має тип даних з плаваючою крапкою, у значення цілочислового типу даних, значення, що перетворюється, округлюється. Перетворення відбувається тільки в тому випадку, коли значення, що перетворюється (для випадку перетворення із типу з плаваючою крапкою у цілочисловий тип даних береться значення після округлення), не виходить із діапазону значень, допустимих для результуючого типу даних. В протилежному випадку виникає помилка.

5. Із значення будь-якого типу даних, окрім об'єктного та типу даних користувача, у значення типу **Variant** та **String**. При перетворенні у значення типу **String** значення вихідного типу даних перетворюється у відповідну послідовність символів, наприклад, число -12.34 перетворюється у текстовий рядок "-12.34", значення **True** – у рядок "True" і т.д.

6. Із значення типу **Variant** до типу даних, для якого вихідне значення не виходить із діапазону значень, допустимих для результуючого типу даних.

7. Значення **Empty** може бути перетворене у числове значення 0.

8. Значення будь-якого об'єктного типу даних, відмінного від **Object**, може бути перетворено у значення типу даних **Object**.

9. Значення типу **String** може бути конвертоване у число типу **Double**, якщо значення типу **String** містить послідовність символів, яка задає число згідно з поточними регіональними налаштуваннями операційної системи.

З іншими автоматичними перетвореннями типів даних можна докладніше ознайомитися в документації з мови програмування VBA [2].

```
Dim a As Integer
Dim b As Boolean
a = 5
b = a + 3
```

У наведеній вище програмі результат виразу a+3 присвоюється змінній b. Оскільки змінна b має тип **Boolean**, а результатом виразу a+3 є значення 8 типу **Integer**, то відбувається автоматичне перетворення значення типу **Integer** у значення типу **Boolean**. Оскільки значення 8 відмінне від нуля, то змінна b набуває значення **True**.

Для уникнення помилок при перетворенні рядкових значень, введених користувачем, до інших типів даних доцільно застосовувати функції перевірки можливості такого перетворення. Можливість перетворення текстового рядка у значення типу **Date** перевіряє функція **IsDate**, розглянута детальніше в п. 2.12. Можливість перетворення текстового рядка у числове значення перевіряє функція **IsNumeric**, яка має такий синтаксис:

Function IsNumeric(expression **As Variant**) **As Boolean**

Функція **IsNumeric** повертає значення **True**, якщо як аргумент expression передано значення числового типу даних або текстовий рядок, вміст якого може бути перетворено до числового типу даних. В іншому випадку функція повертає значення **False**.

В стандартних українських регіональних налаштуваннях операційної системи Windows у записі дійсних чисел як десятковий відокремлювач використовується символ коми, а не крапки. Тому вираз **IsNumeric**("1,2") буде повертати значення **True**, а вираз **IsNumeric**("1.2") буде повертати значення **False**.

Функції перевірки типів даних слід використовувати для організації стійкості програми до помилок, які можуть виникнути у випадку неправильного введення даних користувачем. У наведеному нижче прикладі виконується перевірка того, чи є значення, введене користувачем, числом:

```
Dim age As Single, str As String
str = InputBox("Введіть ваш вік:")
If IsNumeric(str) Then
        age = str
Else
        MsgBox "Вік введено неправильно!"
End If
```

1.11. Вирази та операції

Вираз – послідовність ***операцій*** та ***операндів***, що задає обчислення, яке необхідно виконати. Результатом цього обчислення завжди є деяке значення певного типу. ***Операнд*** – значення, над яким виконується ***операція***.

Вираз складається із одного або більше операндів та нуля або більше операцій. Як операнди операцій можуть виступати ***константи***, ***змінні*** та ***виклики функцій***.

Існують операції з одним (унарні) та двома (бінарні) операндами. Кожна операція завжди має результат.

В мові програмування VBA є такі ***операції***:

1) Арифметичні:
   - додавання (**+**);
   - віднімання (бінарний **–**);
   - множення (*****);
   - ділення (**/**);
   - ціла частина від ділення (****);
   - остача від ділення (**Mod**) ;
   - піднесення до степеня (**^**);
   - зміна знаку числа (унарний **–**).
2) Рядкові:
   - конкатенація (**&**, а також **+** у тих випадках, коли обидва операнди мають тип даних **String**);
   - операція порівняння рядка із шаблоном (**Like**).
3) Об'єктні:
   - операція порівняння посилань на об'єкти (**Is**).
4) Логічні (бітові) операції:
   - "і" (**And**) ;
   - "або" (**Or**) ;
   - виключальне "або" (**Xor**) ;
   - заперечення (**Not**) ;
   - еквіваленція (**Eqv**) ;
   - імплікація (**Imp**).
5) Операції порівняння:
   - дорівнює (**=**);

- не дорівнює (<>);
- менше (<);
- менше або дорівнює (<=);
- більше (>);
- більше або дорівнює (>=).

Пріоритет операцій визначає послідовність, в якій операції будуть виконуватися в заданому виразі. Спочатку виконуються операції з найвищим пріоритетом, якщо пріоритет двох операцій однаковий, операції виконуються зліва направо в тій послідовності, в якій вони зустрічаються у виразі.

Пріоритет операцій мови VBA наведено у табл. 1.1. Операції, які знаходяться в одному рядку таблиці, мають однаковий пріоритет. Операції, які знаходяться в кожному наступному рядку таблиці, мають нижчий пріоритет відносно операцій в попередньому рядку таблиці.

Круглі дужки можуть застосовуватися для змінення послідовності обчислення операцій у виразі (так само, як це відбувається в математиці в алгебраїчних виразах).

Якщо значення хоча б одного з операндів будь-якої операції є **Null**, результатом операції буде значення **Null**.

Всі арифметичні операції передбачають як операнди значення чисельного типу даних. Якщо операнди бінарної арифметичної операції мають різні типи даних, відбувається перетворення значення операнда, який має менш точний тип даних, до типу даних операнда з більш точним типом даних. Результат операції буде мати такий самий або ще більш точний тип даних, якщо результуюче значення виходить за межі типу даних операнда з точнішим типом даних.

Обидва операнди операцій **Mod** та \ перетворюються до значень цілочислового типу даних. Результат операцій **Mod** та \ теж має цілочисельний тип даних.

Операції порівняння передбачають операнди числового або рядкового типу даних. Результатом операцій порівняння є значення **True**, якщо виконується відповідна рівність чи нерівність, та **False** в протилежному випадку.

Операція **Is** призначена для порівняння значень об'єктного типу даних. Якщо значення обох операндів посилаються на той самий об'єкт, то результатом операції буде значення **True**, інакше – **False**.

Операція конкатенації очікує операндів та має результат рядкового типу даних.

Таблиця 1.1

Пріоритети операцій

Операція	Назва
^	піднесення до степеня
–	унарний мінус
* /	множення ділення
\	ціла частина від ділення
Mod	остача від ділення
+ –	додавання віднімання
&	конкатенація
= <> < > <= >= Like Is	дорівнює не дорівнює менше більше менше дорівнює більше дорівнює порівняння рядка із шаблоном порівняння посилань на об'єкти
Not	заперечення
And	"і"
Or	"або"
Xor	виключальне "або"
Eqv	еквіваленція
Imp	імплікація

Операція **Like** призначена для порівняння текстового рядка (перший операнд) із заданим шаблоном (другий операнд). Якщо рядок відповідає шаблону, результатом операції буде значення **True**, якщо не відповідає – значення **False**. Другий операнд операції **Like** може містити так звані групові або шаблонні символи, які операція **Like** ставить у відповідність символам свого першого операнду згідно з таблицею 1.2.

Про методи порівняння текстових рядків:
- **Option Compare Statement**

Для задання відповідності символам **?**, *, **#** та **!** їх можна брати у квадратні дужки. Для задання відповідності символу **[** його можна застосовувати сам по собі.

У квадратних дужках можна також задавати діапазони символів за допомогою символу мінус (–). Наприклад, всі великі літери англійського алфавіту можна задати так: [A–Z]. Декілька діапазонів всередині квадратних дужок задаються без будь-якого розділення.

Таблиця 1.2

Групові символи оператора **Like**

Груповий символ	Чому відповідає
?	один будь-який символ
*	нуль або більше будь-яких символів
#	будь-яка одна цифра (0-9)
[*charlist***]**	один із символів, що задані в квадратних дужках
[!*charlist***]**	будь-який один символ, що відсутній у квадратних дужках

Якщо символ мінус (–) зустрічається на початку або в кінці квадратних дужок, то він задає відповідність самому собі, інакше вважається, що він задає діапазон символів.

Подана нижче програма перевіряє, чи відповідає назва файлу, введена користувачем, формату назв багатотомних архівів архіватора rar:

```
Dim file_name As String
file_name = InputBox("Введіть назву файлу:")
If file_name Like "*.[rR][aA][rR]" Then
MsgBox "Файл є початком багатотомного rar-архіву"
ElseIf file_name Like "*.[rR]##" Then
MsgBox "Файл є продовженням багатотомного rar-архіву"
Else
MsgBox "Файл не є rar-архівом"
EndIf
```

1.12. Логічні (бітові) операції

Результат виконання однієї і тієї самої логічної (бітової) операції залежить від типів, які мають її операнди. В тому випадку, коли всі операнди операції мають тип **Boolean**, операція виконується як логічна. Якщо хоча

б один операнд має тип даних, відмінний від **Boolean**, відбувається автоматичне перетворення значень операндів до цілочислового типу даних (див. п. 1.10) і операція виконується як бітова. Розглянемо яким чином формується результат логічних (бітових) операцій мови VBA.

Результатом логічної операції **Not** є логічне значення, протилежне до логічного значення її операнда.

Результатом бітової операції **Not** є числове значення, в якому всі біти мають значення, протилежні значенням відповідних бітів операнда операції.

Результатом логічної операції **And** є значення **True** тільки в тому випадку, коли обидва операнди мають значення **True**, інакше результатом логічної операції **And** є значення **False**.

Результатом бітової операції **And** є числове значення, в якому значення 1 мають тільки ті біти, які мали значення 1 в обох операндах операції. Всі інші біти результату операції **And** мають значення 0.

Результатом логічної операції **Or** є значення **False** тільки в тому випадку, коли обидва операнди мають значення **False**, інакше результатом логічної операції **Or** є значення **True**.

Результатом бітової операції **Or** є числове значення, в якому значення 0 мають тільки ті біти, які мали значення 0 в обох операндах операції. Всі інші біти результату операції **Or** мають значення 1.

Результатом логічної операції **Xor** є значення **True** тільки в тому випадку, коли обидва операнди мають різні значення, інакше результатом логічної операції **Xor** є значення **False**.

Результатом бітової операції **Xor** є числове значення, в якому значення 1 мають тільки ті біти, які мали різні значення в обох операндах операції. Всі інші біти результату операції **Xor** мають значення 0.

Результатом логічної операції **Eqv** є значення **False** тільки в тому випадку, коли обидва операнди мають різні значення, інакше результатом логічної операції **Eqv** є значення **True**.

Результатом бітової операції **Eqv** є числове значення, в якому значення 0 мають тільки ті біти, які мали різні значення в обох операндах операції. Всі інші біти результату операції **Eqv** мають значення 1.

Результатом логічної операції **Imp** є значення **False** тільки в тому випадку, коли значення першого операнда **True**, а другого операнда **False**, інакше результатом логічної операції **Imp** є значення **True**.

Результатом бітової операції **Imp** є числове значення, в якому зна-

чення 0 мають тільки ті біти, які мали значення 1 в першому операнді та значення 0 в другому операнді операції. Всі інші біти результату операції **Imp** мають значення 1.

Розглянутий вище принцип роботи бітових операцій ілюструють приклади, наведені в табл. 1.3.

Таблиця 1.3

Результати бітових операцій над цілими операндами

Операція	Значення операндів у двійковій системі числення		Результат операції у двійковій системі числення
	першого	другого	
Not	1100		0011
And	1100	1010	1000
Or	1100	1010	1110
Xor	1100	1010	0110
Eqv	1100	1010	1001
Imp	1100	1010	1011

```
Dim a As Integer, b As Integer
a = 7
b = (a > 5) And a
```

Оскільки у наведеній вище програмі другий операнд операції **And** має тип даних **Integer**, то результат виразу (a > 5), який виступає як перший операнд операції **And**, теж перетворюється до типу **Integer**. Результатом виразу (a > 5) є значення **True** типу **Boolean**, яке в даному випадку перетворюється у значення -1. В змінну b заноситься результат бітової операції **And** над двійковими значеннями 1111111111111111 (відповідає десятковому значенню -1) та 0000000000000111 (відповідає десятковому значенню 7). Результатом бітової операції **And** над такими двійковими значеннями буде двійкове значення 0000000000000111 (десяткове 7), яке і потрапляє у змінну b.

2. ОПЕРАТОРИ, ПРОЦЕДУРИ, ФУНКЦІЇ

2.1. Оператори

Оператор – синтаксично закінчена інструкція у програмі. Може містити ключові слова та вирази.

Всі оператори мови програмування VBA поділяються на три категорії:

1) ***оператори опису***, які задають змінні, константи, процедури та функції, типи даних користувача;

2) ***оператори присвоєння***, які заносять значення у змінні;

3) ***виконувані оператори***, які ***ініціюють*** дію (наприклад, виклик функції, виконання циклу або обчислення виразу).

Серед виконуваних операторів також виділяють:

- ***оператори вибору (умовні оператори)***;
- ***оператори циклу***;
- ***оператори переходу***.

Кожен оператор VBA записується в окремому текстовому рядку. Символ підкреслення (_), який називається ***символом продовження рядка***, може бути використаний для продовження одного оператора у наступному текстовому рядку. Для того, щоб записати декілька операторів в одному текстовому рядку, необхідно відокремити їх між собою за допомогою символу двокрапки (:).

Оператори програми виконуються послідовно, за винятком випадків, коли якийсь оператор (наприклад, оператор циклу або оператор переходу) спеціально змінює цю послідовність.

2.2. Оператори присвоювання

В мові програмування VBA є два оператори присвоювання: оператор **Let** та оператор **Set**. Оператор **Set** використовується тільки для створення нових об'єктів та присвоювання об'єктним змінним адрес існуючих об'єктів. Оператор **Let** використовується для присвоювання значень змінним усіх інших типів.

Оператор присвоєння **Let** має такий синтаксис:

[Let] *ident* = *exp*

ident – ідентифікатор, який задає назву змінної.
exp – вираз.

Оператор присвоєння **Let** обчислює значення виразу *exp*, за необхідності перетворює його (див. п. 1.10) до типу даних, який має змінна *ident*, та заносить отримане значення у змінну *ident*. Якщо автоматичне перетворення значення виразу *exp* до типу даних, який має змінна *ident*, неможливе, виникає помилка.

Оператор присвоювання **Set** має такий синтаксис:

Set *ident* = {**New** *type* | *exp*}

ident – ідентифікатор, який задає назву змінної об'єктного типу **Object** або типу **Variant**.
type – об'єктний тип даних, для якого дозволено створення нових об'єктів (наприклад, тип **Collection**). Крім того, якщо змінна *ident* має об'єктний тип даних, відмінний від **Object** та **Variant**, тип *type* повинен бути таким самим типом, який має змінна *ident*.
exp – вираз, результат обчислення якого повинен бути перетворюваним до об'єктного типу даних.

При використанні ключового слова **New**, оператор **Set** створює в пам'яті новий об'єкт типу *type* та заносить його адресу у змінну з назвою *ident*.

Якщо ключове слово **New** не вказано, оператор **Set** обчислює значення виразу *exp*, перетворює його (див. п. 1.10) у об'єктний тип даних, після чого за необхідності перетворює його до типу даних, який має змінна *ident*, та заносить отримане значення у цю змінну *ident*.

Якщо автоматичне перетворення значення виразу *exp* у об'єктний тип даних або подальше його перетворення до типу даних, який має змінна *ident*, неможливе, то виникає помилка.

У наведеній далі програмі використано два оператори **Let** та один оператор **Set**. В переважній більшості випадків під час запису оператора **Let** ключове слово **Let** не вказують. Саме так записано оператор **Let** у третьому рядку наведеної вище програми.

```
Dim a As Integer
Let a = 10
a = a+2
Dim b As Object
Set b = Nothing
```

2.3. Оператори вибору

Оператори вибору призначені для вибору одного із декількох шляхів виконання програми залежно від виконання однієї з умов із заданого набору умов. До операторів вибору відносяться оператор **If** та оператор **Select Case**.

Оператор вибору **If** має такий синтаксис:

```
If exp Then
    [statement]
[ElseIf exp Then
    [statement]] ...
[Else
    [statement]]
End If
```

exp – вираз.
statement – один або більше операторів.

Формально наведене вище правило задає синтаксис одразу для чотирьох операторів: **If**, **ElseIf**, **Else** та **End If**. Але через те, що оператори **ElseIf**, **Else** та **End If** не можуть використовуватися у програмі без відповідного їм оператора **If**, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **If**" буде розумітися сукупність всіх чотирьох наведених вище операторів.

Алгоритм виконання оператора **If** наведено нижче.

1. Обчислюється значення виразу *exp*, вказаного після ключового слова **If**, та перетворюється (див. п. 1.10) до типу даних **Boolean**. Якщо отримане значення дорівнює **True**, виконуються оператори *statement*, розташовані в наступному рядку програми, і виконання оператора **If** закінчується. Інакше відбувається перехід до наступного кроку.
2. Обчислюється значення виразу *exp*, вказаного після ключового

слова **ElseIf**, та перетворюється до типу даних **Boolean**. Якщо отримане значення дорівнює **True**, виконуються оператори *statement*, що розташовані в наступному рядку програми, і виконання оператора **If** закінчується. Інакше відбувається перехід до наступного кроку.

3. Дії, аналогічні кроку 2, повторюються для кожного ключового слова **ElseIf**.

4. Виконуються оператори *statement*, розташовані після ключового слова **Else**, після чого виконання програми продовжується з оператора, розташованого після оператора **If**.

За відсутності операторів **ElseIf** та **Else** відповідні їм кроки алгоритму пропускаються.

Оператор **If** повинен бути першим оператором в текстовому рядку програми, тобто він не може розташовуватися після символу двокрапки.

```
Dim a As Integer, b As Integer
a = 7
If a Then
    b = 100 / a
End If
```

Наведена вище програма заносить у змінну b результат ділення числа 100 на значення змінної a. Ділення відбувається тільки в тому випадку, коли значення змінної a відмінне від 0.

Спочатку значення змінної a, яка має тип даних **Integer**, перетворюється до типу **Boolean**. При цьому числове значення 7 перетворюється у значення **True**. Оскільки результат виразу, який знаходиться після ключового слова **If**, відмінний від нуля, то виконується оператор присвоювання b = 100/a.

Оператор вибору **Select Case** має такий синтаксис:

```
Select Case exp-select
    [Case explist
        [statement]] ...
    [Case Else
        [statement]]
End Select
```

exp-select – вираз, результат якого повинен мати числовий тип, тип **Boolean** або **String**.

statement – один або більше операторів.
explist має такий синтаксис:

{*exp-eq* | *exp-from* **To** *exp-to* | **Is** *compop exp-is*} [, _
{*exp-eq* | *exp-from* **To** *exp-to* | **Is** *compop exp-is*}] ...

exp-eq, *exp-from*, *exp-to*, *exp-is* – вирази. Значення виразу *exp-from* повинно бути меншим значення виразу *exp-to*.
compop – операція порівняння (див. п. 1.11).

Ключове слово **Is**, яке використовується в *explist*, не є в даному контексті операцією **Is** (див. п. 1.11).

Формально наведене вище правило задає синтаксис одразу для чотирьох операторів: **Select Case**, **Case**, **Case Else** та **End Select**. Але через те, що оператори **Case**, **Case Else** та **End Select** не можуть використовуватися у програмі без відповідного їм оператора **Select Case**, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **Select Case**" буде розумітися сукупність всіх чотирьох наведених вище операторів.

Нижче наведено алгоритм виконання оператора **Select Case**.

1. Якщо оператори **Case** відсутні, відбувається перехід до кроку 4, інакше для розгляду обирається *explist* першого оператора **Case**.

2. Для кожного *exp-eq* із *explist* оператора **Case**, що розглядається, утворюються вирази (*exp-select* = *exp-eq*), для кожної пари *exp-from*, *exp-to* утворюються вирази (*exp-select* >= *exp-from* **And** *exp-test* <= *exp-to*), для кожного *exp-is* утворюються вирази (*exp-select compop exp-is*). Всі утворені вирази об'єднуються в заданій послідовності в один вираз за допомогою операцій **Or**. Обчислюється значення отриманого виразу. Якщо воно дорівнює **True**, виконуються оператори *statement*, задані після оператора **Case**, що розглядається, після чого виконання програми продовжується з оператора, розташованого після оператора **Select**.

3. Для розгляду обирається наступний оператор **Case** та відбувається перехід до кроку 2. В разі відсутності наступного оператора **Case** відбувається перехід до кроку 4.

4. Виконуються оператори *statement*, розташовані після ключових слів **Case Else**, після чого виконання програми продовжується з оператора, розташованого після оператора **End Select**.

```
Dim a As Integer
a = InputBox("Введіть ціле число a")
Select Case a
        Case 20 To 30
                MsgBox "a>=20 та a<=30"
        Case 15, Is > 40
                MsgBox "a=15 або a>40"
        Case Else
                MsgBox "a - це щось інше"
End Select
```

Дана програма запрошує користувача ввести число. Залежно від введеного користувачем числа програма відображає на екрані текстовий рядок *"a>=20 та a<=30"* у випадку, якщо введене число знаходиться в діапазоні від 20 до 30 включно. Якщо введене число дорівнює 15 або більше за 40 на екрані відображається текстовий рядок *"a=15 або a>40"*. У випадку будь-якого іншого значення введеного числа на екрані відображаться повідомлення "*a - це щось інше*".

2.4. Оператори циклу

Оператор циклу призначений для виконання групи довільних операторів, які називаються ***тілом циклу***, декілька разів. Кожне повторення тіла циклу називається ***ітерацією***. Тіло циклу виконується в тому випадку, якщо значення деякого виразу, який називається ***умовою виконання циклу***, дорівнює **True**.

До операторів циклу відносяться оператори **For**, **Do** та **While**.

Оператор циклу **For** має такий синтаксис:

For *ident* = *exp-start* **To** *exp-end* [**Step** *exp-step*]
    [*statement*]
Next [*ident*]

ident – змінна чисельного типу, яку називають ***лічильником циклу***.
exp-start – вираз, який задає початкове значення змінної *ident*
exp-end – вираз, який задає кінцеве значення змінної *ident*
exp-step – вираз, який задає величину змінення значення змінної *ident* після кожної ітерації циклу (значення цього виразу називається ***кроком циклу***).

statement – один або більше операторів.

Формально наведене вище правило задає синтаксис одразу для двох операторів: **For** та **Next**. Але через те, що оператор **Next** не може використовуватися у програмі без відповідного йому оператора **For**, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **For**" буде розумітися сукупність обох наведених вище операторів.

Як *ident* в операторі **For** та операторі **Next** повинна бути вказана одна і та сама змінна.

Якщо ключове слово **Step** не вказано, береться крок циклу, рівний 1. Значення кроку циклу визначає умову виконання циклу, яка перевіряється перед кожною ітерацією циклу. Якщо значення кроку циклу додатне число або 0, то умова виконання циклу *ident* <= *exp-end*, інакше умова виконання циклу *ident* >= *exp-end*.

Алгоритм виконання оператора **For** наведено нижче.

1. У змінну *ident* заноситься значення виразу *exp-start*.
2. Перевіряється умова виконання циклу. Якщо вона не виконується, цикл закінчується і виконання програми продовжується з оператора, який розташований після оператора **For**.
3. Виконуються оператори *statement*.
4. До значення змінної *ident* (лічильника циклу) додається значення кроку циклу і відбувається перехід до кроку 2.

Оператор циклу **Do** має дві синтаксичні форми.

Синтаксична форма 1:

Do [{**While** | **Until**} *exp*]
    [*statement*]
Loop

Синтаксична форма 2:

Do
    [*statement*]
Loop [{**While** | **Until**} *exp*]

exp – вираз, значення якого визначає умову виконання циклу.
statement – один або більше операторів.

Формально кожна із наведених вище синтаксичних форм задає син-

таксис одразу для двох операторів: **Do** та **Loop**. Але через те, що оператор **Loop** не може використовуватися у програмі без відповідного йому оператора **Do** і навпаки, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **Do**" буде розумітися сукупність обох наведених вище операторів.

При використанні ключового слова **While** умова виконання циклу буде мати вигляд *exp* = **True**, а при використанні ключового слова **Until** умова виконання циклу буде мати вигляд *exp* <> **True**.

Алгоритм виконання оператора **Do** синтаксичної форми 1.

1. Перевіряється умова виконання циклу. Якщо вона не виконується – цикл закінчується і виконання програми продовжується з оператора, який розташований після циклу **Do**.
2. Виконуються оператори *statement*.
3. Відбувається перехід до кроку 1.

Алгоритм виконання оператора **Do** синтаксичної форми 2.

1. Виконуються оператори *statement*.
2. Перевіряється умова виконання циклу. Якщо вона виконується – відбувається перехід до кроку 1, інакше цикл закінчується і виконання програми продовжується з оператора, який розташований після циклу **Do**.

Оператор циклу **While** має такий синтаксис:

While *exp*
[*statement*]
Wend

exp – вираз, значення якого визначає умову виконання циклу.
statement – один або більше операторів.

Формально наведене вище правило задає синтаксис одразу для двох операторів: **While** та **Wend**. Але через те, що оператор **Wend** не може використовуватися у програмі без відповідного йому оператора **While**, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **While**" буде розумітися сукупність обох наведених вище операторів.

Умова виконання циклу **While** має вигляд *exp* = **True**.

Алгоритм виконання оператора **While**.

1. Перевіряється умова виконання циклу. Якщо вона не виконується, цикл закінчується і виконання програми продовжується з оператора,

який розташований після оператора **While**.

2. Виконуються оператори *statement*.
3. Відбувається перехід до кроку 1.

```
Dim a As Integer
For a = 1 To 10 Step 2
    MsgBox a
Next a
While a
    a = a - 1
    MsgBox a
Wend
```

За допомогою циклу **For** у наведеній вище програмі на екран послідовно виводяться цілі непарні числа від 1 до 9 включно. Після завершення циклу **For** змінна a набуває значення 11.

За допомогою циклу **While**, розташованого далі, на екран послідовно виводяться цілі числа від 10 до 0 включно.

2.5. Оператори переходу

До операторів переходу відносяться оператори **Exit For**, **Exit Do**, **GoTo**, які мають такий синтаксис:

```
Exit For
Exit Do
GoTo line
```

line – мітка рядка програми або номер рядка програми.

Оператор **Exit For** закінчує виконання найглибше вкладеного циклу **For**, всередині якого даний оператор знаходиться. Виконання програми продовжується з оператора, який розташований одразу після циклу **For**.

Оператор **Exit Do** закінчує виконання найглибше вкладеного циклу **Do**, всередині якого даний оператор знаходиться. Виконання програми продовжується з оператора, який розташований одразу після циклу **Do**.

Оператор **GoTo** передає управління на рядок програми з міткою або номером *line*. Цей рядок програми повинен розташовуватися в тій самій процедурі або функції, що і оператор **GoTo**.

Мітка рядка програми – ідентифікатор, який позначає рядок тексту програми. Повинен розташовуватися на самому початку текстового рядка та закінчуватися символом двокрапки.

Номер рядка програми – число, яке позначає рядок тексту програми. Повинен розташовуватися на самому початку текстового рядка та закінчуватися символом двокрапки.

```
Dim a As Integer, s As Integer
Do While 1
    a = InputBox("Enter a")
    If a < 0 Then
        Exit Do
    End If
    s=s+1
Loop
MsgBox s
```

Наведена вище програма підраховує суму додатних цілих чисел, введених користувачем, та виводить її на екран. Цикл **Do While** виконується до тих пір, поки користувач не введе від'ємне ціле число, в результаті чого виконається оператор **Exit Do** і цикл буде завершено.

2.6. Процедури та функції

Процедура, ***функція*** – іменована послідовність команд, які виконуються як одне ціле. ***Процедура*** не повертає значення, а ***функція*** – повертає. Значення, що повертається, може використовуватися у виразах.

Оскільки ***процедури*** та ***функції*** мають багато спільного, надалі все сказане щодо ***процедури*** буде справедливо і для ***функції***, якщо інше не обумовлено спеціально.

В момент виклику процедури створюються автоматичні по відношенню до неї змінні (див. п. 1.9) з іменами, які були вказані під час опису процедури за допомогою оператора **Sub**. Кожна така змінна має область дії – відповідну процедуру та називається ***аргументом процедури***.

Якщо ***аргумент процедури*** передається ***за значенням*** (з використанням ключового слова **ByVal**), то аргумент процедури буде містити те значення, яке задане у місці виклику процедури. При цьому значення змінної, яка передається через аргумент в процедуру, після виконання процедури

не буде змінено.

Якщо ***аргумент процедури*** передається ***за посиланням*** (з використанням ключового слова **ByRef**), то аргумент процедури буде містити адресу тієї зовнішньої змінної, яка була вказана на місці відповідного аргументу під час виклику процедури. За рахунок цього нове значення, яке присвоюється такому аргументу всередині процедури, потрапляє у зовнішню змінну, яка була вказана на місці відповідного аргументу під час виклику процедури. Тобто значення змінної, яка передається через аргумент в процедуру, після виконання процедури може бути змінено.

Якщо під час виклику процедури як аргумент, що передається ***за посиланням***, виступає значення деякого виразу, то в аргумент процедури потрапляє адреса тимчасової змінної, яка містить результат обчислення відповідного виразу. Така тимчасова змінна створюється перед викликом процедури та знищується після її завершення.

Синтаксично використання аргументу, переданого ***за посиланням***, всередині процедури нічим не відрізняється від використання аргументу, переданого ***за значенням***.

Таким чином, ***аргумент процедури*** – це автоматична змінна з областю дії процедура, яка створюється в момент виклику процедури та ініціалізується (див. п. 1.5) значенням, яке вказано у тому місці програми, в якому викликається дана процедура.

Функції та процедури описуються за допомогою операторів опису **Sub** та **Function**, відповідно.

Оператор опису процедури **Sub** задає назву, аргументи та код, який утворює тіло процедури. Він має такий синтаксис:

[Private|Public] **[Static]** Sub ident **[([** arg **[,** arg**]**... **])]**
[statement]
End Sub

ident – назва процедури.
statement – один або декілька операторів, які утворюють тіло процедури та будуть виконуватися після виклику процедури. Серед даних операторів не може бути операторів **Sub** та **Function**.
arg – аргумент процедури, має такий синтаксис:

[Optional] **[ByVal | ByRef]** *ident* **[As** *type***]** **[=** *exp***]**

ident – назва аргументу. Якщо аргументом є масив, після назви аргументу необхідно вказувати круглі дужки.

type – тип аргументу.

exp – константний вираз (див. п. 1.6). Результат цього виразу задає ***значення за замовчуванням*** для ***необов'язкового аргументу*** (тобто те значення, яке отримає відповідний необов'язковий аргумент процедури, якщо в місці виклику процедури не було явно вказано значення для даного аргументу).

Формально наведені вище правила задають синтаксис одразу для двох операторів: **Sub** та **End Sub**. Але через те, що оператор **End Sub** не може використовуватися у програмі без відповідного йому оператора **Sub**, в даному посібнику, так само як і в документації до мови програмування VBA, під терміном "оператор **Sub**" буде розумітися сукупність обох операторів.

При використанні ключового слова **Private** процедура буде мати область дії модуль. При використанні ключового слова **Public** процедура буде мати область дії проект і до неї можна буде звертатись з будь-якого місця проекту. Якщо не вказано жодного з ключових слів **Private** або **Public**, процедура буде мати область дії проект.

Якщо при описанні процедури задане ключове слово **Static**, всі змінні, описані всередині такої процедури, будуть мати статичний час життя.

Ключове слово **Optional** вказує на те, що відповідний аргумент процедури є необов'язковим, тобто його можна не вказувати під час виклику процедури. Якщо дане ключове слово використане для деякого аргументу, всі наступні аргументи також повинні бути необов'язковими.

Ключове слово **ByVal** вказує на те, що аргумент передається в процедуру ***за значенням***. Ключове слово **ByRef** – вказує на те, що аргумент передається в процедуру ***за посиланням***. Якщо не вказано жодного з ключових слів **ByVal** та **ByRef**, аргумент передається ***за посиланням***.

Масиви та структури не можуть бути ***необов'язковими аргументами***. Також масиви та структури можуть передаватися тільки ***за посиланням***.

Оператор опису функції **Function**, який задає назву, аргументи, тип значення, що повертається функцією, а також код, який утворює тіло функції, має такий синтаксис:

[Private | Public] [Static] Function *ident* **[(**[*arg* [**,** *arg*]...]**)] _**
[As *type*]
[*statement*]
End Function

type – тип значення, яке повертається функцією. Якщо тип не вказаний, функція буде повертати значення типу **Variant**.

В усьому іншому оператор опису функції **Function** повністю аналогічний оператору опису процедури **Sub**.

В момент виклику функції, окрім її аргументів автоматично створюється ще одна спеціальна змінна, областю дії якої є відповідна функція. Ім'я цієї змінної збігається з іменем самої функції, а її тип збігається із типом значення, яке повертається даною функцією. Використання даної змінної в тілі функції нічим не відрізняється від використання будь-якої іншої змінної. Функція буде повертати те значення, яке знаходиться в цій змінній на момент завершення свого виконання.

Для передання управління процедурі або функції (тобто виклику процедури або функції) використовується оператор **Call**. Він має такий синтаксис:

[Call] *ident* [*callarglist*]

Про іменовані аргументи функцій та процедур:
- **Calling Sub and Function Procedures**

callarglist – перелік виразів, відокремлених комою, кожен з яких задає значення для відповідного аргументу із заданого під час опису процедури або функції переліку аргументів. Якщо аргумент є необов'язковим (описаний за допомогою ключового слова **Optional**), вираз, який задає значення для даного аргументу у відповідному місці *callarglist*, може бути опущений. Після необов'язкового аргументу, значення якого в *callarglist* опущено, необхідно ставити знак коми, якщо надалі в *callarglist* вказується значення одного із наступних аргументів.

Якщо ключове слово **Call** вказано, то перелік *callarglist* повинен бути взятий у круглі дужки. Якщо ключове слово **Call** не вказано, то перелік *callarglist* задається без дужок.

Якщо оператор **Call** використовується для виклику функції і ключове слово **Call** вказано, значення, що повертається функцією, втрачається.

Ім'я функції також може використовуватися у виразах. При обчисленні такого виразу відбувається виклик функції і замість її імені у вираз підставляється значення, яке повертається функцією. Якщо функція викликається у такий спосіб, то перелік *callarglist* повинен бути взятий у круглі дужки.

Для дострокового завершення процедури та повернення в місце виклику процедури, в тілі процедури можна використовувати оператор **Exit Sub** (у випадку функції – оператор **Exit Function**). Оператор **Exit Sub** має такий синтаксис:

```
Exit Sub
```

Як приклад розглянемо таку програму:

```
Static Function f(a As Integer, _
            Optional b As Integer = 5)
    MsgBox a+b
    Dim n As Integer
    n = n + 1
    f = n
End Function

Sub s()
    Call f(3)
    MsgBox f(2, 2)
End Sub
```

Функція f відображає на екрані суму значень своїх аргументів та повертає кількість своїх викликів. Дійсно, оскільки функція описана з використанням ключового **Static**, то всі змінні, описані всередині функції мають ***статичний час життя***, а отже, зберігають своє значення після завершення роботи функції. Тому під час кожного виклику функції f значення змінної n буде відповідати кількості викликів функції f.

Функція f викликається із процедури s два рази. Під час першого виклику значення для другого аргументу функції f не вказано, а отже другий аргумент функції f отримає значення 5, яке вказано як значення за замовчуванням в операторі опису функції f. В результаті функція f виведе на екран суму 3+5, тобто число 8. Оскільки перший раз функція була викликана за допомогою оператора **Call**, то значення, яке функція повернула, втрача-

ється. Значення, яке функція повертає під час другого виклику, виводиться на екран за допомогою функції **MsgBox**.

2.7. Області дії процедур та функцій

Процедури та функції мають тільки дві області дії: модуль і проект.

Процедури та функції, описані з використанням ключового слова **Private**, мають область дії модуль. Процедура або функція з областю дії модуль доступна (тобто може бути викликана) з усіх процедур та функцій тільки того модуля, в якому вона описана.

Процедури та функції, описані з використанням ключового слова **Public** або без використання ключового слова **Private**, мають область дії проект. Процедура або функція з областю дії проект доступна з усіх процедур та функцій всіх модулів проекту.

Так само, як і у випадку змінних, опис процедури або функції та його місце розташування у тексті програми визначають область дії цієї процедури або функції.

```
Sub test1()
    Call test2
    Call test3
End Sub

Private Sub test2()
    Call test1
    Call test3
End Sub

Public Function test3()
    Call test1
    Call test2
End Function
```

У наведеній вище програмі тільки процедура test2() має область дії модуль, тобто не може бути викликана із процедур інших модулів проекту. Процедура test2() може викликатися тільки із процедур та функцій, розташованих у тому самому модулі, в якому описана процедура test2(). Тобто процедуру test2() можна викликати із процедур test1(), test2() та функції test3().

Процедура test1() і функція test3() мають область дії проект і тому їх можна викликати із будь-якої процедури чи функції будь-якого модуля проекту.

2.8. Оператори та функції для роботи з файлами

Перш ніж виконати будь-яку операцію із вмістом файлу, його необхідно відкрити. Під час відкриття файлу задається один із таких режимів доступу до вмісту файлу:

- тільки для читання;
- тільки для записування;
- для читання та записування одночасно.

Виконання операції над файлом, яка суперечить режиму доступу до файлу, призводить до виникнення помилки. Після закінчення усіх операцій із файлом його необхідно закрити. Після того, як файл буде закрито, його можна знову відкривати для виконання операцій читання/записування.

Для кожного відкритого файлу операційна система зберігає значення ***файлового покажчика***, який задає позицію (номер байта від початку файлу), з якої буде виконуватися наступна операція читання або записування. Після виконання кожної операції читання або записування значення ***файлового покажчика*** автоматично збільшується на кількість прочитаних або записаних байтів. Програма може спеціально встановити ***файловий покажчик*** у довільну позицію (задати йому довільне значення).

Оператор **Open** призначений для відкриття файлу і має такий синтаксис:

Open *exp* **For** *mode* [*lock*] **As** [#]*filenumber*

exp – рядковий вираз, результат якого задає шлях та ім'я файлу, який необхідно відкрити.

mode – одне з ключових слів, які визначають режим доступу до файлу:

- **Append** – режим доступу – тільки для записування; файловий покажчик встановлюється в кінець файлу; якщо вказаний файл не існує, створюється новий файл;
- **Input** – режим доступу – тільки для читання; файловий покажчик встановлюється на початок файлу; якщо вказаний файл не існує – виникає помилка;

- **Output** – режим доступу – тільки для записування; якщо вказаний файл існує, його розмір встановлюється рівним нулеві і дані, що зберігалися в ньому, втрачаються; якщо вказаний файл не існує, створюється новий файл;
- **Binary** – режим доступу – для читання та записування; файловий покажчик встановлюється на початок файлу; якщо вказаний файл не існує, створюється новий файл. Єдиний режим, який дозволяє використовувати для даного файлу оператори **Get** та **Put**.

lock – одне з ключових слів, що задає, в якому режимі доступу вказаний файл може бути одночасно відкритий із іншої програми:

- **Shared** – в режимі для читання та записування;
- **Lock Read** – тільки в режимі для записування;
- **Lock Write** – тільки в режимі для читання;
- **Lock Read Write** – не може бути відкритий одночасно іншою програмою.

filenumber – ціле число із діапазону від 1 до 511 включно, яке пов'язується із файлом, що відкривається. При використанні усіх інших операторів, призначених для роботи з файлами, необхідно вказувати даний номер для того, щоб ідентифікувати відкритий файл, з яким необхідно виконувати ту чи іншу операцію. Якщо в програмі одночасно відкривається декілька файлів, кожному з них повинен бути присвоєний унікальний номер.

Оператор **Close** призначений для закриття одного чи декількох файлів і має такий синтаксис:

Close [[#]*filenumber*] [,[#]*filenumber*] ...

filenumber – номер відкритого раніше файлу. Після закриття файлу номер *filenumber* стає вільним та може бути знову використаний в операторі **Open**.

Якщо в операторі **Close** не вказано жодного файлового номера, закриваються всі відкриті раніше файли.

Оператор **Reset** закриває всі відкриті раніше файли і має такий синтаксис:

Reset

Оператор **Line Input** зчитує послідовність символів від поточної по-

зиції файлового покажчика до кінця текстового рядка або кінця файлу (залежно від того, що буде досягнуто раніше) з відкритого файлу та заносить її у вказану змінну. Може використовуватися тільки для файлів, відкритих в режимах **Input** та **Binary**. Оператор **Line Input** має такий синтаксис:

Line Input #*filenumber*, *ident*

filenumber – номер відкритого раніше файлу.
ident – назва змінної типу **String** або **Variant**, в яку заноситься послідовність символів, прочитана із файлу. Символи з десятковими кодами 13 та 10, які в текстових файлах позначають кінець рядка тексту, теж зчитуються із файлу, але в дану змінну не заносяться.

Оператор **Input** зчитує дані із відкритого файлу та заносить їх у вказані змінні. Може використовуватися тільки для файлів, відкритих в режимах **Input** та **Binary**. Оператор **Input** має такий синтаксис:

Input #*filenumber*, *varlist*

filenumber – номер відкритого раніше файлу.
varlist – перелік змінних, розділених комами, в які заносяться дані, прочитані із файлу.

Оператор **Input** розглядає файл, із якого зчитуються дані, як текстовий файл, в якому окремі елементи даних розділені пропусками та/або символами нового рядка та (необов'язково) комами. Зчитування даних розпочинається з позиції у файлі, заданої поточним значенням файлового покажчика. Символи пропуску, табуляції та нового рядка при зчитуванні кожного наступного елемента даних пропускаються. Значення першого зчитаного із файлу елемента даних після перетворення заноситься у першу змінну із переліку *varlist*, значення другого елемента даних – у другу змінну і т.д.

Кожний елемент даних, прочитаний із файлу, є текстом (тобто послідовністю символів), який оператор **Input** намагається перетворити до типу даних, який має відповідна змінна із переліку *varlist*. Якщо таке перетворення можливе, у відповідну змінну заноситься перетворене значення, а інакше – значення, яким ініціалізується змінна даного типу (див. п. 1.5), після чого виконання оператора **Input** продовжується.

Послідовність символів, взятих у подвійні лапки ("), розглядається

оператором **Input** як окремий елемент даних, який може бути перетворений тільки у текстовий рядок. При такому перетворенні символи подвійних лапок опускаються, а символи пропуску, табуляції та нового рядка всередині подвійних лапок розглядаються як складова частина текстового рядка.

Якщо під час зчитування даних із файлу буде досягнуто кінця файлу, виникає помилка.

Оператор **Print** перетворює дані у текст та записує у відкритий файл. Може використовуватися тільки для файлів, відкритих в режимах **Output** та **Append**. Оператор **Print** має такий синтаксис:

Print #*filenumber*, [[*output-exp*] *output-sep*] ... [*output-exp*]

output-exp – має такий формат:

{ **Tab[(***n***)]** | **Spc(***n***)** | *exp* }

output-sep – має такий формат:

{ **,** | **;** }

filenumber – номер відкритого раніше файлу.

n – ціле невід'ємне число.

exp – вираз будь-якого типу, окрім об'єктного та типу даних користувача, результат якого перетворюється у текст (послідовність символів) і записується у файл.

Призначення ключових слів **Tab** та **Spc**, а також ***пунктуаторів*** **,** та **;** в операторі **Print** таке:

1. **Spc**(*n*) – у файл виводиться *n* символів пропуску;
2. **Tab** – якщо кількість символів X, записана даним оператором **Print** у файл до цього моменту (починаючи від першого виведеного символу або від останньої виведеної пари символів з кодами 13 та 10) вже є кратною 14, у файл виводиться 14 символів пропуску, інакше – стільки символів пропуску, скільки необхідно для того, щоб кількість символів X стала кратною 14;
3. **Tab**(*n*) – якщо кількість символів X, записана даним оператором **Print** у файл до цього моменту (починаючи від першого виведеного символу або від останньої виведеної пари символів з кодами 13 та 10) вже до-

рівнює або більша за n, у файл виводяться символи з кодами 13 та 10, після чого записується n-1 символ пропуску, інакше – стільки символів пропуску, скільки необхідно для того, щоб кількість символів X стала рівною n;

4. **;** – не приводить до виведення у файл ніяких додаткових символів;

5. **,** – те саме, що **Tab**.

Перетворення та записування у файл результату кожного виразу *exp* відбувається за такими правилами.

1. Якщо результатом виразу є число, воно перетворюється у текст (послідовність символів), яка записується у файл. Як десятковий роздільник виступає символ, заданий в регіональних налаштуваннях операційної системи.

2. Якщо результатом виразу є текстовий рядок, він записується у файл без змін.

3. Якщо результатом виразу є значення типу **Date**, воно перетворюється у текст згідно з форматом короткої дати, заданим у регіональних налаштуваннях операційної системи.

4. Якщо результатом виразу є значення **True**, **False** або **Null**, у файл записується текст **True**, **False** або **Null**, відповідно.

5. Якщо результатом виразу *exp* є значення **Empty**, для даного виразу *exp* у файл нічого не записується.

Якщо наприкінці оператора **Print** відсутній *output-sep*, у файл додатково виводиться пара символів з кодами 13, 10. Зокрема, це означає, що наступний виклик оператора **Print**:

Print #1,

виведе у файл тільки символи з кодами 13, 10.

Оператор **Write** призначений для перетворення даних у текст та запису їх у відкритий файл. Може використовуватися тільки для файлів, відкритих в режимах **Output** та **Append**.

Оператор **Write** має такий самий синтаксис, як і оператор **Print**, але порівняно з оператором **Print** оператор **Write** має ряд відмінностей у функціонуванні, які описані нижче.

Призначення ключових слів **Tab** та **Spc**, а також ***пунктуаторів*** **,** та **;** в операторі **Write** таке саме, як і в операторі **Print** за винятком:

1. **Tab** – не приводить до виведення у файл ніяких додаткових символів;

2. **Tab**(*n*) – якщо кількість символів X, записана даним оператором

Write у файл до цього моменту (починаючи від першого виведеного символу або від останньої виведеної пари символів з кодами 13 та 10) вже дорівнює або більша за n, у файл виводяться символи з кодами 13 та 10, після чого записується n-1 символ пропуску, інакше – стільки символів пропуску, скільки необхідно для того, щоб кількість символів X стала рівною n;

3. **;** – якщо пунктуатору **;** передує *output-exp*, відмінний від **Tab**(*n*) та **Spc**(*n*), у файл виводиться символ коми, інакше у файл не виводиться ніяких додаткових символів;

4. **,** – приводить до виведення у файл символу коми.

Перетворення та запис у файл результату кожного виразу *exp* відбувається так само, як і в операторі **Print**, за винятком:

1. Якщо результатом виразу є число, воно перетворюється у текст (послідовність символів), який записується у файл. Як десятковий роздільник завжди виступає крапка, незалежно від регіональних налаштувань, заданих в операційній системі.

2. Якщо результатом виразу є текстовий рядок, він записується у файл взятим у подвійні лапки.

3. Якщо результатом виразу є значення типу **Date**, воно перетворюється у текст в форматі #yyyy-mm-dd hh:mm:ss#.

4. Якщо результатом виразу є значення **True**, **False** або **Null**, у файл записується текст #TRUE#, #FALSE# або #NULL#, відповідно.

Оператор **Put** призначений для записування у відкритий файл даних в бінарному форматі (тобто без перетворення у текст). Може використовуватися тільки для файлів, відкритих в режимі **Binary**.

Оператор **Put** має такий синтаксис:

Put [#]*filenumber*, [*offset*], *exp*

filenumber – номер відкритого файлу.

offset – додатне ціле число, яке задає нову позицію від початку файлу, у яку буде встановлено файловий покажчик перед записом у файл (номер байта від початку файлу: 1 – перший байт, 2 – другий байт і т.д.).

exp – вираз, результат обчислення якого безпосередньо без перетворення у текст записується у файл. Якщо результат обчислення виразу має тип **Variant**, спочатку у файл записується ціле число із двох байтів, яке вказує підтип значення типу **Variant**, а потім записується саме значення цього підтипу.

файлу та занесення його без перетворення у вказану змінну. Може використовуватися тільки для файлів, відкритих в режимі **Binary**.

Оператор **Get** має такий синтаксис:

Get [#]*filenumber*, [*offset*], *ident*

filenumber – номер відкритого файлу.
offset – додатне ціле число, яке задає нову позицію від початку файлу, у яку буде встановлено файловий покажчик перед читанням із файлу (номер байта від початку файлу: 1 – перший байт, 2 – другий байт і т.д.).
ident – змінна, у яку заноситься значення із N байтів, прочитане із файлу (N – розмір у байтах змінної *ident*). Якщо змінна має тип **Variant**, спочатку із файлу зчитується ціле число із двох байтів, яке вказує підтип значення типу **Variant**, а потім зчитується саме значення цього підтипу і заноситься в змінну *ident*.

Оператор **Seek** використовується для встановлення нового значення файлового покажчика. Оператор **Seek** має такий синтаксис:

Seek [#]*filenumber*, *position*

filenumber – номер відкритого файлу.
position – додатне ціле число, яке задає нове значення для файлового покажчика від початку файлу (1 – перший байт файлу, 2 – другий і т.д.).

Для роботи з файлами призначені також такі функції.

Function FreeFile() As Integer

Функція **FreeFile** повертає наступний вільний (невикористаний в операторі **Open**) на момент виклику функції файловий номер.

Function Loc(filenumber **As Integer**) **As Long**

Функція **Loc** повертає поточне значення файлового покажчика (0 – перший байт файлу, 1 – другий і т.д.) для файлу з номером *filenumber*, відкритого в режимі **Binary**.

```
Sub test()
    Open "test.txt" For Output As #1
    For i=1 To 10
        Write #1, i
    Next i
    Close #1
End Sub
```

Наведена вище програма відкриває файл "test.txt" за допомогою оператора **Open**, виводить в файл десять цілих чисел від 1 до 10 за допомогою оператора **Write** та закриває файл за допомогою оператора **Close**. Кожне наступне число виводиться у файл з нового рядка.

2.9. Функції MsgBox та InputBox

Функція **MsgBox** призначена для відображення на екрані діалогового вікна з текстовим повідомленням. Вона має такий синтаксис:

```
Function MsgBox(prompt As String, _
Optional buttons As Integer=vbOKOnly Or vbDefaultButton1, _
Optional title As String = app_name) As Integer
```

app_name – назва програми, в рамках якої функціонує VBA (наприклад, якщо VBA використовується в рамках середовища текстового редактора Microsoft Word, *app_name* буде текстовим рядком "Microsoft Word").


Про константи, пов'язані з функцією **MsgBox**:
- **MsgBox Constants**

Функція **MsgBox** відображає на екрані модальне діалогове вікно з повідомленням, яке задається текстовим рядком prompt, та кнопками, що задаються аргументом buttons. Повертає одну із цілочислових констант, що вказують, яка кнопка була натиснута у діалоговому вікні: **vbOK**, **vbCancel**, **vbAbort**, **vbRetry**, **vbIgnore**, **vbYes**, **vbNo**.

Аргумент prompt задає повідомлення, яке відображається у вікні. Максимальна допустима довжина повідомлення – 1024 символи. Спеціальні символи з кодами 13 та 10 можуть бути використані для розташування тексту повідомлення в декількох рядках.

Аргумент buttons є комбінацією наведених нижче констант-прапорців, які поділяються на декілька груп.

Група констант-прапорців, що задають кнопки, які будуть відображатися у вікні:

vbOKOnly – кнопка "OK";

vbOKCancel – кнопки "OK" та "Отмена" ("Cancel");

vbAbortRetryIgnore – кнопки "Прервать" ("Abort"), "Повтор" ("Retry") та "Пропустить" ("Ignore");

vbYesNoCancel – кнопки "Да" ("Yes"), "Нет" ("No") та "Отмена" ("Cancel");

vbYesNo – кнопки "Да" ("Yes") та "Нет" ("No");

vbRetryCancel – кнопки "Повтор" ("Retry") та "Отмена" ("Cancel").

Група констант-прапорців, що задають іконку, яка буде відображатися разом з повідомленням:

vbCritical – знак стоп;

vbQuestion – знак питання;

vbExclamation – знак оклику;

vbInformation – маленька літера і.

Група констант-прапорців, що задають кнопку за замовчуванням: **vbDefaultButton1**, **vbDefaultButton2**, **vbDefaultButton3**. Номер в імені константи вказує порядковий номер кнопки, на яку буде встановлено фокус (яка буде виділена) при виведенні вікна повідомлення.

Про інші можливості функції **MsgBox**:
- MsgBox Function

При заданні значення для аргументу buttons необхідно використовувати по одному прапорцю із кожної групи. Якщо прапорець із першої групи не задано, вікно буде містити тільки кнопку **vbOKOnly**. Якщо прапорець із другої групи не задано, вікно не буде містити іконки. Якщо прапорець із третьої групи не задано, кнопкою за замовчуванням буде перша кнопка.

Аргумент title задає заголовок діалогового вікна.

Наведена нижче програма відображає на екрані вікно з текстом "Вивести на друк результати роботи програми?". Вікно містить іконку із знаком запитання та двома кнопками "Так" та "Ні". У випадку, якщо користувач натисне кнопку "Так", програма викликає процедуру **PrintResults**.

```
If MsgBox "Вивести на друк результати роботи програми?", _
      vbYesNo Or vbQuestion = vbYes Then
      Call PrintResults
End If
```

Функція **InputBox** призначена для отримання від користувача інформації у вигляді текстового рядка. Функція **InputBox** має такий синтаксис:

```
Function InputBox(prompt As String, _
        Optional title As String = app_name, _
        Optional default As String = "") As String
```

app_name – назва програми, в рамках якої функціонує VBA.

Функція **InputBox** відображає на екрані модальне діалогове вікно з повідомленням prompt, одним полем введення та кнопками "OK" та "Отмена" ("Cancel"). При натисненні кнопки "OK" функція повертає текстовий рядок, введений користувачем у поле введення. При натисненні кнопки "Отмена" ("Cancel") функція повертає пустий текстовий рядок.

Аргумент prompt задає повідомлення, яке відображається у вікні. Максимальна допустима довжина повідомлення – 1024 символи. Спеціальні символи з кодами 13 та 10 можуть бути використані для розташування тексту повідомлення в декількох рядках.

Аргумент title задає заголовок діалогового вікна.

Аргумент default задає текст, який буде містити поле введення одразу після відображення вікна на екрані.

Наведена нижче програма відображає на екрані вікно із запитом до користувача "Введіть назву файлу з даними". Вікно містить поле введення, в якому відображається назва файлу, яка приймається за замовчуванням. Після натиснення кнопки "OK" назва файлу, задана користувачем у полі введення, заноситься у змінну FileName. У випадку, коли користувач натисне кнопку "Отмена" ("Cancel"), у змінну FileName буде занесено назву файлу за замовчуванням.

```
Const DefaultFileName As String = "data.txt"
Dim FileName As String

FileName = InputBox("Введіть назву файлу з даними",,_
      DefaultFileName)
If FileName = "" Then
      FileName = DefaultFileName
End If
```

Для вставлення в повідомлення prompt, яке виводиться в діалоговому вікні функціями **MsgBox** та **InputBox**, спеціальних символів табуляції та нового рядку можна використовувати стандартні константи **vbNewLine**

та **vbTab**, відповідно. Також для вставлення цих та будь-яких інших спеціальних символів можна використовувати функцію **chr**.

Наприклад:

```
Dim file_name As String
file_name = "C:\My Documents\data.txt"
MsgBox "Не вдається відкрити файл:" & vbNewLine _
& chr(34) & file_name & chr(34)
```

Наведена вище програма виводить на екран діалогове вікно з повідомленням, розташованим в двох текстових рядках. В першому текстовому рядку знаходиться фраза "Не вдається відкрити файл". В другому текстовому рядку відображається назва файлу в подвійних лапках, заданих в програмі за допомогою функції **chr**.

Про функцію **chr**:
- **Chr Function**

2.10. Деякі математичні функції

В даному розділі наведено синтаксичний опис та призначення окремих математичних функцій мови VBA. На базі цих функцій можна створити будь-яку іншу "складну" математичну функцію (наприклад, функцію, що обчислює котангенс кута), відсутню у мові VBA.

Про інші математичні функції VBA і їх,застосування:
- **Math Functions**
- **Derived Math Functions**

Function Cos(number **As Double**) **As Double**

Функція **Cos** повертає значення косинуса кута, значення якого в радіанах задається аргументом number.

Function Sin(number **As Double**) **As Double**

Функція **Sin** повертає значення синуса кута, значення якого в радіанах задається аргументом number.

Function Tan(number **As Double**) **As Double**

Функція **Tan** повертає значення тангенса кута, значення якого в радіанах

задається аргументом number.

Function Atn(number **As Double**) **As Double**

Функція **Atn** повертає значення арктангенса свого аргументу в радіанах в діапазоні від $-\pi/2$ до $+\pi/2$.

Function Abs(number **As** *type*) **As** *type*

type – числовий тип даних.

Функція **Abs** повертає значення свого аргументу, взяте за модулем. Тип значення, що повертається, такий самий, як і тип аргументу функції.

Function Sqr(number **As Double**) **As Double**

Функція **Sqr** повертає значення квадратного кореня числа number. Якщо значення аргументу number від'ємне, виникає помилка.

Function Exp(number **As Double**) **As Double**

Функція **Exp** повертає значення числа *e*, піднесеного до степені number.

Function Log(number **As Double**) **As Double**

Функція **Log** повертає значення натурального логарифму числа number.

Function Fix(number **As** *type*) **As Integer**

type – числовий тип даних.

Функція **Fix** відкидає дробову частину числа number та повертає його цілу частину. Тип значення, що повертається, такий самий, як і тип аргументу функції.

MsgBox Fix(3.4) : **MsgBox Fix**(3.7) ' виводиться 3
MsgBox Fix(-3.4) : **MsgBox Fix**(-3.7) ' виводиться -3

Function Int(number **As** *type*) **As Integer**

type – числовий тип даних.

Функція **Int** повертає цілу частину числа number. Для від'ємних значень number повертає найближче від'ємне ціле число, яке менше або дорівнює значенню number.

```
MsgBox Int(3.4) : MsgBox Int(3.7) ' виводиться 3
MsgBox Int(-3.4) : MsgBox Int(-3.7) ' виводиться -4
```

Таку функцію можна використовувати для визначення арксинуса довільного кута X:

```
Function ArcSin(X As Double)
        ArcSin = Atn(X / Sqr(-X * X + 1))
End Sub
```

Таку функцію можна використовувати для визначення логарифма числа X за довільною основою N:

```
Function LogN(X As Double, N As Double)
        LogN = Log(X) / Log(N)
End Sub
```

2.11. Деякі функції для роботи з текстовими рядками

В даному розділі наведено синтаксичний опис та призначення функцій, які дозволяють оперувати текстовими рядками.

```
Function Len(string As String) As Long
```

Функція **Len** повертає кількість символів, що містяться в текстовому рядку, заданому аргументом string.

```
Function LTrim(string As String) As String
Function RTrim(string As String) As String
Function Trim(string As String) As String
```

Функція **Trim** повертає текстовий рядок, заданий аргументом string, в якому видалено *усі початкові та прикінцеві* символи пропуску. Функція **LTrim** повертає текстовий рядок, заданий аргументом string, в якому видалено *усі початкові* символи пропуску. Функція **RTrim** повертає текстовий

рядок, заданий аргументом string, в якому видалено *усі прикінцеві* символи пропуску.

```
MsgBox Trim(" какаду ")  ' результат "какаду"
MsgBox LTrim(" какаду ") ' результат "какаду "
MsgBox RTrim(" какаду ") ' результат " какаду"
```

```
Function Left(string As String, length As Long) As String
Function Right(string As String, length As Long) As String
```

Функція **Left** повертає текстовий рядок, який містить перших length символів з текстового рядка, заданого аргументом string. Функція **Right** повертає останніх length символів з текстового рядка, заданого аргументом string. Якщо значення аргументу length від'ємне, виникає помилка. Якщо значення аргументу length перевищує кількість символів в текстовому рядку string, функції **Left** та **Right** повертають весь текстовий рядок string.

Про інші рядкові функції:
- **InStr Function**
- **Replace Function**


```
Function Mid(string As String, start As Long, _
        Optional length = 2147483647 As Long)
```

Функція **Mid** повертає текстовий рядок, який містить length символів з текстового рядка string, починаючи з символу з номером start. Якщо значення аргументу length або string від'ємне, виникає помилка. Якщо значення аргументу start перевищує кількість символів в текстовому рядку, заданому аргументом string, функція **Mid** повертає пустий текстовий рядок. Якщо значення аргументу length перевищує кількість символів в текстовому рядку string починаючи з символу з номером start, функція **Mid** повертає всі символи від заданого до кінця текстового рядка string.

```
MsgBox Left("кавун", 2)  ' результат "ка"
MsgBox Right("кавун", 2) ' результат "ун"
MsgBox Mid("кавун", 2)   ' результат "авун"
MsgBox Mid("кавун", 2, 3) ' результат "аву"
```

```
Function LCase(string As String)
Function UCase(string As String)
```

Функція **LCase** повертає текстовий рядок string, в якому всі символи перетворені до нижнього регістра (тобто до малих символів). Функція **UCase** повертає текстовий рядок string, в якому всі символи перетворені до верхнього регістра (тобто до великих символів).

2.12. Деякі функції для роботи з датою і часом

В даному розділі наведено синтаксичний опис та призначення деяких функцій, які дозволяють оперувати значеннями дати/часу (тобто значеннями типу **Date**).

Функції, призначені для визначення поточного часу та дати:

> Про арифметичні операції над датою/часом:
> - **DateAdd Function**
> - **DateDiff Function**

Function Date() As Date

Функція **Date** повертає значення типу **Date** з поточною на момент виклику функції датою та часом 00:00:00.

Function Time() As Date

Функція **Time** повертає значення типу **Date** з датою 30.12.1899 та поточним на момент виклику функції часом.

Function Now() As Date

Функція **Now** повертає значення типу **Date** з поточними на момент виклику функції датою та часом.

Наведені нижче функції призначені для виділення окремих складових дати (року, місяця, дня) із заданого значення типу **Date**:

Function Year(date **As Date**) **As Integer**
Function Month(date **As Date**) **As Integer**
Function Day(date **As Date**) **As Integer**

Значення року, місяця, дня для дати, заданої аргументом date, повертається функціями **Year**, **Month** та **Day**, відповідно.

Номер дня тижня можна визначити за допомогою функції **Weekday**:

```
Function Weekday(date As Date, _
        Optional firstdayofweek As Integer) As Integer
```

Функція **Weekday** провертає номер дня тижня для дати, заданої аргументом date (1 – неділя, 2 – понеділок, 3 – вівторок і т.д.). Аргумент firstdayofweek задає, який день тижня вважається першим (0 – вважати першим день тижня, заданий в регіональних налаштуваннях операційної системи, 1 – вважати першим неділю, 2 – вважати першим понеділок і т.д.).

Про форматування значень дати/часу:

- **Format Function**
- **WeekdayName Function**
- **MonthName Function**

Наведені нижче функції призначені для виділення окремих складових часу (години, хвилини, секунди) із заданого значення типу **Date**:

```
Function Hour(date As Date) As Integer
Function Minute(date As Date) As Integer
Function Second(date As Date) As Integer
```

Значення годин, хвилин, секунд для дати, заданої аргументом date, повертається функціями **Hour**, **Minute** та **Second**, відповідно.

Для отримання значення типу **Date** із окремих складових дати призначена функція **DateSerial**:

```
Function DateSerial(year As Integer, month As Integer, _
        day As Integer) As Date
```

Функція **DateSerial** повертає значення типу **Date**, яке отримується шляхом додавання до 1.1.year 00:00:00, кількості місяців та днів, заданих аргументами month та day. Значення аргументу year повинно знаходитися в межах від 100 до 9999 включно. Значення аргументів month та day можуть бути як додатними, так і від'ємними.

```
MsgBox DateSerial(2007, 5, 25) ' 25.05.2007 00:00:00
MsgBox DateSerial(2007, -2, 25) ' 25.10.2006 00:00:00
MsgBox DateSerial(2007, 5, 45) ' 14.06.2007 00:00:00
```

Для отримання значення типу **Date** із окремих складових часу призначена функція **TimeSerial**:

Function TimeSerial(hour **As Integer**, minute **As Integer**, _
second **As Integer**) **As Date**

Функція **TimeSerial** повертає значення типу **Date**, яке отримується шляхом додавання до 30.12.1899 hour:00:00, кількості хвилин та секунд, заданих аргументами minute та second. Значення аргументу hour повинно знаходитися в межах від 0 до 23 включно. Значення аргументів minute та second можуть бути як додатними, так і від'ємними.

```
MsgBox TimeSerial(15, 35, 47) ' 30.12.1899 15:35:47
MsgBox TimeSerial(15, -45, 47) ' 30.12.1899 14:15:47
MsgBox TimeSerial(4, -30*60, 0) ' 29.12.1899 02:00:00
```

Function IsDate(expression **As Variant**) **As Boolean**

Функція **IsDate** повертає значення **True**, якщо значення аргументу expression може бути перетвореним у значення типу **Date**. Які саме текстові рядки можуть бути перетворені до типу **Date** залежить від регіональних налаштувань операційної системи.

У наведеному нижче прикладі вказано, яке значення повертає функція **IsDate** у випадку стандартних українських регіональних налаштувань операційної системи Windows:

```
MsgBox IsDate("February 12, 2007") ' виводить False
MsgBox IsDate("Лютий 12, 2007") ' виводить True
MsgBox IsDate("12 лют. 2007") ' виводить True
MsgBox IsDate("12 лютий 2007") ' виводить True
MsgBox IsDate("12 лютого 2007") ' виводить False
MsgBox IsDate("12.02.2007") ' виводить True
MsgBox IsDate(#2/12/2007#) ' виводить True
MsgBox IsDate("Текст") ' виводить False
```

3. ОСНОВИ РОБОТИ З РЕДАКТОРОМ VBA

3.1. Інтерфейс редактора

З будь-якого програмного компонента Microsoft Office редактор програмного коду мови VBA викликається за натисканням комбінації клавіш **[Alt] + [F11]**. Вигляд вікна редактора та його складові зображено на рис 3.1.

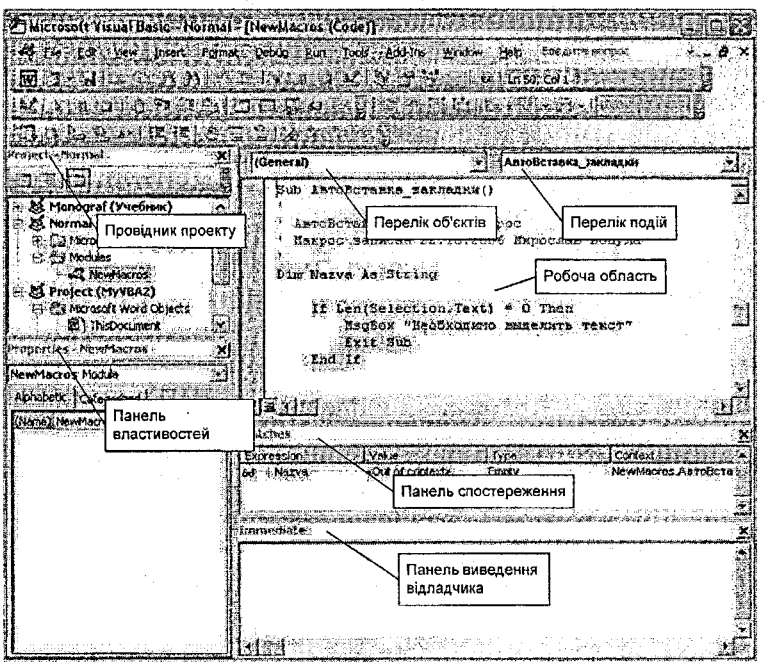


Рис. 3.1. Інтерфейс редактора VBA

В ***робочій області*** виконується набір програмного коду. ***Робоча область*** виконує функції простого текстового редактора, в якому автоматично виконується перевірка синтаксису мовних виразів, розфарбовування та форматування лексем відповідно до їх типу, виводиться "підказка" щодо можливих варіантів введення коду.

Провідник проекту подає документи (файли), до яких відноситься

програма, у вигляді дерева об'єктів. Для того, щоб почати створювати нову процедуру або функцію, необхідно створити у проекті хоча б один об'єкт – ***модуль***. Для цього слід викликати покажчиком миші на елементі проекту контекстне меню і обрати з нього команду "Insert" – "Module".

Для створеного модуля можна змінити його ім'я у ***панелі властивостей***. Для цього слід виділити покажчиком миші модуль та змінити його властивість "Name".

Перелік об'єктів при роботі з програмуванням форм містить перелік всіх об'єктів форми разом з об'єктом форми. При роботі з процедурами ***модуля*** у переліку наявний лише один пункт "General".

Перелік подій містить пункт "Declarations" (об'явлення змінних, констант та бібліотек на рівні ***модуля***) та назви всіх процедур-функцій модуля. При виборі з переліку назви процедури курсор введення буде переміщено на початок вказаної процедури. І навпаки, якщо курсор введення знаходиться всередині будь-якої процедури модуля, то у переліку подій буде виділено відповідний пункт.

Редактор дозволяє працювати з чотирма панелями інструментів, кожна з яких може бути налагоджена так само, як це виконується, наприклад, у Microsoft Word. Розглянемо ті інструменти, що будуть необхідні при розробленні програм без використання форм та об'єктів (рис. 3.2).

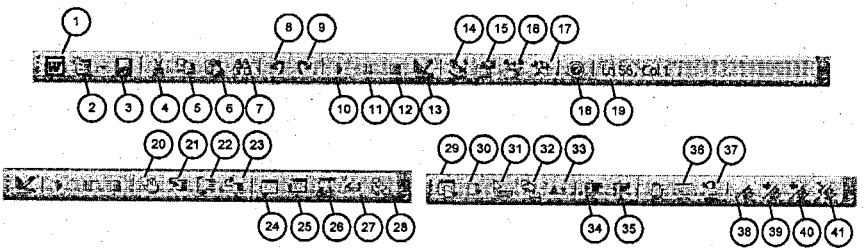


Рис. 3.2. Деякі інструменти редактора VBA

Кнопки панелі "Standard".

1 – кнопка переходу до відповідної host-програми. Якщо проект виконується в Microsoft Word, то буде виконано перехід до вікна Word, якщо в Microsoft Excel, то перехід до вікна Excel;

2 – кнопка створення нового об'єкта у дереві об'єктів провідника проекту;

3 – збереження проекту;

4, 5, 6 – інструменти вирізання, копіювання і вставлення виділеного тексту чи об'єкта;

7 – інструмент пошуку і заміни тексту;

8, 9 – інструменти відміни та повторення дії користувача;

10, 11, 12 – кнопки компіляції і пуску на виконання, паузи виконання та припинення виконання проекту;

13 – кнопка переключення host-програми в режим дизайну, в якому дозволяється використовувати елементи форм безпосередньо в документі host-програми;

14 – кнопка відкриття панелі провідника проектів "Project";

15 – кнопка відкриття панелі властивостей;

16 – кнопка відкриття браузера об'єктів;

17 – кнопка виклику панелі елементів форм;

18 – кнопка виклику панелі довідки;

19 – область виведення позиції курсора – вказується номер рядка програми (Ln) і символу (Col), де знаходиться курсор;

Кнопки панелі "Debug":

20 – кнопка встановлення точки останову;

21, 22, 23 – кнопки керування покроковим виконанням програми;

24 – відкриває панель локальних змінних, яка дозволяє слідкувати за значеннями всіх змінних, що використовуються в поточній програмі;

25 – відкриває вікно панелі виведення налагоджувача коду, в яке можна виводити текст, з метою відслідковування роботи програми під час її налагодження;

26 – відкриває вікно панелі спостереження змісту заданих змінних, в якому можна переглянути структуру і зміст змінних, які задані для спостереження;

27 – дозволяє додати виділену змінну в список змінних панелі спостереження;

28 – відкриває вікно стека викликів, що дозволяє прослідкувати за порядком виклику на виконання процедур та функцій;

Кнопки панелі "Edit":

29 – відкриває поряд з поточною лексемою або змінною список можливих значень, що дозволяє швидко набирати програмний код. Дублюється комбінацією клавіш **[Ctrl] + [J]**;

30 – відкриває поряд з поточною лексемою або змінною список можливих констант-прапорців, які можуть бути застосовані для поточної фун-

кції. Дублюється комбінацією клавіш **[Ctrl] + [Shift] + [J]**;

31 – відкриває поряд з поточною лексемою або змінною підказку із синтаксису поточної функції або оператора. Дублюється комбінацією клавіш **[Ctrl] + [I]**;

32 – відкриває поряд з поточною лексемою або змінною підказку із синтаксису параметрів, які можуть бути застосовані для поточної функції. Дублюється комбінацією клавіш **[Ctrl] + [Shift] + [I]**;

33 – відкриває поряд з поточною лексемою список можливих завершень лексеми відповідно до існуючих операторів, функцій, змінних. Дублюється комбінацією клавіш **[Ctrl] + [Space]**;

34, 35 – дозволяють збільшити або зменшити відступ рядка від лівої границі текста програми. Дублюється комбінацією клавіш **[Tab]** та **[Tab] + [Shift]**;

36, 37 – дозволяють закоментувати та зняти коментування з виділених рядків програми;

38, 39, 40, 41 – дозволяють оперувати закладками в програмному коді, а саме: встановлювати закладку, перейти до наступної, перейти до попередньої закладки, зняти поточну закладку.

3.2. Застосування закладок і лінії розбиття

Іноді в процесі створення програмного коду в одному місці виникає потреба внести зміни до іншої частини коду. Треба перейти до іншого місця, але після цього слід розшукати той рядок, де була перервана робота.

В цьому випадку зручно використовувати закладки. Закладка - це мітка, якою можна помітити рядок тексту програми, натиснувши кнопку 38  (див. рис. 3.2). Для того, щоб видалити закладку, потрібно встановити покажчик введення на рядок, що помічений нею, і натиснути кнопку 41  . Кнопки 39 та 40 дозволяють пересуватись між закладками в програмі.

Також зручною є можливість розділення вікна редагування на дві частини - для одночасного перегляду різних частин програми. Це виконується за допомогою лінії розбиття - маленького бігунка  відразу над смугою вертикальної прокрутки. Достатньо пересунути покажчиком миші цей бігунок на необхідну відстань для вертикального розбиття робочої області на дві зони.

3.3. Поради з роботи в редакторі коду

Для автоматичного завершення набору назв змінних, функцій та процедур достатньо почати вводити назву і натиснути кнопку 33 або комбінацію клавіш **[Ctrl] + [Space]**.

Якщо надрукувати один рядок коду з відступом, то такий самий відступ буде встановлений і для наступних рядків. Змінити поведінку можна за допомогою параметра Auto Indent в діалоговому вікні Options.

Якщо редактор коду розпізнає ключове слово, він автоматично робить його першу букву великою і виділяє синім кольором.

Для того, щоб закоментувати або розкоментувати декілька рядків відразу, можна включити відображення панелі інструментів Edit і скористатися кнопками 36  Comment Block і 37  Uncomment Block.

Якщо при створенні процедури написати ключове слово **Sub** або **Function**, то редактор автоматично дописує оператори **End Sub** або **End Function**. Ця дія буде виконана при зміненні ключового слова **Sub** на **Function** і навпаки.

За замовчуванням при переході на новий рядок редактор коду виявить синтаксичну помилку, то буде видано попередження. Таку поведінку редактора можна відключити, знявши прапорець Auto Syntax Check в діалоговому вікні Options. Роботі це не перешкодить, тому що синтаксично неправильний текст у будь-якому випадку автоматично виділятиметься червоним кольором.

В редакторі Visual Basic цілком допускається робота відразу з декількома вікнами редагування коду. Перехід між ними виконується за натисканням комбінацій клавіш **[Ctrl] + [Tab]** або **[Ctrl] + [F6]**.

За замовчуванням редактор коду працює в режимі Full Module View – показ всього вмісту модуля. Якщо потрібно переглядати процедури окремо, можна перейти в режим Procedure View. Кнопки для перемикання режимів  розташовані в лівому нижньому кутку вікна редактора коду.

Для того, щоб отримати довідку з будь-якого ключового слова VBA достатньо виділити це слово і натиснути клавішу **[F1]**.

3.4. Покрокове виконання програми та інструменти налагодження програмного коду

Для того, щоб виконати поточну процедуру модуля, слід натиснути кнопку 10 або клавішу **[F5]**. Програма буде відкомпільована та запущена на виконання. Щоб перервати виконання програми, наприклад, у випадку зациклювання, можна натиснути комбінацію клавіш **[Ctrl] + [Break]**.

Для пошуку помилок у програмі і її налагодження редактор VBA має багато засобів. Як правило пошук помилки виконується у покроковому режимі виконання програми. Для виконання процедури в цьому режимі слід процедура має бути запущена кнопкою 21 або клавішею **[F8]**.

При цьому операції програми будуть виконуватись послідовно з паузою після кожного виконання. За повторним натисканням кнопки 21 або клавіші **[F8]** буде виконуватись наступна операція. Поточна операція виділяється в редакторі жовтим кольором.

Якщо в операції програми використано виклик іншої процедури/функції, то покрокове виконання перейде до неї. Покрокове виконання програми буде відбуватись всередині цієї "підлеглої" процедури і після її завершення повернеться до наступної операції "головної" програми.

Щоб уникнути перенесення покрокового виконання у "підлеглу" процедуру, одразу виконати її та перейти до наступної операції "головної" програми слід користатись кнопкою 22 або комбінацією клавіш **[Shift] + [F8]**.

Для того, щоб перейти від режиму повного виконання програми (за кнопкою 10 або клавішею **[F5]**) до покрокового є два способи:

- встановити у потрібному рядку коду точку зупинення виконання програми кнопкою 20 або клавішею **[F9]**;
- встановити у окремому рядку перед потрібним рядком коду оператор **Stop**.

Для спостереження за значеннями змінних слід використовувати можливості панелі стеку "Locals" або панелі спостереження "Watches". Щоб додати змінну до елементів панелі спостереження, слід виділити назву змінної і натиснути кнопку 27 або комбінацію клавіш **[Shift] + [F9]**. Вигляд панелі спостереження і перегляд стану змінної bb показано на рис. 3.3.

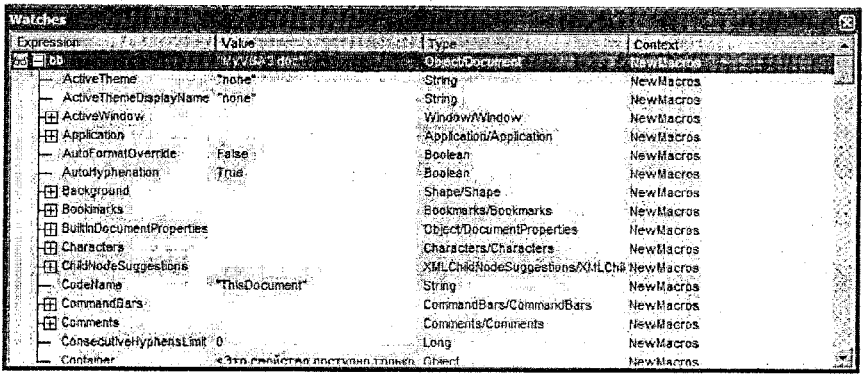


Рис. 3.3. Панель спостереження "Watches"

Також для спостереження за змінними і службового виведення тексту зручно використовувати можливості об'єкта **Debug**. Для того, щоб подивитись на вміст змінної, використовується метод **Debug.Print**. Результат буде виведено у панель "Immediate" виведення налагоджувача у вигляді текстового рядка (див рис. 3.4). Панель "Immediate" відкривається натисканням комбінації клавіш **[Ctrl] + [G]**.

```
Dim x As Integer, st As String
    x = 10 : st = "текст"
    Debug.Print "Значення X = " & x ; " st = " ; st
```

Рис. 3.4. Панель спостереження "Watches"

ДОДАТОК 1

ТАБЛИЦЯ ВІДПОВІДНОСТІ ДЕЯКИХ УКРАЇНСЬКИХ ТЕРМІНІВ АНГЛІЙСЬКИМ

Український термін	Англійський термін
оператор	statement
вираз	expression
операція	operator
ключове слово	keyword
функція	Function procedure
процедура	Sub procedure
host-програма	host-application
лексема	token
автоматичне перетворення типів даних	implicit data types conversion
секція описів	declarations section
іменовані аргументи	named arguments
верхній регістр	uppercase
нижній регістр	lowercase
чутливість до регістру	case sensitivity
модуль	module
проект	project
беззнакове число	unsigned number
умовний оператор	conditional statement
оператор циклу	iteration statement
оператор переходу	jump statement
літеральна константа дати	date literal

ДОДАТОК 2

WEB-РЕСУРСИ З ПРОГРАМУВАННЯ МОВОЮ VBA

MSDN Home Page (Россия)
(http://msdn2.microsoft.com/ru-ru/default.aspx)

Microsoft Developers Network – база знань з програмування від Microsoft. Першоджерело знань з мови програмування VBA.

VBNET.RU (http://vbnet.ru)

Один з найпопулярніших сайтів. Містить розділи: Новини; Приклади; Бібліотека кодів; НеЧаВо; VBA; Довідник з VB; Довідник з WinAPI; елементи керування ActiveX; Статті з VB; Майстерня розробника; Форум VBnet; Голосування; Корисні програми; Магазин VBnet; Розсилка; Конкурси; Чат; Посилання; Бета-тестування.

VB.KIEV.UA (http://www.vb.kiev.ua)

Потужна і постійно оновлювана база знань, яка містить приклади кодів, довідники, поради за розділами: API; ActiveX; Compression; COM; Використання елементів керування; Криптографія; Бази даних; Enterprise; Розробка ігор; Графіка; Інтерфейс юзера; Math; Мультимедіа; Мережі; ООП; Registry; Ресурси (.res); Друк; Оболонка; Скріптинг; Рядки; XML; Xtras; Основи мови і алгоритми.

Исходники.RU (http://sources.ru/vb/index.html)

Розглядаються питання системного і мережевого програмування, програмування графіки, алгоритмів і математики, вікон, діалогів і форм, створення ігор, призначеного для користувача інтерфейсу, баз даних

MSDN: Питання і відповіді по роботі з VBA
(http://microsoft.com/rus/msdn/activ/msvb/archive/vba/default.mspx)

Тематична категоризація публікацій Андрія Колесова і Ольги Павлової («Програмування в середовищі Microsoft Visual Basic - це цілком доступно»). Містить питання і відповіді щодо роботи в середовищі VBA.

CodeNet.Ru (http://www.codenet.ru/cat/Languages/Visual-Basic/)

Потужний і популярний сайт з питань програмування. В розділі "Языки программирования – Visual Basic" містить підрозділи: API; VB-Script; VBA; Бази даних; Графіка і мультимедіа; Інтернет і мережа; Класи і

об'єкти; Елементи керування і форми; Математик; Обробка помилок; Загальні питання; Оптимізація; Друк; Програмні додатки і розповсюдження програм; Робота з файлами і каталогами; Реєстр; Інструкції.

VBStreets - улицы Visual Basic (http://www.vbstreets.ru)

Visual Basic Streets - все про Visual Basic: бібліотека ActiveX, FAQ, статті, конференції, свіжі новини, програмні ресурси, приклади і багато корисного в допомогу VB-програмістам.

VBCoder (http://vbcoder.narod.ru)

Сайт про програмування мовою Visual Basic. Все необхідне (статті, приклади, програми, ActiveX-елементи і т.д.) для створення повноцінного програмного продукту в даному середовищі розробки.

Visual Basic на русском (http://www.vbrussian.com)

Найстаріший сайт про VB. На сайті потужний форум, на якому можна отримати відповідь практично на будь-яке питання за декілька годин. Містить статті та огляди, присвячені VB, поради з оптимізації коду, прийоми програмування, FAQ. Посилання на російськомовні сайти VB. Програми для VB-програмістів. Огляди книжок. Самовчитель із створення елементів керування. Двосторонній гейт між Фідо і Інтернетом.

VbMania (http://www.vbmania.ru)

Електронний журнал. Містить різні аспекти програмування мовою Visual Basic (VB). Є цікавий розділ "Моя манія", в котрий кожен може додати власні новини.

Visual Basic в примерах (http://hotmix.narod.ru)

Назва цього сайту говорить само за себе, тут можна зустріти що завгодно - цікаві книги, корисні програмки, скрінсейвери, програмні коди цих програм.

AlgoList (http://algolist.manual.ru)

Початкові коди і книги про алгоритми. Найширша тематика.

Alglib.ru (http://alglib.sources.ru)

Бібліотека алгоритмів з початковими програмними кодами.

ЛІТЕРАТУРА

1. ISO/IEC 14977:1996(E) Information technology – Syntactic meta-language – Extended BNF.

2. Розділ "Visual Basic Language Reference" файлу електронної допомоги "Microsoft Visual Basic Documentation", що поставляється з host-програмами мови програмування VBA.

3. Андрей Гарнаев. VBA. Наиболее полное руководство. – С-Пб. : БХВ-Петербург, 2005 г. – 848 с.

4. Кузьменко В. Г. VBA 2003: Самоучитель. – М.: БИНОМ, 2004. – 429с.

5. Александрова В. М. VBA в прикладах та задачах: Навч. посіб. для студ. екон. спец. вищ. навч. закл. / Київський національний торговельно-економічний ун-т. – К. : Книга, 2004. – 368с.

6. Делявський М. В., Жмуркевич А. Є., Одрехівський М. В., Чаповська Р. Б. Основи алгоритмізації та програмування: середовище VBA: Навч. посібник для студ. екон. спец. вищих навч. закл. – Чернівці: Книги-XXI, 2006. – 430с.

7. Каленюк П. І., Обшта А. Ф., Гоблик Н. М., Клочко Н. Ф., Ментинський С. М. Практикум з програмування на VBA: Навч. посіб. для студ. екон. спец. – Л.: Видавництво Національного університету "Львівська політехніка", 2005. – 208с.

8. Биллинг Владимир Арнольдович. Средства разработки VBA-программиста. – М.: Русская Редакция, 2001. – 462с.

Навчальне видання

Андрій Вячеславович Камінський
Мирослав Павлович Боцула

Практичний посібник з основ мови програмування VBA

Навчальний посібник

Оригінал-макет підготовлено Боцулою М. П.

Редактор Старічек Т. О.

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК №746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 26.05.2008 р.
Формат 29, 7×42¼
Друк різографічний
Тираж 100 прим.
Зам. № 2008-068

Гарнітура Times New Roman
Папір офсетний
Ум. друк. арк. 4.9

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ