ФАКУЛЬТЕТ АВТОМАТИКИ, ТЕЛЕМЕХАНІКИ ТА ЗВ'ЯЗКУ

Кафедра обчислювальної техніки і систем управління

ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ VISUAL BASIC 6.0

КОНСПЕКТ ЛЕКЦІЙ

з дисциплін

«ОБЧИСЛЮВАЛЬНА ТЕХНІКА, ПРОГРАМУВАННЯ, МОДЕЛЮВАННЯ СИСТЕМ», «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ»

Частина 2

Харків – 2014

Програмування в середовищі Visual Basic 6.0: Конспект лекцій / В.С. Меркулов, І.Г. Бізюк, О.В. Чаленко. – Харків: УкрДАЗТ, 2014. – Ч. 2. – 65 с.

Конспект лекцій розроблено відповідно до програм з дисциплін «Обчислювальна техніка, програмування, моделювання систем» та «Обчислювальна техніка та програмування».

Метою лекцій є опанування студентами алгоритмічної мови високого рівня, здобуття необхідних знань для організації обчислень в середовищі Visual Basic 6.0, використання теоретичних засад об'єктно-орієнтованого підходу до розробки програмних проектів.

Рекомендується для студентів спеціальностей 6.100501 та 6.092200 всіх форм навчання.

Іл. 31, табл. 13.

Конспект лекцій розглянуто та рекомендовано до друку на засіданні кафедри обчислювальної техніки та систем управління 27 лютого 2012 р., протокол № 7.

Рецензент

проф. Г.І. Загарій

ПРОГРАМУВАННЯ В СЕРЕДОВИЩІ VISUAL BASIC 6.0

КОНСПЕКТ ЛЕКЦІЙ

з дисциплін

«ОБЧИСЛЮВАЛЬНА ТЕХНІКА, ПРОГРАМУВАННЯ, МОДЕЛЮВАННЯ СИСТЕМ», «ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ»

Частина 2

Відповідальний за випуск Бізюк І.Г.

Редактор Решетилова В.В.

Підписано до друку 22.03.12 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 2,5. Тираж 100. Замовлення №

Видавець та виготовлювач Українська державна академія залізничного транспорту, 61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб'єкта видавничої справи ДК № 2874 від 12.06.2007 р.

ЗМІСТ

4 ПРОГРАМНІ МОДУЛІ

4.1 Оформлення програмних кодів

При написанні програмних кодів необхідно подбати про те, щоб програма була читабельною, тобто із достатньою кількістю коментарів, а оператори великої довжини були розміщені на декількох рядках. Про засоби, що дозволяють цього досягти, піде мова в даному розділі.

4.1.1 Коментарі

Крім команд і виразів, ви можете включити у свої методи й процедури будь-який довільний текст — коментарі, що пояснюють код програми, роблять її більш читабельною і допоможуть краще орієнтуватися в програмі.

Для включення в текст програми коментаря необхідно ввести символ ('), що може бути першим символом рядку або перебувати в будь-якому її місці. Цей символ означає початок коментаря. Будь-який текст, розташований у рядку слідом за цим символом, буде сприйматися як коментар, тобто VB не буде транслювати цей текст.

Приклад:

' коментар, що починається з початку рядка

Print strName ' коментар, що міститься за оператором

4.1.2 Розміщення оператора на декількох рядках

У тому випадку, коли оператор має велику довжину, його можна розбити на кілька рядків, використовуючи символи продовження рядка: пробіл, за яким треба поставити символ підкреслення (_).

Приклад: *помістимо на двох рядках оператор, що поєднує прізвище, ім'я та по батькові:*

strName = strLastname & strFirstname & strSecondname

Одержимо таке:

strName = strLastname _

& strFirstname & strSecondname

4.1.3 Розміщення декількох операторів на одному рядку

Як правило, при написанні програм оператори розміщають на окремому рядку. Якщо оператори мають невелику довжину, VB дозволяє їх помістити на одному рядку, розділивши двокрапкою.
Приклад:

strLastname = "Іванов ": strFirstname = "Іван"

4.2 Види програмних модулів

Програми VB зберігаються в програмних модулях, які бувають трьох видів:

- ***модуль форми;***
- ***стандартний модуль;***
- ***модуль класу.***

Простий додаток, що складається з однієї форми, містить, як правило, тільки модуль форми. В міру ускладнення додатка повторювані функції, виконувані з декількох модулів форми, можна виділити в окремий програмний код, що буде загальним для всіх. Такий програмний код називається стандартним модулем. При використанні в VB об’єктно-орієнтованого програмування створюються модулі класів.

Модулі форми *можуть містити оголошення змінних, констант, типів даних, зовнішніх процедур, використовуваних на рівні модуля, процедур обробки подій. Вони зберігаються у файлах, що мають розширення* ***frm****. У модулі можна також посилатися на інші форми й об'єкти даного додатка.*

Стандартні модулі *можуть містити оголошення глобальних і локальних змінних, констант, типів, зовнішніх процедур і процедур загального характеру, доступних для інших модулів даного додатка. Вони зберігаються у файлах з розширенням* ***bas****.*

Використовуючи об’єктно-орієнтоване програмування, в VB можна створювати нові об'єкти з розробленими для них властивостями й методами, розміщуючи їх у ***модулі класів****, що зберігаються у файлах з розширенням* ***cls****.*

4.2.1 Структура проекту

Проект звичайно включає у свій склад один або кілька контейнерів (рисунок 4.1). Контейнером може бути форма або модуль.

Форма може містити об'єкти (написи, кнопки й т.д.) і програмний код (рисунок 4.2). При виконанні додатка вона відображається на екрані монітора у вигляді вікна. Модуль відрізняється від форми тим, що він може містити програмний код, але не може містити об'єктів і ніяк не відображається при виконанні проекту.

З контейнером звичайно пов'язаний програмний код (рисунки 4.2, 4.4). Початкова (верхня) частина програмного коду контейнера, розташована до оголошення першої процедури або

Рисунок 4.1 — Структура проекту

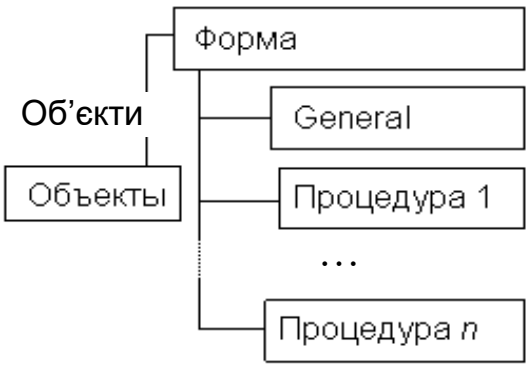

Рисунок 4.2 — Структура форми

функції, називається головною секцією (**General**). В головній секції можуть бути тільки інструкції оголошень і не можуть бути виконувані інструкції (наприклад, інструкції присвоєння).

…

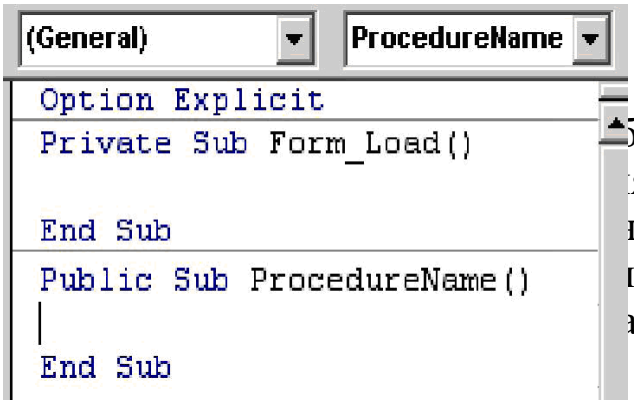

Рисунок 4.3 — Вікно програми з процедурою з ім'ям ProcedureName

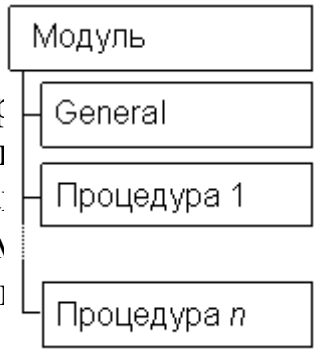

Рисунок 4.4— Структура програмного коду модуля

4.3 Редагування вихідних кодів

Для створення програмних кодів в VB використовується редактор коду. Щоб його запустити, необхідно у вікні **Project Explorer** установити курсор на форму або на модуль, для якого створюється код, і виконати одну з таких дій:

- у меню **View** вибрати команду **Code**;
- з контекстного меню модуля вибрати команду **View Code**.

При виконанні кожної із цих дій відкривається вікно редагування (рисунок 4.5), у яке можна вводити текст програми.

Для кожного модуля в VB створюється окреме вікно коду, розділене усередині на секції. У верхній частині вікна **Project** розташовані два списки, що розкриваються: **Object** (Об'єкт) і **Procedure** (Процедура). Лівий список **Object** дозволяє здійснювати вибір секції та перехід між секціями. Для стандартного модуля список **Object** містить загальну секцію (**General**).

У модулі класу цей список містить загальну секцію й секцію класу.

У формі список **Object** містить загальну секцію, секцію для форми (**Form**), а також секції для всіх розміщених у формі об'єктів.

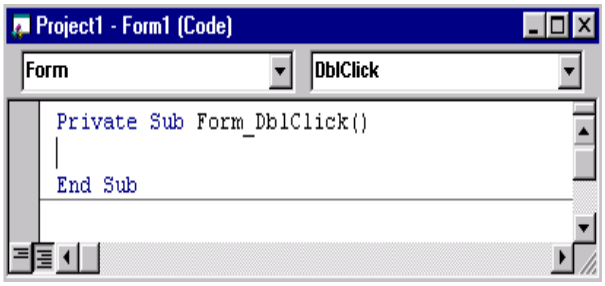

Рисунок 4.5 — Вікно редагування вихідного кода

Для кожної секції, обраної зі списку **Object**, можна створити процедуру, доступ до якої здійснюється за допомогою списку **Procedure** вікна редактора коду, що містить події. При виборі зі списку **Object** загальної секції модуля (**General**) список **Procedure** містить тільки одне значення **Declarations** (Оголошення), що дозволяє оголосити локальні змінні, константи й бібліотеки DLL.

Написання програмних кодів в VB полегшується тим, що редактор автоматично пропонує розроблювачеві в міру необхідності список операторів, функцій, властивостей об'єктів. Наприклад, при введенні імені елемента управління форми на екрані з'являється список властивостей даного об'єкта (рисунок 4.6). Вам досить двічі клацнути мишею в списку на потрібній властивості, і вона буде перенесена в програмний код.

Редактор VB також надає користувачеві підказки із синтаксисом при написанні операторів і функцій. Вони з'являються на екрані під курсором при введенні найменування оператора або функції (рисунок 4.7).

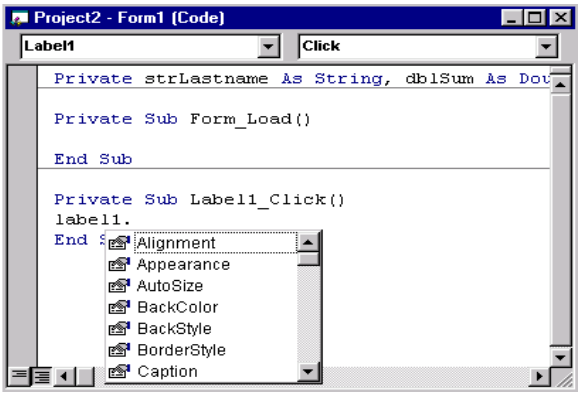

Рисунок 4.6 — Використання списку властивостей об'єкта для напису коду

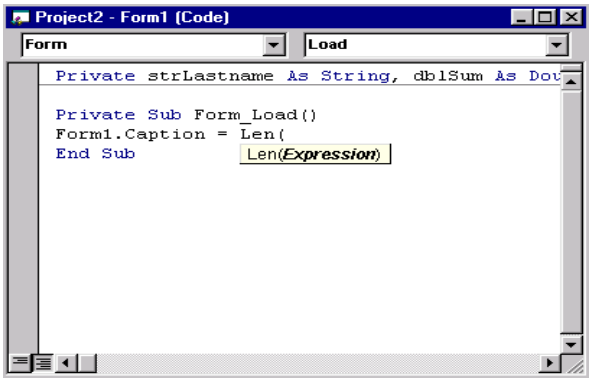

Рисунок 4.7 — Використання підказок синтаксису операторів і функцій

Зауваження: *якщо при написанні коду не з'являється список із властивостями об'єктів і підказки із синтаксисом операторів і функцій, то в меню Tools (Сервіс) виберіть команду Options (Параметри) і на вкладці Editor (Редактор) діалогового вікна Options встановіть прапорці Auto List Members (Члени автоматичного списку) і Auto Quick Info (Швидка автоматична інформація). Для підключення цих можливостей редактора можна також використовувати комбінації клавіш <Ctrl>+<J> та <Ctrl>+<I>, відповідно.*

4.4 Процедури

При програмуванні широко використовуються процедури, що дозволяють розбивати програмні коди на невеликі логічні блоки, які, по-перше, легше налагоджувати, а по-друге, можна у свою чергу використовувати при створенні інших процедур.

В VB існують такі види процедур:

- **Sub**
- **Function**
- **Property**

4.4.1 Процедури Sub

Процедура **Sub** не повертає значення й найбільш часто використовується для обробки пов'язаної з нею події. Її можна поміщати в стандартні модулі, модулі класів і форм. Вона має такий синтаксис:

[**Private**] [**Public**] [**Static**] **Sub** *Ім'яПроцедури* (*аргументи*)
операторы
End Sub

Між ключовими словами Sub і End Sub у процедурі розташовуються виконувані при її виклику оператори програмного коду. Параметр *аргументи* можна застосовувати для оголошення переданих у процедуру змінних.

Процедури Sub підрозділяються на ***загальні процедури*** й ***процедури обробки подій.***

Загальні процедури служать для розміщення повторюваних операторів, використовуваних процедурами з обробки події, тим самим розвантажуючи їх, і крім того для дублювання кодів, які часто зустрічаються, що у свою чергу полегшує підтримку додатка.

Процедури обробки подій пов'язані з об'єктами, розміщеними у формах VB, або із самою формою й виконуються при настанні події, з якою вони пов'язані.

Для події, пов'язаної з формою, процедура **Sub** має такий синтаксис:

Private Sub Form_*Ім'яПодії* (*аргументи*)
операторы
End Sub

Як видно із синтаксису, найменування процедури обробки події для форми містить слово **Form**, потім символ підкреслення (_) і ім'я події.

Приклад: *ім'я процедури, виконуваної при завантаженні форми, буде Form_Load, а процедури, виконуваної при клацанні миші на формі — Form_click. При формуванні процедури для форми MDI її ім'я буде містити перед словом Form приставку MDI, тобто записуватися MDiForm.*

Для події, пов'язаної з елементом управління форми, процедура обробки подій **Sub** має такий синтаксис:

Private Sub *ім'я елемента управління_ім'я події* (аргументи)
операторы
End Sub

Найменування процедури обробки події для елемента управління форми містить *ім'я елемента управління*, задане у властивості **Name**, потім символ підкреслення (_) і *ім'я події*.

Приклад: *ім'я процедури, виконуваної при клацанні миші на кнопці управління, що має найменування cmdPrint, буде Form_Click*.

VB полегшує формування імен створюваних процедур. Розроблювачеві необхідно виконати для цього такі дії:

- у вікні Properties за допомогою властивості Name (Ім'я) задати ім'я об'єкта, для якого створюється процедура. Якщо ім'я не буде задано, то при створенні процедури VB використовує ім'я, що присвоюється об'єкту за замовчуванням при його розміщенні у формі. При такій зміні найменування об'єкта необхідно буде змінити й ім'я процедури;
- у вікні редактора коду зі списку Object (Об'єкт) вибрати об'єкт, для якого створюється процедура;
- зі списку Procedure (Процедура) вибрати подію, обробка якої буде виконуватися.

Після виконання зазначених дій в області розміщення процедур редактора коду будуть розміщені оператори Sub і End Sub із вказівкою найменування процедури. Необхідно розмістити між цими операторами виконуваний при настанні цієї події програмний код.

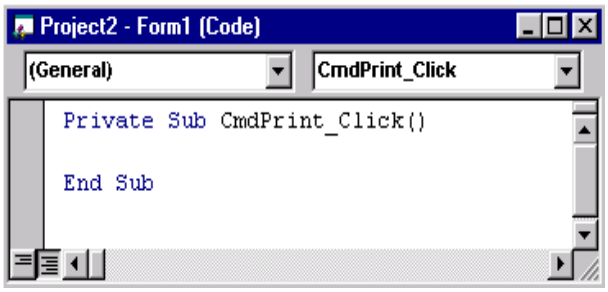

Рисунок 4.8 —.Visual Basic формує найменування процедури

У випадку, якщо після створення процедури ім'я елемента управління буде змінено, а ім'я пов'язаної з ним процедури залишиться колишнім, процедура стане загальною.

4.4.2 Процедури Function

Процедури **Function,** на відміну від процедур **Sub**, можуть повертати значення в процедуру, що її викликає. Синтаксис

[**Private**] [**Public**] [**Static**} **Function** *Ім'я Процедури* (*аргументи)* [As type]
операторі
End Function

Процедури Function, як і змінні, мають тип, що задається за допомогою ключового слова **As**. Якщо тип процедури не заданий, за замовчуванням їй присвоюється тип **Variant**. Тип процедури визначає в свою чергу тип значення, що повертається нею.

Значення, що повертається процедурою, присвоюється *ІменіПроцедури*[1].

Приклад: розглянемо процедуру, що обчислює площу квадрата:

```
Function Square (int As Integer) As Integer
Square = int ^ 2
End Function
```

Для виклику цієї процедури у формі можна використовувати, наприклад, код:

```
TxtSquare.Text = Square(TxtSide.Text)
```

Виклик процедур. Процедура **Sub** не повертає значення, однак вона може змінювати значення змінних, заданих для неї як параметри. Виклик процедури **Sub** можна здійснювати двома способами.

Перший спосіб припускає використання ключового слова **Call**.

Приклад: процедуру з ім'ям ***NameProc*** можна викликати оператором

Call *NameProc* (*аргумент1, аргумент2, ...аргумент N*)

Другий спосіб дозволяє викликати процедуру **Sub,** вказавши на її ім'я.

Приклад:

NameProc аргумент1, аргумент2, ...аргумент N

При виклику процедури модуля форми з іншого модуля необхідно вказувати посилання на ім'я модуля форми, що містить процедуру.

Приклад: для виклику процедури з ім'ям ***NameProc***, що перебуває в модулі форми ***Form1***, оператор повинен виглядати так:

Call *Form1.NameProc* (*аргумент1,аргумент2 ...аргумент N*)

Виклик процедури **Function** аналогічний виклику вбудованих функцій VB. Крім цього, процедуру **Function** можна викликати так само, як процедуру **Sub**:

```
Call Square(Side)
Square Side
```

У цьому випадку VB ігнорує значення, що повертається функцією.

Передача параметрів. Змінні, передані процедурі, називають ***параметрами процедури***.

Під ***формальними*** слід розуміти параметри, які вказуються в ***описі*** процедури. Вони є локальними змінними й з їхньою допомогою можна міняти значення змінних у процедурі.

Під ***фактичними*** слід розуміти параметри, які вказуються у ***виклику*** процедури. Вони є глобальними змінними, тобто тими реальними значеннями, з якими будуть зроблені обчислення.

---

[1] ***Ім'яПроцедури*** **може бути використане у виразах програмного коду аналогічно стандартним функціям VB.**

У розділі «параметри» в описі процедур через кому записуються формальні параметри.

При виклику процедур параметри, зазначені в дужках, називаються фактичними й передаються в процедуру для одержання значення, що обчислюється і повертається в основну програму. Як фактичні параметри можуть використовуватися масиви, арифметичні вирази, змінні й константи.

Між фактичними й формальними параметрами повинна бути відповідність за кількістю, типом даних і послідовністю розташування.

У список аргументів процедури рекомендується включати всі вхідні й всі вихідні для цієї процедури дані.

У список аргументів функції рекомендується включати всі вхідні для цієї функції дані. Її єдиний результат повертається в програму через ім'я функції.

VB дозволяє задавати тип параметрів за допомогою ключового слова **As**:

Function Square (int As Integer) As Integer

Передача параметрів у процедуру може здійснюватися двома способами: ***за значенням*** (by value) і ***за посиланням*** (by reference).

У першому випадку в процедуру як змінна передається не сама змінна, а її копія. Тому зміна параметра в процедурі зачіпає не змінну, а її копію.

Якщо аргументом виявиться зовнішня щодо процедури змінна, то ніякі маніпуляції із цим аргументом у тілі процедури або функції не змінять значення цієї зовнішньої змінної. Такий спосіб застосовується для вхідних даних. Ключове слово **ByVal** вказувати обов'язково.

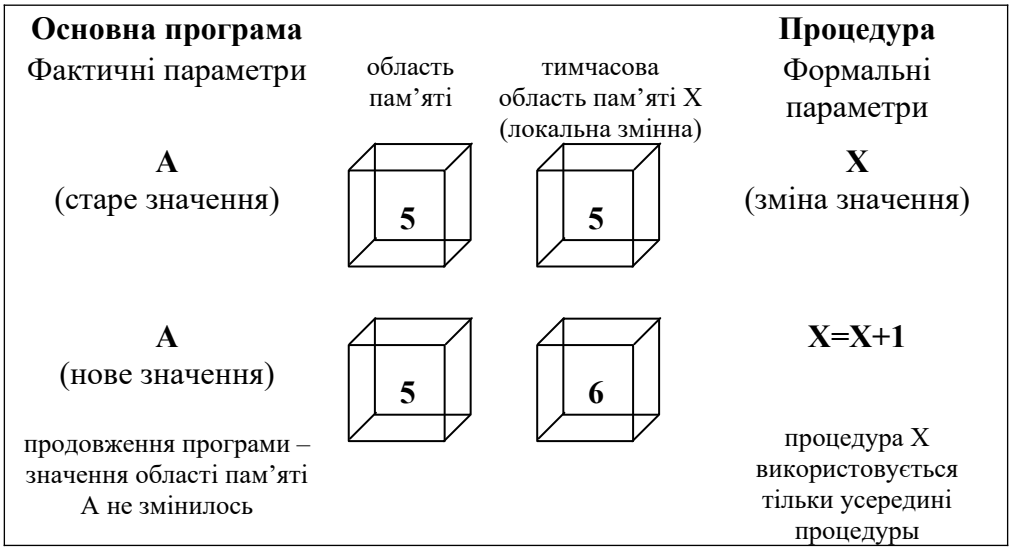

Рисунок 4.9 — Передача параметра за значенням

Для передачі в процедуру параметрів за посиланням використовується ключове слово **By Ref**.

Приклад:
Sub NameProc (ByVal strArg As String)
тіло процедури
End Sub

При передачі параметрів за посиланням процедура одержує доступ до області пам'яті, у якій ця змінна зберігається, у результаті чого при зміні в процедурі параметра відбувається зміна значення змінної. За замовчуванням в VB передача параметрів у процедуру здійснюється за посиланням. Ключове слово **ByRef** можна опустити. При звертанні до процедури або функції відповідний аргумент може бути тільки змінною (не константою).

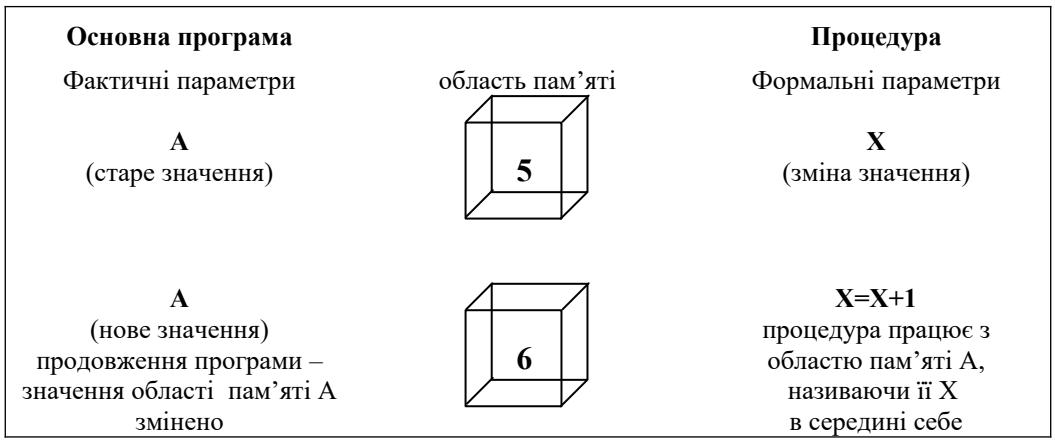

Рисунок 4.10 — Передача параметра за посиланням

Передача масиву. Масив або елемент масиву завжди передаються в процедуру за посиланням. Ім'я масиву або елемента завжди вказує адресу фактичного параметра, що дозволяє процедурі змінювати їхні значення. Наявність порожніх дужок після імені вказує, що передається масив, а не скалярна змінна. При цьому формальний параметр теж повинен містити круглі дужки.

Якщо потрібно передати *частину масиву*, то потрібно додати параметри, що вказують границі цієї частини.

Якщо потрібно передати *елемент масиву* — то вказати номер елемента в дужках після імені масиву.

Стандартні функції **LBOUND(A)** і **UBOUND(A)** визначають *найбільше* й *найменше значення індексів масиву* A.

Приклад:

```
Sub P2(a(),s)
S=0:
For i= Lbound(a) to Ubound(a)
        S=s+a(i)
Next i
End Sub
```

Використовуючи ключове слово **optional**, можна задавати необов'язкові параметри процедури. При цьому необхідно мати на увазі, що якщо який-небудь із параметрів заданий як необов'язковий, то й розташовані після нього параметри також повинні бути оголошені необов'язковими:

Приклад:

```
 Sub NameProc (Optional strArg1 As String, Optional strArg2 As String)
тіло процедури
 End Sub
```

Обов'язкові параметри процедури повинні задаватися до необов'язкових.

Приклад:

```
Sub NameProc (strArg1 As String, Optional strArg2 As String)
 тіло процедури
End Sub
```

Для того щоб процедура або функція була доступна у всіх модулях і формах проекту, вона повинна бути оголошена в модулі із ключовим словом **Public.** Процедура або функція, оголошена у формі, доступна процедурам і функціям тільки цієї форми (незалежно від застосованого ключового слова **Private** або **Public**).

5 НАЛАГОДЖЕННЯ ПРОГРАМ

Будь-яка програма, написана навіть кваліфікованим програмістом, містить помилки, які виявляються й виправляються в процесі налагодження програми. **Налагодження** — неодмінний етап роботи над будь-яким проектом. Як правило, це перевірка функціонування проекту й виправлення помилок перед передачею його на тестування. Для виконання налагодження в VB існує набір спеціальних інструментів. Крім того, при роботі з налагодженою програмою користувач може створити ситуацію, що програмою не

обробляється коректно. Розглянемо процес налагодження програм і способи обробки помилок при виконанні програми й наявні для цього засоби в VB.

5.1 Синтаксичний контроль

При помилці в наборі тексту програми у вікні коду автоматично (при активізації в діалоговому вікні **Environment Options** (*Параметри Середовища*) прапорця **Display Syntax Errors** (*Показувати синтаксичні помилки*) інверсним підсвічуванням виділяється неправильний фрагмент оператора програми. На цьому етапі VB відслідковує синтаксичні помилки (неправильно написані ключові слова, неправильний порядок операндів в операторах, некоректну пунктуацію й т.п.).

Виклик діалогового вікна **Environment Options** здійснюється з пункту **Tools** Головного меню. Це вікно дозволяє, крім вказівки на видачу синтаксичних помилок, визначити виведені вікна середовища (**Toolbox, Properties, Project, Debug**), задати обов'язковість оголошення змінних (**Require Variable Declaration)**, показати сітку форми (**Show Grid**) і її крок (**Width, Height**), вирівнювання елементів управління щодо сітки (**Align Controls to Grid**), автоматичне збереження поточних версій файлів форм і проекту перед кожним запуском програми (**Save Before Run**).

5.2 Контроль коректності алгоритму

Для подальшого налагодження синтаксично правильної програми існують спеціальні засоби, що дозволяють контролювати значення змінних на різних етапах виконання програми.

Інструментарії налагодження дозволяють проконтролювати вибрані ділянки коду додатка для локалізації помилки. Виконуючи додаток по кроках, зупиняючись у точках останову, вони дають можливість перевірити значення змінних, властивостей об'єктів і іншу інформацію, що цікавить, і з'ясувати, таким чином, джерело помилки.

У набір інструментарію налагодження VB входять такі *основні інструменти*:

- панель інструментів **Debug** (Налагодження) із кнопками команд для виконання налагодження додатка;
- вікно **Immediate** (Безпосереднє виконання), призначене для безпосереднього введення команд, що вимагають негайного виконання;
- вікно **Watches** (Спостереження), призначене для перегляду значень виразів, включених у список перегляду;
- вікно **Locals** (Локальні), призначене для перегляду значень змінних;
- редактор коду з вбудованими можливостями перегляду змінних, констант, властивостей, виразів при налагодженні додатка в точках останову й покроковому виконанні додатка;
- вікно **Call Stack** (Стек викликів) для перегляду викликаних, але незакінчених процедур.

Панель інструментів **Debug** (рисунок 5.1) активізується при виборі з меню **View** команди **Toolbars**, а потім значення **Debug**.

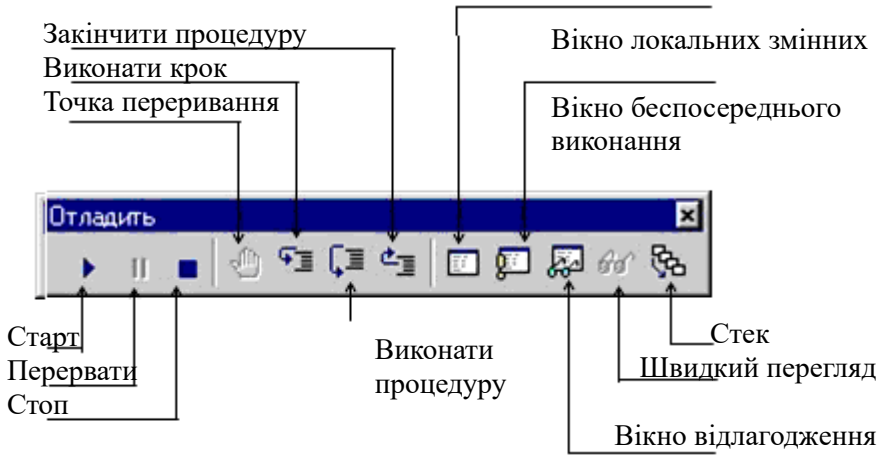

Рисунок 5.1 — Панель інструментів **Debug**, використовувана для налагодження додатка

На панелі інструментів **Debug** містяться кнопки, що забезпечують роботу з налагодження додатка. Призначення цих кнопок описане в таблиці 5.1.

Таблиця 5.1 — Кнопки панелі інструментів Debug

Кнопка	Назва	Призначення
	**Start/Continue** (Продовжити)	Запускає програму або продовжує її виконання після переривання
	**Break** (Останов)	Викликає переривання програми в необхідному місці (без використання точок останову)
	**End** (Завершення)	Завершує виконання програми
	**Toggle Breakpoint** (Установити точку останову)	Установлює точку останову на поточному рядку коду
	**Step Into** (Крок із заходом у процедури)	Здійснює покрокове виконання процедури, включаючи також процедури, що викликаються нею
	**Step Over** (Крок без заходу в процедури)	Здійснює покрокове виконання процедури без трасування процедур, що викликаються нею
	**Step Out** (Крок з виходом із процедури)	Виконує покрокове виконання поточної процедури до виходу з неї
	**Locals Window** (Вікно Locals)	Відкриває діалогове вікно Locals для контролю значень змінних
	**Immediate Window** (Вікно Immediate)	Відкриває вікно Immediate для введення й виконання команд
	**Watch Window** (Вікно Watch)	Відкриває діалогове вікно Watches для перегляду виразів, включених у список спостереження
	**Quick Watch** (Швидкий перегляд)	Відкриває вікно Quick Watch для перегляду виразів й значень, що воно повертає в цей момент у точці останову програми або на певному кроці при покроковому запуску програми
	**Call Stack** (Стек викликів)	Відкриває вікно Call Stack, у якому поданий список виконуваних процедур

Команда контекстного меню редактора коду **Toggle** (Установка) відкриває меню, що містить команди, описані в таблиці 5.2.

Таблиця 5.2 — Команди меню Toggle

Команда	Призначення
Breakpoint (Точка останову)	Встановлює точку останову в поточному рядку
Break on All Errors (Переривати на всіх помилках)	Встановлює автоматичний перехід у режим переривання програми з можливістю продовження роботи з місця переривання при виникненні будь-якої помилки
Break in Class Module (Перервати в модулі класу)	Призначає переривання роботи програми при виникненні помилок у модулях класів
Break on Unhandled Errors (Перервати за помилками, що не управляються)	Призначає переривання роботи програми за помилками, що не управляються
Bookmark (Закладка)	Установлює спеціальні мітки (закладки) у лівому вертикальному полі редактора

Зауваження. Нагадаємо, що настроювання параметрів редактора здійснюється на вкладках Editor і Editor Format діалогового вікна Options, що відкривається командою Options меню Tools.

Вікно **Debug** є основним засобом для пошуку помилок. У верхньому полі вікна виводяться:

- вид виразу (стовпчик **Expression**);
- значення виразу (стовпчик **Value**);
- місцезнаходження виразу (стовпчик **Context**).

Встановлена точка переривання виділиться підсвічуванням (рисунок 5.2). При запуску програми в точці останову виконання програми припиняється й для контролю роботи додатка можна використовувати увесь інструментарій налагодження: переглядати значення змінних і виразів при позиціонуванні маркера на обраній змінній або виразі (рисунок 5.3).

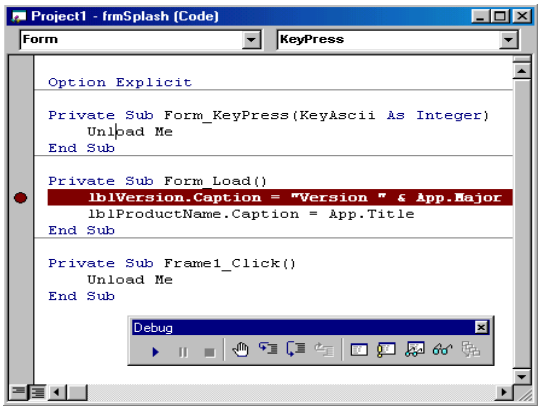

Рисунок 5.2 — Точка останова у вікні редактора коду

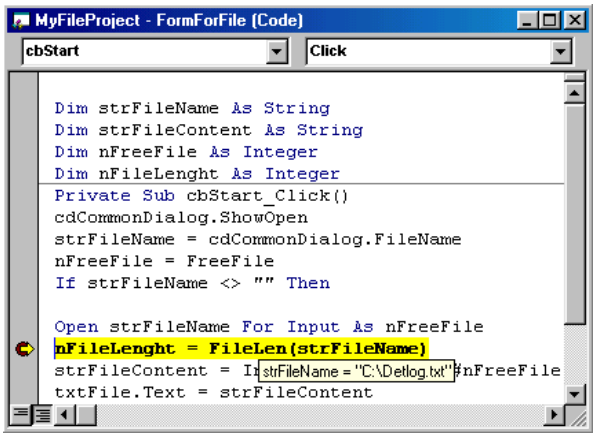

Рисунок 5.3 — Перегляд значення виразу в точці останова

Для більш ретельного контролю роботи додатка можна використовувати *вікна перегляду*: **Immediate** (Негайне виконання), **Watches** (Спостереження), **Locals** (Локальні), **Quick Watch** (Швидкий перегляд). **Call Stack** (Стек викликів).

Вікно **Immediate** (Безпосереднє виконання) призначено для ручного введення й виконання команд VB (рисунок 5.4). Це вікно з'являється автоматично при перериванні роботи програми в точках останову програми. Для виконання команди або оператора VB необхідно набрати рядок команди й натиснути клавішу **<Enter>**.

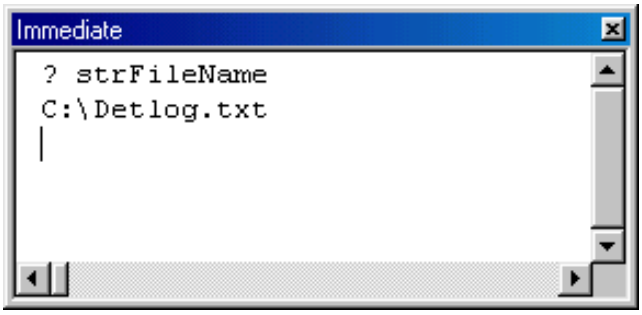

Рисунок 5.4 — Вікно **Immediate** для введення команд і одержання результатів

Крім прямого введення й виконання команд, вікно **Immediate** служить для з'ясування значення змінних і виразу. Для цього необхідно виконати команду з вказівкою виразу, що цікавить, як це показано на рисунок 5.4. Значення виразу виводиться в наступному рядку цього ж вікна.

У сполученні з вікнами **Locals** і **Watches** вікно **Immediate** є зручним засобом для налагодження й перевірки роботи програми в покроковому режимі. Частини, що перевіряються, або блоки програми можна копіювати із програмних модулів додатка у вікно **Immediate**, і після перевірки й внесення необхідних змін за результатами контролю повертати в модуль додатка.

При роботі з вікном **Immediate** зручно використовувати об'єкт **Debug** і його метод **Print**. У режимі запуску метод **Debug.Print** виводить текстове повідомлення у вікні **Immediate**. Синтаксис цього методу:

Debug.Print *StringMessage*

де ***StringMessage*** — текст виведеного у вікно **Immediate** повідомлення.

Вікно **Locals** (Локальні) призначено для перегляду списку локальних змінних додатка й контролю за їхніми значеннями (рисунок 5.5). Викликається це вікно командою **Locals Window** (Вікно локальних змінних) меню **View**. У вікні **Locals** можна переглянути локальні змінні, оголошені в поточній процедурі, їхній тип і значення. Ця інформація автоматично з'являється у вікні при його виклику. Дане вікно використовується для налагодження й перевірки роботи додатків. За допомогою його можна проконтролювати всі локальні змінні в точках останову програми.

При роботі з вікном Locals (Локальні) необхідно мати на увазі, що глобальні змінні в цьому вікні для перегляду недоступні.

Для вибору процедури використовується поле зі списком. Натиснувши на кнопку із точками, можна вибрати процедуру, що цікавить, зі списку. Якщо значення змінних або їхні назви не містяться в стовпцях, то необхідні стовпці можна розширити, переміщаючи за допомогою лівої кнопки миші границі стовпців у заголовку.

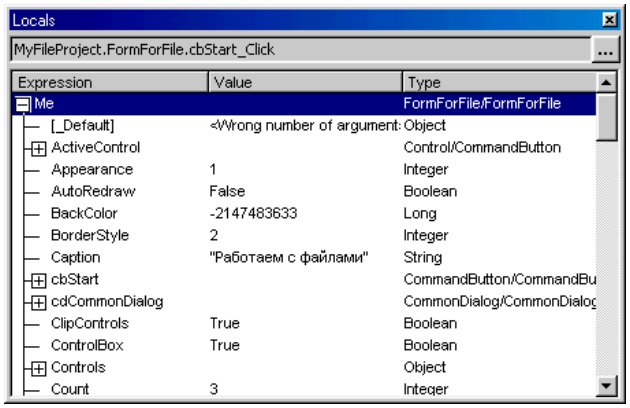

Рисунок 5.5 — Вікно **Locals** для контролю за змінними

Якщо вікно відкрите постійно, то дані між точками останову програми при роботі додатка автоматично обновлюються.

У вікні **Locals** можна не тільки переглядати змінні і їх значення в момент роботи програми. Дуже корисною властивістю цього вікна є можливість змінювати значення змінних для перевірки реакції програми на ці значення. Для цього в стовпці **Value** (Значення) необхідно клацнути на змінюваному значенні. При цьому значення переведеться в режим редагування, і його можна буде змінити. Клавішею **<Enter>** або переміщенням покажчика миші на інше поле встановлюється нове значення, якщо воно має припустиме значення.

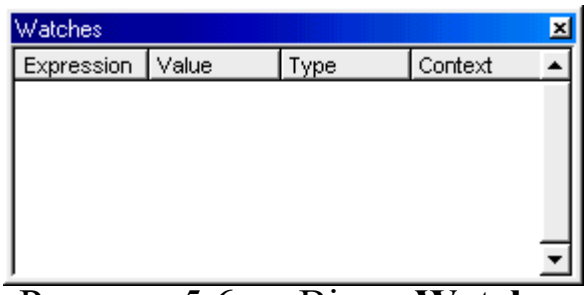

Рисунок 5.6 — Вікно **Watches** для контролю за значеннями виразів

Основним вікном для перегляду значень виразів є вікно **Watches** (Спостереження). (рисунок 5.6). Це вікно викликається командою **Watch Window.** В ньому можна проконтролювати будь-який вираз зі списку, введеного для перевірки.

Список виразів, що перевіряються, поповнюється в діалоговому вікні **Add Watch** (рисунок 5.7), що відкривається командою **Add Watch** (Додати спостереження) контекстного меню редактора коду або однойменною кнопкою на панелі інструментів **Debug**. Кнопки в області **Watch Type** визначають умови виведення виразу.

Перемикач **Watch Type** (Тип перегляду) визначає, як VB реагує на спостережуваний вираз. При установленні опції **Watch Expression** (Вираз спостереження) значення виразу відображається в діалоговому вікні **Watches** у режимі переривання. Опція **Break When Value Is True** (Призупинити при рівності виразу значенню **Істина**) — ***Установити точку переривання*** — автоматично задає режим переривань і звернення до контролю виразів за умови рівності виразу значенню **True** або ненульовому значенню для числових виразів. Опція **Break When Value Changes** (Призупинити при зміненні значення виразу) — ***Установити точку спостереження*** — аналогічна опції **Break When Value Is True**, але додаток входить у режим переривання при зміні значення виразу.

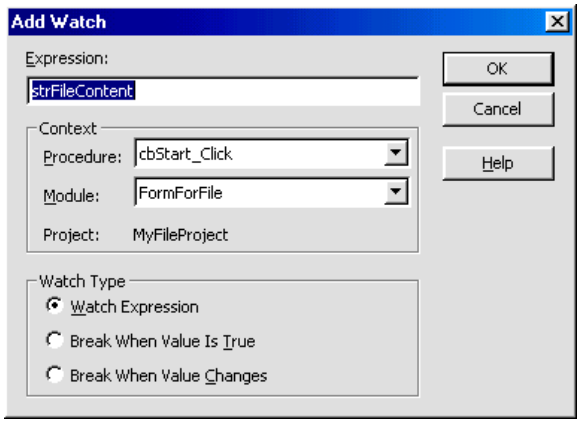

Рисунок 5.7 — Вікно **Add Watch** для додавання контрольованого виразу

Видалити точку переривання можна, клацнувши мишею по пункту **Clear Toggle Breakpoint** — ***Забрати точку переривання*** меню, що розкривається, **Run** Головного меню.

При налагодженні програм доцільно комбінувати установлення точок переривання й точок спостереження. Точки

спостереження сповільнюють виконання програми. Краще встановити точку переривання в підозрілому місці програми й виконувати програму з нормальною швидкістю до цієї точки. Далі поставити одну або кілька точок спостереження й продовжувати більше повільний пошук помилок у локалізованій області програми.

Область **Context** діалогового вікна **Add Watch** дозволяє встановити область видимості змінних, спостережуваних у виразі. Список, що розкривається, **Procedure** задає видимість усередині процедур, а список, що розкривається, **Module** — усередині програмних модулів. Ці списки дозволяють задати місце розташування виразу в програмі. Для додавання виразу в список контрольованих необхідно ввести його вручну в текстове поле **Expression** (Вираз) цього вікна. Крім того, якщо в тексті програми перед викликом вікна виділений який-небудь оператор або його частина, то воно автоматично з'являється у вікні.

На додаток до вікна **Watches**, для перегляду виразів можна використовувати вікно **Quick Watch** (Швидкий перегляд) для швидкого доступу до значення виразу (рисунок 5.8). Для цього вікна не потрібно вводити список контрольованих виразів, а досить виділити вираз, що цікавить, у тексті коду й викликати вікно **Quick Watch** однойменною кнопкою на панелі інструментів **Debug**.

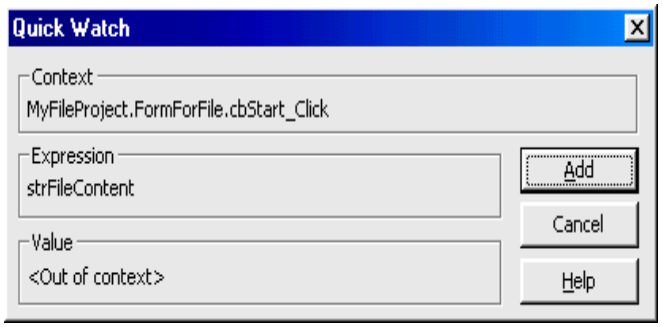

Рисунок 5.8 — Вікно **Quick Watch** для швидкого перегляду

Якщо проект включає багато процедур, то корисним засобом налагодження є трасування виклику процедур. Для контролю виконуваних процедур використовується вікно **Call Stack** (Стек викликів), у якому показані всі викликані й активні в цей момент процедури (рисунок 5.9). Вікно показує всю послідовність викликів від вихідної процедури до поточної (у верхній частині розташована остання викликана процедура, у нижній — перша). Список дозволяє визначити, як відбувся перехід у поточну точку програми. З вікна **Call Stack** можна перейти до коду програми (рисунок 5.10), звідки був виконаний виклик обраної в списку процедури, за допомогою кнопки **Show** (Показати).

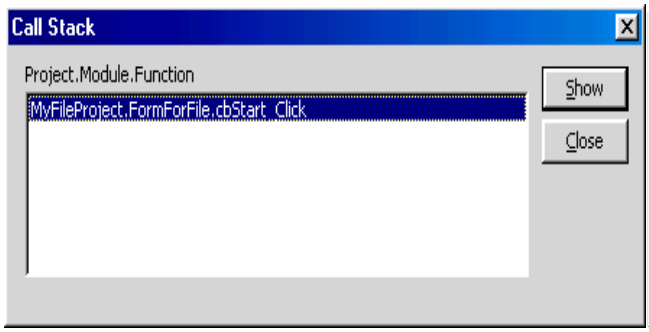

Рисунок 5.9 — Вікно **Call Stack** для перегляду викликаних процедур

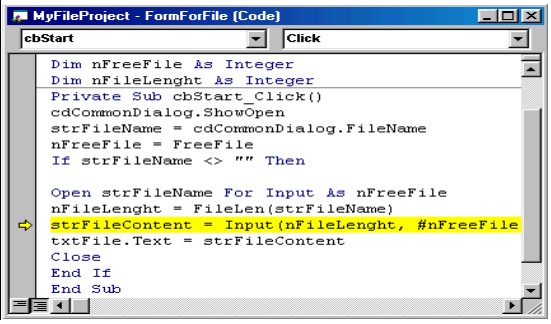

Рисунок 5.10 — Місце виклику процедури, обраної на рисунку 5.9

На рисуноку 5.10 показане місце виклику процедури, обраної у вікні **Call Stack** (рисунок 5.9). Ефективним засобом налагодження є також покрокове виконання програми й спостереження результатів виконання кожного оператора. Після припинення виконання програми (точки переривання або кнопка **Break**) необхідно ввійти у вікно редагування коду. Для виконання одного оператора використовується клавіша **F8.** При цьому буде виконаний оператор, виділений рамкою, а виділення буде перенесено на такий оператор. Повторюючи натискання клавіші, можна здійснювати покрокове виконання програми.

Якщо поточний оператор містить виклик процедури, натискання клавіші **F8**, клацання миші по пункту **Step Into** (*Крок усередину*) меню, що розкривається, **Run** або клацання миші по піктограмі **Step Into** Головного меню викличе виконання першого оператора процедури, яку можна продовжувати виконувати по кроках. Для виконання всієї процедури (не по кроках) і переходу до оператора, що випливає за викликом процедури, використовується клацання миші по пункту **Step Over** (*Обійти*) або комбінація клавіш **Shift+F8**.

При покроковому виконанні деякий блок операторів можна обійти. Це можна зробити, установивши курсор на оператор, на якому необхідно виконати наступне переривання виконання програми, і клацнувши мишею по пункту меню **Step To Cursor** або комбінацією клавіш **Ctrl+F8**.

Покрокове виконання можна комбінувати із процедурним виконанням. Налагоджені процедури однієї програми можна виконувати попроцедурно, а не налагоджені — покроково. Засоби налагодження дозволяють також змінювати порядок виконання операторів. При виконанні оператора **Set Next Statement**

(Установити такий оператор) меню, що розкривається, **Run** можна перейти до виконання будь-якого оператора процедури (навіть до тих, які розташовані перед поточним).

5.3 Контроль помилок на етапі виконання програми

Помилки етапу виконання (**runtime errors**) можуть виникати, наприклад, через недостачу пам'яті або дискового простору, спроби відкрити відкритий іншим додатком файл, за умови виходу індексу за межі розмірності масива й ін. У цьому випадку оброблювач помилок VB виводить діалогове вікно з відповідним повідомленням і припиняє виконання програми. Для цього призначені перераховані нижче вбудовані можливості VB:

- службова змінна **Err**, що містить код помилки;
- службова змінна **Error**, що містить текст системного повідомлення про помилку;
- об'єкт **Err**, що містить одночасно код і повідомлення про помилку.

Список кодів та повідомлень міститься в довідковій системі (**Help**) і документації до VB.

Такі помилки можна обробляти методом перехоплення помилок (**error trapping**). Для цього VB містить спеціальний оператор

On Error

Перший варіант синтаксису цього оператора має вигляд

On Error Go To *StringLabel*

Якщо в період виконання помилка виникне в одному з операторів процедури, розташованих за **On Error**, то управління передається оброблювачеві помилок, зазначеному міткою ***StringLabel,*** *яка* повинна бути унікальною в межах процедури.

Приклад: *код виконує обробку помилки*

```
On Error Go To ErrorLabel
текст коду процедури
ErrorLabel:
Call ErrorProcedure ()
End
```

У цьому випадку при виникненні помилки буде виконуватися оператор **Call ErrorProcedure ()**, що викликає процедуру обробки помилки.

Для того щоб відключити обробку помилок у якій-небудь процедурі, фрагмент обробки помилок можна закінчити оператором з *другим варіантом синтаксису*

On Error Go To 0

Для *ігнорування помилки* в оброблювач помилок можна включити оператор **Resume** і продовжити виконання програми після виправлення помилки операторами, що вставляються в оброблювач помилок.

Оператор **Resume** має кілька форм:

- **Resume** відновляє виконання програми з оператора, що викликав помилку;
- **Resume Next** відновляє виконання програми з оператора, розташованого наступним за тим, що викликав помилку;
- **Resume *StringLabel*** відновляє виконання програми з оператора, позначеного міткою ***StringLabel.***

Приклад: *код обробляє помилки:*

On Error Go To ErrorLabel

Текст коду процедури

ErrorLabel:
Call ErrorProcedure ()
Resume NextStatement
текст коду процедури
NextStatement:
текст коду

При використанні цього варіанта обробки помилки виконання програми не зупиниться, як у випадку з попереднім кодом, а будуть виконуватися оператори коду, розташованого після мітки продовження роботи.

5.4 Оптимізація додатків

Метою будь-якої розробки є створення максимально ефективного додатка. Однак часто деякі параметри додатків можуть бути суперечливими (наприклад, розмір додатка й швидкість його роботи), і потрібно підібрати оптимальне вирішення для їхньої реалізації в проекті. Пошук реалізації додатка, який ефективно працює в умовах протиріч і обмежень, і є оптимізація.

Як правило, метою оптимізації є створення додатка, що виконує всі поставлені перед ним функціональні завдання максимально швидко й при цьому з використанням мінімальної кількості ресурсів системи, зокрема місця на диску й пам'яті комп'ютера при завантаженні додатка. Перед проведенням оптимізації необхідно чітко уявити, які параметри є більш важливими: швидкість або ресурси. Оптимізація додатка, таким чином, це ціла стратегія створення ефективного додатка.

Розглянемо оптимізацію додатка, що включає в себе такі основні розділи:

- оптимізацію швидкості виконання додатком обчислень і інших дій, вимірюваних часом;
- оптимізацію розміру додатка;
- оптимізацію розміру графіки додатка.

Звичайно, це далеко не повний список, але головні напрямки оптимізації він відбиває.

Перед тим як будувати стратегію оптимізації, рекомендується відповісти на такі три питання:

- Що оптимізувати?
- Де оптимізувати?
- Коли завершити оптимізацію?

Відповідаючи на перше питання, необхідно виділити цілі оптимізації. Наприклад, додаток для розрахунку заробітної плати повинний працювати максимально швидко, при цьому можна пожертвувати його розміром. Навпаки, створюючи додаток для Internet, необхідно пам'ятати, що проект великого розміру може працювати в мережі Internet проблемно.

Друге питання визначає місце оптимізації для досягнення максимального ефекту за певний час. Оптимізовувати додаток можна нескінченно. Однак, як правило, на цю процедуру приділяється певний час, який можна використовувати на вирішення інших питань або на розробку інших додатків. Тому треба прагнути домагатися максимальної оптимізації мінімальними зусиллями. Наприклад, якщо оптимізується швидкість додатка, першою справою потрібно звернути увагу на цикли й на роботу додатка усередині циклів, тобто зменшити кількість кроків циклу до необхідного. І навпаки, на процедури, викликувані рідко, варто звернути увагу в останню чергу або взагалі не оптимізувати їх.

Третє питання має на увазі, що оптимізацію не слід проводити нескінченно, якщо існують обмеження, не пов'язані з роботою коду додатки. Деякі параметри додатка можуть прямо залежати від параметрів системи: швидкості роботи диска або мережі. Тому варто вчасно зупинитися в оптимізації параметрів додатка, коли її результат уже в малому ступені залежить від

додатка. Наприклад, при роботі з диском таким обмеженням буде швидкість виконання операцій запису/читання даних з диска.

Оптимізація швидкості роботи додатка

Основний спосіб оптимізації швидкості роботи — це оптимізація коду додатка. При цьому бажано прислухатися до рекомендацій розроблювачів VB.

Уникайте змінних типу **Variant** і присвоюйте відповідні їхньому застосуванню типи. При використанні у виразі змінних типу **Variant** губиться час на їхнє приведення до конкретного типу відповідно до типу виразу.

Причому, необхідно мати на увазі, що при множинному описі ключові слова **As Type** відносяться тільки до однієї змінної, тобто Dim A, B,C As String означає, що A і B — змінні типу variant (за замовчуванням) і тільки C — рядкова. У цьому випадку краще написати Dim A As string, B As String, C As string.

Використовуйте для арифметичних обчислень змінні типу **Long** або **Integer**, оскільки вони, на відміну від **Currency**, **Single** або **Double**, найбільше відповідають машинному коду. В таблиці 5.3 наведене ранжирування за відносною швидкістю типів змінних при арифметичних обчисленнях.

Таблиця 5.3 — Відносна швидкість обчислення для типів змінних

Тип змінної	Відносна швидкість
Long	Найвища швидкість
Integer	Нижче, ніж в Long
Byte	Нижче, ніж в Integer
Single	Нижче, ніж в Byte
Double	Нижче, ніж в Single
Currency	Найнижча швидкість

Призначайте часто використовувану властивість об'єкта в змінні, тому що призначення й читання змінних працює швидше (від 10 до 20 разів). Для порівняння наведені два приклади із двома варіантами тексту коду, при цьому другий варіант коду більше швидкий, ніж перший, і в тім, і в іншому прикладі.

Приклад 1:

‘Перший варіант коду:

```
For nCounter = 1 To 20
 Object(nCounter).Property = ObjectDef.Property
Next nCounter
```

‘Другий варіант коду:

```
 valProperty = ObjectDef.Property
 For nCounter = 1 To 20
 Object(nCounter).Property = valProperty
Next nCounter
```

Приклад 2:

‘Перший варіант коду:

```
For nCounter = 1 To 20
 Object.Property = Object.Property & sValue
Next nCounter
```

‘Другий варіант коду:

```
For nCounter = 1 To 20
 sValueAll = sValueAll & sValue
Next nCounter
Object.Property = sValueAll
```

Для зберігання однакових значень у процедурах замість змінних типу **Static** використовуйте змінні рівня **модуля**, оскільки вони працюють швидше. При цьому, щоправда, текст коду додатка збільшується за рахунок дублювання й стає менш читабельним і зрозумілим.

Для критичних за часом обчислення випадків використовуйте код процедур безпосередньо в місці їхнього виконання замість виклику цих процедур, що забирає певний час. Правда, при цьому необхідно пам'ятати, що розмір коду збільшується за рахунок дублювання коду процедур.

Замість змінних, наскільки це можливо, використовуйте константи, оскільки значення констант включаються при компіляції в код додатка, а змінні щораз повинні бути знайдені в пам'яті й зчитані, що, звичайно, забирає певний час.

Оптимізація коду додатка, звичайно, зв'язана й з оптимізацією його розміру. Як видно з деяких способів

оптимізації коду, розмір коду або збільшується, або зменшується при оптимізації швидкості додатка.

При оптимізації розміру додатка під розміром будемо розуміти як розмір файлу, що виконується, так і розмір завантаженого в пам’ять додатка. Особливо критичним розмір є для додатків, що працюють у мережі Internet. Тому такі додатки варто робити якомога меншого розміру. Якщо це не виходить, можна розділити великий додаток на декілька невеликих, які будуть виконувати закінчено і функції й завантажуватися в міру необхідності.

Часткову оптимізацію розміру коду виконує сам VB. При компіляції додатка у файл, що виконується, порожні рядки й рядки коментарів пропускаються, тому на них можна не заощаджувати. У тому числі можна не заощаджувати на довжині імен ідентифікаторів, які також оптимізуються компілятором.

Для оптимізації розміру коду підійдуть такі основні рекомендації:

- зменшуйте кількість завантажених форм. Для форм, які закриваються, застосовуйте оператор: set Form = Nothing;
- зменшуйте у формах, наскільки це можливо, кількість елементів управління. При цьому краще користуватися масивами елементів управління;
- для виведення тестових значень максимально використовуйте об'єкти Label замість TextBox;
- для зберігання даних використовуйте файл ресурсів і завантажуйте дані тільки при необхідності;
- уникайте змінних типу Variant, що вимагають 16 байт для зберігання. Для порівняння: змінні типу Integer вимагають 2 байти, змінні типу Double — 8 байт;
- уникайте "мертвого" коду — тобто процедур і змінних, які колись були потрібні, але в цей час не використовуються. Їх треба видалити або закоментувати.

Оптимізація розміру графіки додатка

Графіка займає істотний розмір додатка, тому важливо використовувати її оптимально. Для цього підходять такі рекомендації:

- для виведення зображень використовуйте елемент управління типу **Image**, а не **Picture**, оскільки останній має атрибути вікна й відповідно вимагає більше ресурсів;
- завантажуйте зображення тільки при необхідності, а не зберігайте їх в елементах управління;

▪ якщо наявність зображення не потрібна, призначайте властивості Picture об'єктів значення Nothing:

Set Object. Picture = Nothing.

Наведений тут список не містить усіляких рекомендацій і прийомів оптимізації додатків. У цьому питанні вам може допомогти тільки практика роботи з VB, оскільки створення оптимальних додатків те саме, що майстерність, відточувана розробкою додатків для VB.

6 РОБОТА З ФАЙЛАМИ Й ОРГАНІЗАЦІЯ ДРУКУ

Багато програм використовують дані, що вводяться із клавіатури: обробляють їх, зберігають у масивах і змінних, виводять безпосередньо на екран. Однак у всіх програмах дані зникають із пам'яті комп'ютера, як тільки програма закінчує свою роботу.

Основним способом збереження інформації, отриманої в ході виконання програми, служить запис її на тверді або гнучкі диски. Це особливо важливо, якщо обсяг інформації великий, а дані передбачається використати неодноразово в цій або в інших програмах.

При проектуванні додатка досить часто виникає необхідність працювати безпосередньо з файлами. Це потрібно, наприклад, для додавання, видалення файлів або каталогів (папок), запису даних у файли або читання з них як програмно, так і в інтерактивному режимі. Необхідність роботи з файлами виникає також при створенні програми інсталяції розробленого додатка на користувальницькі комп'ютери, читання даних з файлів при ініціалізації додатка з використанням файлів настроювання, організації виведення файлів на друк. Для цих цілей VB надає повний набір функцій, які працюють із файлами, папками й пристроями, що дає можливість робити з ними всі необхідні дії.

Використання файлів дає такі переваги:

a) можливість зберігати дані на зовнішніх носіях тривалий час, а не тільки під час виконання програми;

b) кількість даних може бути більшою, заздалегідь невідомою, тобто під час роботи з файлом не потрібно знати число записів у ньому;

c) дані в запису можуть бути різних типів;

d) дані зберігаються окремо від програм, які їх обробляють, і можуть бути використані різними програмами й користувачами;

е) між даними різних файлів може бути певна відповідність (релятивістські зв'язки), що дасть можливість не дублювати ті самі дані в різних файлах.

Інформація, розташована на зовнішньому носії, що має певну логічну структуру й ім'я, називається *набором даних*. Він може містити як програми на вхідних мовах, так і оброблені дані (каталог книг бібліотеки, відомості про абітурієнтів ВНЗ й т.п.).

Набір даних складається з *фізичних записів*.

Фізичний запис – найменша порція даних, що переноситься із МД (магнітного диску) або на МД за одну фізичну операцію введення-виведення. Вона являє собою одиницю носія — сектор МД (512 Б).

Файл – символічне подання набору даних як сукупності *логічних записів* однакової структури.

Логічний запис – змістовна одиниця збереженої у файлі інформації, що є сукупністю відомостей про деякий об'єкт.

Приклад 1. Файл — список студентів групи, логічний запис — містить поля: прізвище, ім'я студента, рік народження.

Розмір логічного запису залежить від розв'язуваного завдання. Файл зв'язується з набором даних при так званому відкритті файлів у програмі.

Кількість записів у файлі може бути довільною. За самим останнім записом знаходиться службова мітка кінця файлу (код 26). Її записує у файл VB. Файл може бути порожнім, тобто в ньому не буде жодного запису, але ім'я файлу й мітка кінця будуть обов'язково. Записи можуть містити дані різних типів.

Приклад 2:

" Іванов", "Петро", 1986

Такий запис містить три поля-параметри:

- прізвище — текстовий тип;
- ім'я — текстовий тип;
- рік народження — цілий числовий тип.

В VB існує поняття *типу файлу*, що визначається організаційною структурою зберігання інформації у файлі й способом доступу до цієї інформації.

Прийнято виділяти такі *типи файлів*:

- ***файли послідовного доступу***. Такі файли являють собою послідовність символів. При цьому дані можуть бути з роздільниками або без роздільників, тобто зміст файлу може мати

якусь структуру. Структурною одиницею вмісту в подібних файлах, як правило, є рядок. Прикладами цих файлів можуть служити текстові файли й файли ініціалізації програм;

- ***файли довільного доступу***. Це структуровані файли, які містять інформацію у вигляді записів. Прикладом можуть служити файли баз даних;
- ***двійкові (бінарні) файли***. Файли з побайтним доступом. В принципі, це теж файли з послідовним доступом, але інформація в них не організована в рядки. Особливість даних файлів — робота з байтами або блоками байтів. До таких файлів можна віднести виконувані програми, файли динамічних бібліотек, файли документів Word.

Подібний розподіл файлів на типи досить умовний і визначається особливостями організації файлів і доступу до даних у них. Тип файлу задає оптимальний набір функцій запису й читання даних з файлу. Тому при роботі з файлами для написання ефективної програми завжди необхідно мати дані про типи файлів, з якими буде працювати програма, і про організацію зберігання даних у цих файлах. Це дає можливість забезпечити оптимальний доступ і використовувати відповідні цьому доступу функції.

6.1 Традиційний підхід при роботі з файлами

Традиційний підхід при роботі з файлами залишається незмінним практично з найперших версій VB і полягає у використанні функцій і операторів, що забезпечують прямий доступ до інформації у файлах. Функції й оператори, використовувані при роботі з файлами, наведені в таблиці 6.1[2]. У даній главі ми розглянемо тільки основні функції й оператори, необхідні для одержання навичок роботи з файлами.

[2] У стовпці **Тип файлу** цієї таблиці прийняті такі скорочення типів файлів: П - файл послідовного доступу; Пр - файл довільного доступу; Б - бінарний файл.

Таблиця 6.1 — Функції й оператори для роботи з файлами

Функція, оператор	Опис	Тип файлу
Open	Відкриває файл	П, Пр, Б
Close	Закриває всі файли	П, Пр, Б
Close #	Закриває файл за ідентифікатором (дескриптором)	П, Пр, Б
Reset	Закриває всі відкриті файли, записує вміст буферів	П, Пр, Б
Print tt	Записує дані у файл	П
FileCopy	Копіює файл	П, Пр, Б
EOF	Визначає мітку кінця файлу	П, Пр, Б
FileAttr	Повертає режим доступу відкритого файлу	П, Пр, Б
FileDateTime	Повертає дату й час створення файлу	П, Пр, Б
FileLen	Повертає розмір файлу в байтах	П, Пр, Б
FreeFile	Повертає номер вільного ідентифікатора файлу (дескриптора)	П, Пр, Б
GetAttr	Одержує атрибути файлу	П, Пр, Б
SetAttr	Установлює атрибути файлу	П, Пр, Б
Loc	Повертає номер поточної позиції у файлі	Пр, Б
LOF	Повертає розмір відкритого файлу в байтах	П, Пр, Б
Seek	Установлює на задану номером позицію або запис у файлі	П, Пр, Б
Dir	Повертає вміст поточної папки	П, Пр, Б
Kill	Видаляє файл	П, Пр, Б
Lock	Блокує файл при роботі в багатокористувальницькому середовищі	П, Пр, Б
Unlock	Знімає блокування файлу в багатокористувальницькому середовищі	П, Пр, Б
Name	Задає (перейменовує) ім'я файлу	П, Пр, Б
Get #	Читає дані з файлу	Ін.Б
Input	Читає дані з файлу	П, Б
Input #	Читає дані з файлу	П
Line Input #	Читає рядок з файлу	П
Put #	Записує дані у файл	Пр, Б
Write #	Записує дані у файл	П

Для зручності згрупуємо функції й оператори за виконуваною дією, як це прийнято в VB. Таке об'єднання зручно при виборі функції або оператора для виконання необхідних дій з файлами (таблиці 6.2).

Таблиця 6.2 — Функції й оператори для роботи з файлами по групах

Виконувана дія	Функції, оператори
Відкрити або створити файл	Open
Закрити файл	Close, Reset
Визначення параметрів виведення даних	Format, Spc, Tab, Width #
Скопіювати файл	FileCopy
Одержати інформацію про файл	EOF, FileAttr, FlleDateTime, FileLen, FreeFile, GetAttr, Loc, LOF
Організувати управління файлами	Dir, Kill, Lock, Unlock, Name
Прочитати дані з файлу	Get #, Input, Input ft, Line Input #
Одержати інформацію про розмір файлу	FileLen
Установити атрибути файлу	SetAttr
Знайти позиції у файлі	Seek
Записати дані у файл	Print tt, Put #, Write ft

6.2 Робота з файлами послідовного (Sequential) доступу

При послідовному доступі (рисунок 6.1), записи зчитуються послідовно один за іншим. Інформація в них зберігається в текстовому форматі (у вигляді ланцюжка кодів ASCII). Щоб ***створити послідовний файл,*** необхідно:

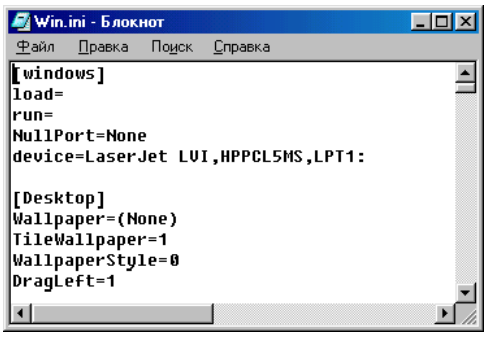

Рисунок 6.1 — Приклад файлу послідовного доступу

а) відкрити файл оператором OPEN для доступу до нього й введення або дозапису в нього даних;
б) записати дані у файл операторами PRINT або WRITE;
в) закрити файл оператором CLOSE.

Щоб ***прочитати послідовний файл,*** необхідно:
а) відкрити його оператором OPEN для доступу до нього й виведення з нього даних;

б) прочитати дані з файлу в програму, використовуючи Input, Line Input# або Input#;

в) закрити файл оператором CLOSE.

Відкриття файлів. Як було зазначено вище, робота з кожним з типів файлів має свої особливості. Однак є дві дії, загальні для всіх типів файлів — їхнє відкриття й закриття. Зрозуміло, що перед тим як записати дані у файл або прочитати дані з файлу, необхідно спочатку відкрити цей файл. Відкриття файлу виконується оператором **Open**:

OPEN *pathName* FOR *режим* AS [#]***номер_файла %***

- *pathName* — повне ім'я файла; містить не тільки безпосередньо його назву, але й назву дискового пристрою та директорії, де знаходиться або буде створений файл;
- *номер_файла %* (дескриптор) — ціле число від 1 до 255;
- # — необов'язковий знак перед номером файла.

При відкритті файлів послідовного доступу можливі ***три режими доступу***:

- **Input** — відкритий для послідовного читання даних;
- **Output** — відкритий для послідовного запису даних, при цьому інформація записується завжди з початку файлу (попередня затирається, якщо у файлі вже щось записано);
- **Append** — відкритий для додавання даних до уже наявних у файлі.

При роботі оператора **Open** створюється спеціальний лічильник номерів (ідентифікаторів) відкритих файлів (в операторі це параметр **fileNumber**) для однозначного визначення файлу, з яким програма працює в даний момент. Якщо номер файлу, що відкривається, спеціально не контролюється й не задається програмою, його можна довідатися за допомогою функції **FreeFile**, що повертає останній вільний номер файлу, який відкривається.

Запис даних у файл послідовного доступу. Дані у файл послідовного доступу записуються за допомогою операторів **Print #** й **Write #.**

WRITE # *номер_файла%*, [*список значень*]

PRINT # *номер_файла%*, [*список значень*]

- ***список значень*** — список констант або /і змінних або/і виразів, значення яких записуються у файл.

Вирази в списку розділяються комами, пробілами або крапкою з комою. Якщо список значень відсутній, у файл буде занесений порожній рядок.

Оператор WRITE. Роздільником у списку значень є кома. Елементи списку заносяться у файл в один текстовий рядок.

Символьні дані (String) записують у лапках. Ця команда сама заносить коми між полями й більш ощадливо розміщає дані на носіях. Після запису останнього елемента рядка автоматично записуються символи “повернення каретки” — CHR(13) і “перехід на новий рядок” — CHR(10).

Оператор PRINT. Роздільниками в списку значень є:

- кома — значення записуються в 14-символьні зони виведення (після кожного даного автоматично заноситься символ табуляції);
- крапка з комою — значення записуються підряд, без проміжків між ними.

PRINT зручний для ретельного редагування тексту вихідного файлу, а WRITE краще використовувати в тому випадку, коли вихідний файл буде використовуватися далі як вхідний для інших програм.

Приклад: *програма запису даних (рядків) у файл із ім'ям* РікНародження.txt

ІвановПетро1986
СуринаМарина1987

```
Dim Fam As String, Im As String, God As Integer
Open “ РікНародження.txt” for Output As#1
For i=1 to 2
        Fam = InputBox(" Прізвище “)
        Im = InputBox (“Ім'я”)
        God = Val(InputBox (“Рік народження”) )
        Write #1, Fam, Im, God
Next i
Close #1
```

Результат
“Іванов”, “Петро”, 1986
“Сурина”, “Марина”, 1987

Замінимо **Write #1, Fam, Im, God оператором**

Print #1, Fam, Im, God

Результат
1 15 30
Іванов Петро 1986
Сурина Марина 1987

А тепер оператором **Print #1, Fam; Im; God**

Результат
1 7 12
ІвановПетро 1986
СуринаМарина 1987

Можна вставити коми в лапках у список значень PRINT, щоб розбити запис на більш дрібні частини й забезпечити доступ до цих частин (у майбутньому) командою читання. Дані можна

записувати не тільки в змінні, але й у заданий блок оперативної пам'яті.

Читання даних з файлу послідовного доступу. Читати дані можна різними способами:

- за допомогою операторів **Input#** і **Line Input#;**
- за допомогою функції **Input**

Кожна з функцій запису працює з певною парою читання**.**
Для Print # — функція **Input** або оператор **Line Input** # ;
Для Write # — оператор **Input#**

Розглянемо кожний спосіб детальніше.

1) **Line Input# номер_файла%, змінна**

де змінна – змінна типа String.

Оператор **Line Input #** посимвольно зчитує весь рядок даних з файлу й розміщує його в рядкову змінну. При цьому роздільником рядків у файлі служить стандартний роздільник рядків — символ повернення каретки-CHR(13) або послідовність символів повернення каретки й переведення рядка CHR(13) + CHR(10), причому в змінну ці роздільники не вставляються.

2) **Input# номер_файлу %, список змінних**

де список змінних – записані через кому змінні будь-якого типу.

Спочатку зчитується цілий рядок, а потім підрядки, розділені роздільниками-комами (відповідні значення полів запису), містяться у відповідні змінні списку. Для коректної роботи оператора рядки файлу повинні мати задану структуру з роздільниками. Якщо текст поля взятий у лапки, ці лапки при читанні файлу відкидаються, відкидається також кома, що розділяє поля. Змінні, використовувані в операторі, можуть бути як рядкового типу, так і числового.

*Для того щоб прочитати всі дані з файлу за допомогою операторів **Input#** або **Line Input#,** необхідно організовувати циклічне зчитування даних з файлу, оскільки дані зчитуються цими операторами по рядках*.

3) **Input (*n* ,# *номер_файла* %)**

де n – число символів (байтів) для зчитування.

Функція **Input** вимагає знання кількості зчитуваних символів. Тому для читання даних з файлу необхідно попередньо обчислити його довжину за допомогою функції **FileLen**.

Перехід на задану позицію у файлі можна організувати за допомогою оператора **Seek.**

Seek ,# номер_файлу%, position

де # номер_файлу% — номер файлу, аналогічний номеру файлу в операторі Open;

position — цілочисельний вираз, що задає позицію покажчика у файлі.

Позиціонування при цьому виконується посимвольно. Оператор Seek встановлює покажчик на необхідну позицію. Якщо після цього використовувати функції зчитування або запису, то дія цих функцій буде виконуватися, починаючи з позиції покажчика, найденого оператором Seek.

Приклади:

Do Until EOF(1)
Input #1, Fam, Im, GodLoop
Close #1

EOF (номер_файлу*)* — повертає "ІСТИНА" при досягненні кінця файлу

Do Until EOF(1)
Line Input #1, text$Loop
Close #1

Input (LOF(1),#1)
Close #1

Закриття файлів послідовного доступу. Для цього необхідно використовувати оператор **Close,** що має синтаксис:

CLOSE [**#номер_файла**] [**,#номер_файла…**]

Якщо вказується номер файлу, то буде закритий саме цей файл.

Оператор **CLOSE** без параметра закриває всі файли, відкриті в цей момент у програмі.

6.3 Робота з файлами довільного (Random) доступу

Файл із довільним доступом має заздалегідь задану структуру й складається із записів. Кожний *запис у файлі* — це деяка порція даних, що має строго певний розмір і свій конкретний номер у файлі. Доступ до даних у файлі довільного доступу здійснюється саме за номером запису. Дані з файлу такого типу читаються й записуються записами.

Прикладами файлу довільного доступу є бази даних, що завжди мають строго певну структуру.

При відкритті файлів довільного доступу можливий тільки один режим доступу — **Random —** *цей режим є режимом за замовчуванням для функції Open*. Використовуючи можливості VB, можна створити файл довільного доступу користувальницької, тобто

своєї власної структури. Продемонструємо це на невеликому прикладі. За допомогою оператора **Type** оголосимо тип змінної, що має задану структуру запису:

```
Type PhisFace
    PhisFaseID As Integer
    FIO As String * 50
End Type
```

У даному прикладі оголошена структура у вигляді запису із двох полів. Першим полем є ідентифікатор, а другим — прізвище, ім'я та по батькові.

*Оголошення користувальницького типу даних необхідно здійснювати в програмному модулі. Для додавання програмного модуля в проект варто виконати команду **Add Module** з меню **Project** і на вкладці **New** діалогового вікна, що з'явилося, **Add Module** вибрати значок з назвою **Module**.*

Відкриття файлу довільного доступу. Файл довільного доступу відкривається трохи інакше, ніж файл послідовного доступу. Синтаксис оператора **Open** при цьому виглядає в такий спосіб:

Для того, щоб використовувати файл із довільним доступом, необхідно *відкрити файл* оператором OPEN

Open *pathName* [For Random] [*Access - доступ*][*Lock*] *As_ #номер_файла Len = довжина запису*]

- *pathName* — ім'я файлу;
- *Access* — доступ задає права доступу до файлу;
- *Lock* (блокування) — визначає права доступу до відкритого файлу користувача/процесу при колективному використанні:
  - *Shared* — всі процеси можуть писати у файл і зчитувати з файлу;
  - *Lock Read Write* — заборона запису у файл і зчитування з файлу;
  - *Lock Read* — заборона зчитування з файлу;
  - *Lock Write* — заборона запису у файл;
- *As # номер_файлу* — номер файлу;
- *Len* визначає довжину запису.

При використанні оператора **Open** для відкриття файлу довільного доступу атрибут **For** не обов'язковий, тому що в VB цей параметр установлюється за замовчуванням. Як видно із синтаксису, на відміну від файлу з послідовним доступом, при відкритті файлу з довільним доступом необхідно обов'язково вказувати довжину запису. При цьому, якщо довжина запису не відома, її можна обчислити.

Запис у файл довільного доступу. Для запису даних у файл довільного доступу використовується оператор Put #, що має синтаксис:

PUT #*номер_файлу* % [, *номер запису*][, *змінна*]

- *#номер_файлу* % — номер відкритого файлу;
- *номер запису* – цілочисельний вираз, що задає номер запису в файлі;
- *змінна* — містить дані для запису в файл.

Якщо номер запису не зазначений, то за замовчуванням приймається поточна позиція покажчика запису.

Зміна даних у файлі довільного доступу. Для зміни даних у записах файлу (редагування, додавання, видалення записів) застосовується оператор **Put #.** При його використанні необхідно мати на увазі, що дані в запису будуть замінені на ті, які ми передаємо у файл. Підкреслимо, що новий запис із даними не створюється.

Для додавання записів у файл необхідно вказувати номер запису на одиницю, більший за номер останнього запису. У цьому випадку запис буде доданий у файл, а не змінений.

Put #FileNum, LastRecord + 1, ForFileRecords

Для обчисленя поточного номера останнього запису можна використовувати довжину запису та розмір файла.

Читання даних з файлу довільного доступу. Дані з файлу довільного доступу, як правило, зчитуються записами. Для цього використовується оператор **Get#**, що має синтаксис:

GET #*номер_файлу% [, номер запису][,змінна]*

- *#номер_файлу* % — номер файлу;
- *номер запису* — номер запису для читання;
- *змінна* — містить дані для прийому введення з файла.

Якщо параметр *номер запису* у функції Get# не зазначений, зчитується поточний запис, на який позиціонований покажчик.

Для позиціонування покажчика можна використовувати функцію Seek. Синтаксис цього оператора такий же, як для файлів послідовного доступу, але має інший зміст. Якщо для послідовних файлів позиціонування виконується за символами, то для файлів довільного доступу — за номером запису:

Seek #*fileNumber, position*

- *fileNumber* — номер файлу;
- *position* — целочисельний вираз, що задає номер запису у файлі.

Видалення даних з файлу довільного доступу. Існують два способи.

Можна просто очистити відповідні поля зазначених записів, тобто записати в них порожні значення. Однак у цьому випадку у файлі залишаються порожні записи. Зрозуміло, що при такому підході ресурси (дисковий простір) використовуються нераціонально. Для остаточного видалення записів рекомендується перезаписувати дані в новий файл, пропускаючи порожні записи. Алгоритм цих дій:

a) створіть новий файл за допомогою оператора Open;

b) перепишіть всі непусті записи в новий файл, використовуючи оператор Put #;

c) закрийте вихідний файл і видаліть його за допомогою оператора Kill;

d) перейменуйте новий файл у вихідний оператором Name.

Одержуємо той же самий файл, але вже без порожніх записів, при цьому заощаджується простір диска й час пошуку даних у такому файлі.

6.4 Організація друку

Друк тексту, тобто виведення даних на принтер, можна організувати за допомогою файлового оператора **Print #.** Дійсно, з погляду виведення даних у принципі однаково, куди пересилати інформацію. Необхідно тільки правильно вказати її одержувача. Для пересилання даних на принтер використовується пряме призначення порту принтера (**LPT1, LPT2**) як одержувач даних. Це можна зробити за допомогою оператора відкриття файлу **Open**:

Open "LPT 1" For Output As #*nPrinterHandle*

Після виконання цього оператора для адресації даних на принтер необхідно використовувати дескриптор (ідентифікатор файлу) ***#nPrinterHandle***. Якщо в додатка не підтримується обчислення ідентифікатора файлів, то варто використовувати функцію **FreeFile** для його обчислення. Аналогічно можна направити дані в будь-який інший порт комп'ютера або мережі. Після того, як порт відкритий для прийому даних, можна використовувати оператор **Print #.** Такий вираз посилає на принтер дані для друку:

Print #*nPrinterHandle*, *strExpression*

де ***strExpression*** задає текст, що друкується.

Використовуючи функції й оператори читання даних з файлу, можна організувати циклічне порядкове виведення файлу на друк. Однак цей метод організації виведення даних на друк має свої тонкощі.

Всі операції роботи принтера (позиціонування друкуючої голівки, переведення рядка, переведення сторінки й ін.) тепер

буде потрібно програмувати за допомогою спеціальних операторів, які розуміє принтер. Такі оператори називаються ***Esc-Кодами*** *(Esc-послідовностями). Їхній опис додається до кожного принтера й тут ми не будемо їх розглядати.*

Після завершення друку даних порт закривається оператором **Close**, що також використовується для закриття файлу: **Close *#nPrinterHandle*** або **Close.** При використанні оператора **Close** без дескриптора одночасно з портом закриваються й файли, з яких виводилася інформація.

7 ВИКОРИСТАННЯ ГРАФІКИ

Графіка в додатках застосовується для того, щоб зробити інтерфейс користувача більш досконалим, інтуїтивно зрозумілим і, звичайно, гарним. Якість оформлення інтерфейсу має велике значення для комерційного успіху додатка, і графіка відіграє в цьому не останню роль. Тому необхідно заздалегідь уточнити, що графіку ми будемо розглядати не як засіб створення мультфільмів або відеокліпів, а як засіб поліпшення й прикрашання інтерфейсу.

За допомогою графіки в VB можна виконувати такі основні дії:

- групувати й виділяти інформацію у формах, використовуючи лінії, рамки, фрейми;
- доповнювати інформацію графічними зображеннями;
- поліпшувати інтерфейс, доповнювати його анімацією.

Використовуючи графіку в додатках, необхідно зберігати почуття міри. Графіка повинна лише доповнювати й підкреслювати інформацію у формах додатка, але не перевантажувати форми й не відволікати від їхнього інформаційного змісту.

Для роботи із графікою й зображеннями в VB використовуються графічні об'єкти й графічні методи. Існують такі можливості:

- додавання й робота із зображенням у формі;
- використання елемента управління **Picture**;
- використання елемента управління **Image**.

7.1 Прості елементи управління для роботи із графікою

На панелі елементів управління є кнопки, що дозволяють створювати у формі прості елементи графіки.

Таблиця 7.1 — Кнопки простих елементів графіки

Кнопка	Назва	Призначення
	Line (Лінія)	Створює у формі лінію
	Shape (Контур)	Створює у формі контури
	Frame (Рамка)	Розміщає у формі рамку, що дозволяє об'єднати об'єкти в групу

Для розміщення у формі графічних зображень використовуються об'єкти **Image** і **Picture**. Їм відповідають такі значки на панелі елементів управління:

Таблиця 7.2 — Значки об'єктів Image і Picture на панелі елементів управління

Значок	Назва	Створюваний об'єкт
	Image	Image
	PictureBox	Picture

Властивість Picture

Щоб розмістити у формі графічне зображення, не обов'язково вибирати графічні об'єкти. Можна просто скористатися властивістю **Picture** форми. Для вставлення графічного зображення у форму необхідно натиснути кнопку із трьома точками у властивості **Picture**, а потім у діалоговому вікні, що відкрилося, **Load Picture** (рисунок 7.1) вибрати файл необхідного зображення.

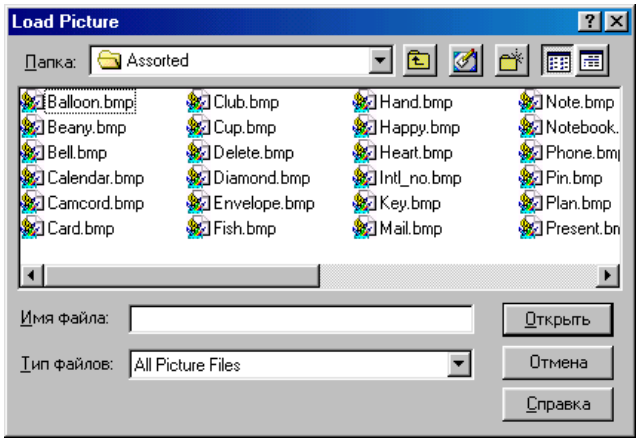

Рисунок 7.1 — Діалогове вікно **Load Picture** пошуку зображення для властивості Picture

Дану властивість форми використовують для розміщення фонового зображення. При цьому необхідно врахувати, що розмір форми повинен відповідати розміру вихідного зображення. Якщо зображення менше форми, то воно розміщається в лівому верхньому куті форми й надалі не переміщується. Тому такий спосіб хоча й простий, але дуже незручний. Властивість **Picture** доступна й у режимі виконання додатка. Простим присвоєнням можна замінити або усунути зображення з форми.

Завдання зображень. Як уже говорилося, визначити графічне зображення в додатка можна як у режимі проектування (режим **Design Time**), так і в режимі виконання (режим **Run Time**). Щоб задати графічне зображення в режимі проектування, використовується властивість **Picture** об'єкта. Для цього необхідно виконати такі дії:

а) відкрити вікно **Properties** об'єкта;

б) вибрати властивість **Picture**;

в) натиснути кнопку із трьома точками в правому стовпці властивості;

г) у діалоговому вікні **LoadPicture**, що відкрилося, знайти необхідний файл;

д) двічі клацнути на файлі мишею або встановити на нього курсор і натиснути кнопку **Відкрити**.

У режимі виконання програми існує більше можливостей для визначення зображення об'єкта.

Зображення можна задати такими способами:

- завантажити зображення за допомогою функції **LoadPicture** із вказівкою файлу зображення. При цьому зображення поміщається у властивість **Picture** об'єкта:

Object.Picture = LoadPicture("C:\...\Картинка.bmp)

- завантажити зображення з ресурсного файлу додатка за допомогою функції **LoadResPicture** із вказівкою ідентифікатора ресурсу і його типу:

Set Object.Picture = LoadResPicture(Resident, vbResBitMap)

- скопіювати зображення з одного об'єкта в інший за допомогою звичайного присвоєння:

ObjectTo.Picture = ObjectFrom.Picture

- скопіювати зображення з об'єкта **ClipBoard**.

Видалення зображень. Для видалення зображення необхідно скористатися завантаженням у властивість **Picture** порожнього значення:

Set Об'єкт.Picture = LoadPicture("")

При цьому зображення видаляється з об'єкта.

Графічні методи. На додаток до графічних елементів управління в VB існує набір графічних методів (таблиця 7.3):

Таблиця 7.3 — Графічні методи VB

Метод	Призначення
Circle	Рисує коло, дугу або еліпс
CIs	Очищає область виведення
Line	Рисує лінії
PaintPicture	Рисує (завантажує) зображення
Point	Повертає кольор точки
Print	Виводить текст
Pset	Розміщає крапку

*Необхідно відзначити, що метод **Print** призначений для виведення тексту й до графіки має непряме відношення. Проте, ми розглянемо його разом із графічними методами, оскільки він використовується разом з іншими методами графіки для "рисування" тексту в зазначеному об'єкті або сполучення тексту й чистої графіки*.

У графічних методах використовуються функції обчислення кольору **RGB** і **QBColor**.

Функції кольору. Функція **RGB** повертає числове подання кольору (тип значення, що повертається, при цьому **long**) відповідно до інтенсивності трьох кольорних складових: **R** (Red — червоний), **G** (Green — зелений) і **B** (Blue — синій). Інтенсивність цих трьох основних кольорів задається шкалою чисел від 0 до 255.

Синтаксис цієї функції:

RGB (Red, Green, Blue)

- **Red** — інтенсивність червоного кольору від 0 до 255;
- **Green** — інтенсивність зеленого кольору від 0 до 255;
- **Blue** — інтенсивність синього кольору від 0 до 255.

Для прикладу в таблиці 7.4 показані часто використовувані кольори, розкладені за інтенсивностями RGB- кольорів.

Таблиця 7.4 — Кольори в RGB-поданні

Кольор	**Інтенсивність червоного**	**Інтенсивність зеленого**	**Інтенсивність синього**
Чорний	0	0	0
Білий	255	255	255
Червоний	255	0	0
Зелений	0	255	0
Синій	0	0	255
Бірюзовий	0	255	255
Пурпурний	255	0	255
Жовтий	255	255	0
Сірий	192	192	192

Приклад: якщо ви хочете зробити фон форми бірюзовим, необхідно присвоїти властивості **BackColor** форми таке значення:

Form.BackColor = RGB (0,255,255).

Для зручності роботи зі стандартними кольорами використовується функція **QBColor**, що повертає числове подання одного з 16 колорів, показаних у таблиці 7.5.

Таблиця 7.5 — Кольори, що повертаються функцією QBColor

Номер кольору	**Кольор**
0	Чорний
1	Синій
2	Зелений
3	Бірюзовий
4	Червоний
5	Пурпурний
6	Жовтий
7	Білий
8	Сірий
9	Світло -синій (блакитний)
10	Світло-зелений (салатний)
11	Світло-бірюзовий
12	Світло -червоний (алий)
13	Світло-пурпурний
14	Світло-жовтий
15	Яскраво-білий

Синтаксис цієї функції:

QBColor (NumberColor),

де **NumberColor** — числове значення (номер кольору), що визначає кольор.

Для того щоб установити для форми бірюзовий кольор фону, досить написати код

Form.BackColor = QBColor(3).

Метод ***Circle.*** Метод **Circle** використовується в такий спосіб:

object.Circle [**Step**] **(x, y), radius,** [**color, start, end, aspect**]

- **object** — об'єкт, у якому застосовується метод. Якщо об'єкт не зазначений, то за замовчуванням використовується форма;
- **step** — ключ, що визначає прив'язку центра кола, дуги або еліпса до координат, що повертається властивостями **currentx** і **currenty** об'єкту;

- **x, y** — числа, що визначають координати центра кола, дуги або еліпса в одиницях властивості **ScaleMode** об'єкта;
- **radius** — число, що визначає радіус кола, дуги або еліпса в одиницях властивості **ScaleMode** об'єкта;
- **color** — значення типу **Long**, що задає кольор лінії. Якщо параметр не зазначений, застосовується значення властивості **ForeColor**. Для задання кольору можна використовувати функції **RGB** () або **QBColor**;
- **start, end** — при рисуванні дуги або частини еліпса задає позицію початку й кінця дуги в радіанах: від 2 π до -2 π радіан;
- **aspect** — задає коефіцієнт "еліптичності" кола. За замовчуванням цей коефіцієнт дорівнює 1.0, що відповідає кола.

*Метод **Cls.*** За допомогою методу **Cls** можна очистити форму або об'єкт **Picture** від тексту й графіки, створених у ньому програмно. Синтаксис методу:

object.Cls

- **object** — об'єкт, у якому застосовується метод. Якщо параметр не зазначений, то за замовчуванням використовується форма.

Необхідно пам'ятати, що при застосуванні цього методу властивості CurrentX і CurrentY обраного об'єкта встановлюються в 0.

*Метод **Line.*** Метод Line призначений для рисування ліній і має такий синтаксис:

object.Line [Step] (x1, y1) [Step] - (x2, y2), [color], [B] [F]

- **object** — об'єкт, у якому застосовується метод. Якщо об'єкт не зазначений, то за замовчуванням використовується форма;
- **Step** — ключ, що визначає прив'язку початку лінії до координат, що повертається властивостями Currentx і Currenty об'єкта;
- **x1, y1** — координати початку лінії. При відсутності цих параметрів початок прив’язується до значень властивостей Currentx і Currenty об'єкта;
- **Step** — ключ, що визначає прив'язку координат кінця лінії до початку, тобто координати кінця задані щодо координат початку лінії;
- **x2, y2** — координати кінця лінії;
- **color** — задає кольор лінії в палітрі **RGB**. Якщо параметр не зазначений, то використовується властивість **ForeColor** об'єкта;

- **B** — задає рисування прямокутника, при цьому координати означають координати лівого верхнього і правого нижнього кутів;
- **F** — задає заповнення прямокутника кольором лінії рисування.

*Метод **PaintPicture.*** Метод **PaintPicture** рисує (завантажує) вміст графічних файлів, що мають розширення **bmp, wmf, emf, cur, ico або dib** у формах або об'єктах типу **picture**.
Метод має такий синтаксис:

object.PaintPicture picture, xl, yl, width1, height1, x2, y2, width2, height2, opcode

- **object** — об'єкт, у якому застосовується метод. Якщо об'єкт не зазначений, то за замовчуванням використовується форма;
- **picture** — вихідне зображення, що буде розміщено в об'єкті. Це повинно бути посилання на властивість **picture** даного або іншого об'єкта;
- **xl, yl** — координати лівого верхнього кута області об'єкта для розміщення вихідного зображення;
- **width1, heightl** — розмір (ширина і висота) області розміщення вихідного зображення. Якщо розмір області відрізняється від розміру вихідного зображення, то зображення пропорційно розтягується або стискується;
- **x2, y2** — координати лівого верхнього кута у вихідному зображенні для вставки в об'єкт. Якщо задані координати, відмінні від нуля, то буде вставлена частина вихідного зображення;
- **width2, height2** — розмір (ширина й висота) частини вихідного зображення, що вставляється;
- **opcode** — встановлює режим вставки зображення за допомогою констант із набору **RasterOp** (таблиці 7.6). Задається тільки для роботи із зображеннями, що мають розширення **bmp**.

Для роботи методу із зображеннями, що мають розширення **bmp**, необхідно використовувати константи з набору **RasterOp** для установлення режиму вставки зображення. Ці константи описані в таблиці 7.6. За допомогою від’ємних значень ширини (width1) і висоти (heightl) можна перевернути зображення вертикально або горизонтально.

Таблиця 7.6 — Набір констант RasterOp для методу PaintPicture

Константа	Значення	Опис
vbDstInvert	&H00550009	Інвертує зазначене зображення
vbMergeCopy	&HOOCOOOCA	Поєднує рисунок і зображення-джерело
vbMe де Paint	&HOOBB0226	Поєднує інвертоване зображення джерела із зазначеним зображенням, використовуючи оператор **Or**
vbNotSrcCopy	&H00330008	Копіює інвертоване зображення-джерело в зазначене зображення
vbNotSrcEraae	&H001100A6	Інвертує результат об'єднання зазначеного зображення й зображення-джерела, використовуючи оператор **Or**
vbPatCopy	&HOOF00021L	Копіює рисунок у зазначене зображення
vbPatInvert	&H005A0049L	Поєднує зазначене зображення з рисунком, використовуючи оператор **Хог**
vbPatPaint	&HOOFBOA09L	Поєднує інвертоване зображення джерела з рисунком, використовуючи оператор **Or.** Поєднує результуючий вираз в цій операції із зазначеним зображенням за допомогою оператора **Or**
vbSrcAnd	&H008800C6	Поєднує пикселі зазначеного зображення джерела, використовуючи оператор **And**
vbSrcCopy	&HOOCC0020	Копіює зображення-джерело в зазначене зображення
vbSrcErase	&H00440328	Інвертує зазначене зображення й поєднує результат із зображенням-джерелом за допомогою оператора **And**
vbSrcInvert	&H00660046	Поєднує пикселі зазначеного зображення й зображення-джерела, використовуючи оператор **Хог**
vbSrcPaint	&HOOEE0086	Поєднує пикселі зазначеного зображення й зображення-джерела за допомогою оператора **Or**

Метод ***Point.*** Метод **Point** повертає кольор у палітрі **RGB** зазначеної точки у формі або на об'єктах типу **PictureBox**.
Синтаксис методу:

object.Point (x, y)

- **object** — об'єкт, у якому використовується метод;
- **x, y** — координати точки в об'єкті.

Необхідно мати на увазі, що у випадку "випадання" точки із границь форми або об'єкта ***Picture*** *(тобто коли координати більше розміру об'єкта) метод повертає від'ємне значення -1.*

Метод ***Print.*** Метод **Print** виводить (друкує) текст у зазначений об'єкт або вікно **Immediate**. Синтаксис методу:

object.Print [outputList]

- **object** — об'єкт, у якому використовується метод. Якщо об'єкт не зазначений, то за замовчуванням використовується форма;
- **outputList** — рядковий вираз або список виразів, виведених в об'єкті.

Список виведення **outputList** має певний синтаксис. Цей список задається в такий спосіб:

{Spc(n) / Tab(n)} expression charpos

- **Spc(n)** — вставляє у виведений текст кількість пробілів, що задається числом **n**;
- **Tab(n)** — задає позицію точки початку вставки;
- **expression** — числовий або строковий вираз для виведення;
- **charpos** — задає позицію вставки.

Отже, при використанні синтаксису методу **Print** не вказуються шрифт, його кольор, координати виведення тексту. Ці параметри виведення тексту визначаються такими властивостями об'єкта, у який виводиться текст:

- **CurrentX** —координата початку тексту по горизонтальній осі;
- **CurrentY** — координата початку тексту по вертикальній осі;
- **Font** — шрифт і розмір виведеного тексту;
- **FontTransparent** — прозорість тексту, тобто властивість, при якому крізь текст видний фон об'єкта;
- **ForeColo**r — кольор.

Метод ***Pset.*** Метод **Pset** призначає кольор зазначеної точки об'єкта. Цей метод є зворотним методу **Point**. Синтаксис методу:

object.Pset [Step] (x, y), [color]

- **object** — об'єкт, у якому використовується метод. Якщо об'єкт не зазначений, то за замовчуванням використовується форма;
- **Step** — ключ, що визначає прив'язку координат точки до координат, що повертається властивостями **CurrentX** і **CurrentY** об'єкта;
- **x, y** — координати точки на об'єкті;
- **color** — кольор точки в палітрі **RGB**. Якщо параметр не зазначений, то використовується властивість **ForeColo**r об'єкта.

При використанні методу **Pset** необхідно мати на увазі, що розмір точки визначається товщиною об'єкта, тобто властивістю об'єкта **DrawWidth**. Для товщини 1 це один піксел, для товщини більше одиниці— це область об'єкта із центром, що має зазначені координати. Спосіб рисування точки при цьому визначається властивостями об'єкта **DrawMode** і **DrowStyle**.

Робота зі шрифтами. Шрифт, використовуваний в об'єкті, визначається властивістю об'єкта **Font**. У свою чергу, властивість **Font** також є об'єктом зі своїм набором властивостей. Діалогове вікно призначення цих властивостей показане на рисунок 7.2. У діалоговому вікні **Font** (Шрифт) можна призначити такі властивості шрифту:

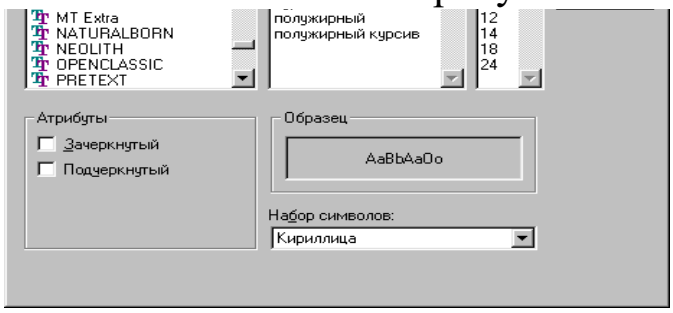

Рисунок 7.2 — Діалогове вікно установлення властивостей шрифту

- **Bold** — призначає напівжирний шрифт (bold);
- **italic** — установлює курсив (*italic*);
- **Strikethrough** — застосовує ефект перекреслення (~~strikethrough~~);
- **Underline** — установлює підкреслення (underline);
- **Name** — вибирає найменування шрифту (рядковий ідентифікатор шрифту, наприклад Arial);
- **size** — призначає розмір шрифту.

Для установлення найменування шрифту використовується список **Шрифт** діалогового вікна. Щоб установити накреслення (напівжирний, курсив, звичайний), використовується список **Накреслення**. Розмір шрифту можна встановити в поле **Розмір**

вручну або вибрати значення зі списку розмірів. Для установлення атрибутів закреслений і підкреслений, необхідно використовувати прапорці **Закреслений** і **Підкреслений**. Таблиця кодування набору символів призначається в списку, що розкривається, **Набір символів**.

Управління кольором. Управління кольором форми й елементів управління можна організувати, використовуючи властивості **ForeColor** і **BackColor**. Властивість **ForeCoior** задає кольор тексту, а **BackColor** установлює кольор фону.

У режимі проектування додатка ці властивості можна встановити у вікні **Properties** об'єкта (рисунок 7.3), викликавши діалогове вікно настроювання кольору. Для цього необхідно виділити властивість, що набудовується, і натиснути на кнопку зі стрілкою в правому стовпці властивості. Як видно з рисунка, це діалогове вікно складається із двох вкладок: **Palette** (Палітра) і **System** (Системні). На вкладці **Palette** можна встановити довільні кольори з палітри, на вкладці **System** можна вибрати кольор зі списку кольорної схеми Windows, що встановлюється в панелі управління Windows викликом вікна настроювання **Властивості**: **Display**. При використанні кольорної схеми необхідно мати на увазі, що при зміні настроювання кольорної схеми Windows відповідно зміняться й кольори додатка.

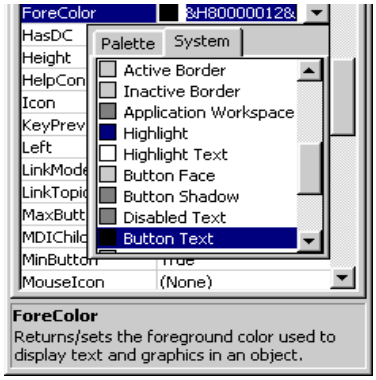

Рисунок 7.3 — Діалогове вікно настроювання кольорів об'єкта

У режимі виконання додатка властивості **ForeColor** і **BackColor** можна встановити простим присвоєнням, використовуючи функції кольору або вбудовані константи VB, подані в таблиці 7.7.

Таблиця 7.7 — Набір констант VB для управління кольором

Константа	Значення	Опис кольору
vbBlack	&HO	Чорний
vbRed	&HFF	Червоний
vbGreen	&HFFOO	Зелений
vbYellow	&HFFFF	Жовтий
vbBlue	&HFFOOOO	Блакитний
vbMagenta	&HFFOOFF	Пурпурний
vbCyan	&HFFFFOO	Бірюзовий
vbWhite	&HFFFFFF	Білий

7.2 Анімаційна графіка

Анімаційну графіку будемо розглядати як засіб для внесення різноманітності й привабливості в користувальницький інтерфейс. Прикладом такої анімації може служити процес копіювання файлів у системі Windows, коли на екрані відображається жива картинка перенесення сторінок з папки в папку. Не займаючи великих ресурсів, анімація в такому виконанні робить інтерфейс додатка більш привабливим, а в остаточному підсумку більше конкурентоспроможним і сам додаток. Ми розглянемо деякі варіанти простої анімації, а саме, зміну й переміщення зображень. У прикладах додатків використовується масив елементів управління, тому попередньо вивніжо, що являє собою такий масив, і як його створити при проектуванні додатка.

Масив елементів управління. Масив елементів управління являє собою групу елементів управління одного типу, які ідентифікуються за тим самим ім'ям й індексом. Індекс діє аналогічно звичайному масиву, як, наприклад, у масиві чисел. Кожний з елементів масиву характеризується властивістю **Name**, що має те саме значення для всіх елементів. Всі вони відносяться до одного типу. Ідентифікуються елементи такого масиву за допомогою властивості **Index**. Масив елементів управління створюється при проектуванні додатка. Для створення масиву існують такі основні способи:

- створений елемент управління дублюється за допомогою команд **Copy** і **Paste** меню **Edit**;
- створюються елементи управління одного типу окремо, а потім їм присвоюється те саме ім'я у властивості **Name** і відповідне значення індексу у властивості **index**.

*В обох випадках властивість **index** підтримується автоматично при позитивній відповіді на питання про додавання елемента управління в масив.*

Перемикання зображень. Найпростіший спосіб анімації — перемикання зображень. Розглянемо це на прикладі. Створіть невеликий додаток, виконавши такі дії:

1) створіть новий стандартний проект;

2) присвойте проекту ім'я ***MyGraphics.*** Для цього відкрийте вікно властивостей проекту, вибравши команду **Project1 Properties** меню **Project** . Після перейменування проекту ця команда буде називатися **MyGraphics Properties**.

3) присвойте формі проекту ім'я ***FormForGraphics***. У властивість **caption** форми введіть заголовок ***Форма для роботи із графікою***;

4) додайте у форму кнопку управління типу **commandBution**, двічі клацнувши мишею кнопку **CommandButton** на панелі елементів управління. Назвіть цю кнопку **cbcontrol** і присвойте властивості **Caption** значення **Змінити стан**. Створена у формі кнопка **cbcontrol** буде служити для перемикання зображень, виконуваних додатком **MyGraphics** по події **click** цієї кнопки;

5) додайте на форму три елементи управління типу **picture** і назвіть їх **picRed**, **picYellow** і **picGreen**. Вставте у властивість **Picture** кожного об'єкта відповідним колорам зображення світлофора з каталогу ***\Common\Graphics\Icons\Trafric***. Властивість **visible** для всіх трьох елементів управління встановіть в стан **False**;

6) додайте на форму об'єкт типу **image**, у якому буде перемикатися зображення, і назвіть його **imgStatus**, при цьому властивість **picture** цього об'єкта залишіть порожньою. Властивість **visible** для даного елемента управління встановіть в стан **True**. Після виконання наведених вище кроків форма додатка буде мати вигляд, показаний на рисунку 7.4;

7) відкрийте вікно редактора й задайте код:

```
Dim fIgEnd As Integer
Private Sub Form_Load()
fIgEnd = 0
imgStatus.Picture = picRed.Picture
FormForGraphics-Caption =
= "Світлофор"
End Sub
Private Sub cbControl_Click ()
If imgStatus.Picture = picRed.Picture
Then
imgStatus.Picture = picYellow.Picture
Elself imgStatus.Picture = picYellow.Picture Then
imgStatus.Picture = picGreen.Picture
Else
imgStatus.Picture = picGreen.Picture
fIgEnd = 1
End If
If imgStatus.Picture = picGreen.Picture And fIgEnd = 1 Then
imgStatus.Picture = picRed.Picture
fIgEnd = 0
End If
End Sub
```

Рисунок 7.4 — Вигляд додатка

10) запустіть додаток на виконання. Натисканням кнопки **Змінити стан** здійснюється послідовне перемикання стану світлофора за допомогою присвоєння властивості **picture** об'єкту **imgStatus** нового значення.

У цьому додатка показана можливість перемикання зображень із набору окремих об'єктів, у яких ці зображення містяться. Однак для зберігання необхідних зображень краще використовувати масиви елементів управління або об'єкта **imageList**.

Переміщення й перемикання зображень. Переміщаючи й перемикаючи зображення, можна організувати нескладну анімацію. Найчастіше цей спосіб використовується для анімації процесів очікування або розрахунку, "пожвавлення" зображень у формах додатка. Для вивчення переміщення й перемикання зображень створимо невеликий проект **MyGraphicsDay**. Виконайте такі дії:

1) створіть новий стандартний проект;

2) присвойте проекту ім'я **MyGraphicsDay**. Відкрийте вікно властивостей проекту, вибравши команду **Project1 Properties**

меню **Project**. Після перейменування проекту ця команда буде називатися **MyGraphicsDay Properties**;

3) присвойте формі проекту ім'я **FormForGraphics**. У властивість **caption** форми введіть заголовок ***Форма для роботи із графікою***;

4) додайте на форму кнопку управління типу **commandButton**. Назвіть цю кнопку **cbRun** і присвойте властивості **Caption** значення **Старт**. Створена у формі кнопка **cbRun** буде служити для запуску дій, виконуваних додатком **MyGraphicsDay** по події **click** цієї кнопки;

5) додайте на форму **FomForGraphic**s ще одну таку ж кнопку й назвіть її **cbStop.** Присвойте властивості **caption** значення **Стоп**. Ця кнопка буде служити для останову анімації й повернення додатка у вихідний стан;

6) при анімації, навіть найпростішій, потрібно організувати зміну об'єктів у часі. Для задання відліку інтервалів часу служить об'єкт типу таймер (**Timer**). Додайте його у форму за допомогою кнопки **Timer** на панелі елементів управління. Цей об'єкт буде виробляти управляючі повідомлення для анімації. Використовуючи властивість **Name**, присвойте таймеру ім'я **tmrGraphicsTimer**. При запуску додатка таймер не баніжо, тому про властивість **visible** для цього об'єкта можна не замислюватися. Властивість таймера **Enabled** встановіть в стан **False**, а властивості **interval** присвойте значення 200.

*Коротко нагадаємо наступне. Таймер відраховує інтервали часу в мілісекундах, відпрацьовуючи подію **Timer**. При цьому мінімальний інтервал відліку часу становить 1, а максимальний — 65 535 мілісекунд, тобто максимально таймер може працювати з інтервалом ледве більше 1 хвилини. Основні властивості об'єкта Timer — **Interval** і **Enabled**. Властивість **Interval** визначає інтервал відпрацьовування події **Timer**, властивість **Enabled** запускає або зупиняє таймер. Якщо **Enabled** установлено в стан **True**, таймер починає відпрацьовувати подію **Timer**, якщо в стан **False** — таймер перебуває в стані очікування. При цьому таймер із установленою при проектуванні властивістю **Enabled** у стан **True** запускається* відразу ж після завантаження форми, у якій він перебуває;

7) додайте у форму елемент управління **image**. Назвіть його **imgSunMoon**;

10) додайте на форму масив елементів управління типу **Picture**, що складається із двох елементів. Для створення масиву скористайтеся будь-яким зручним для вас способом. Наприклад, розмістіть у формі об'єкт **picture**, двічі клацнувши мишею кнопку **PictureBox** на панелі елементів управління. Потім скопіюйте об'єкт і вставте його копію у форму. Об'єктам масиву присвойте ім'я **picSunMoon.** В елемент масиву, індекс якого дорівнює 0 (властивість **index**), введіть зображення сонця. В елемент масиву, індекс якого дорівнює 1, уведіть зображення місяця. Властивість **visible** для елементів масиву встановіть в стан **False**. Отриманий додаток показаний на рисунок 7.5;

11) відкрийте вікно редактора й розмістіть в ньому зазначений нижче код:

```
Dim xAdd As Integer
Dim yAdd As Integer
Dim figSunMoon As Integer
Private Sub Form Load()
' "Зміна дня й ночі"
xAdd = 100
yAdd = -100
figSunMoon = 1
xScaleSunMoon = ImgSunMoon.Width
FormForGraphics.Caption = "Зміна дня й ночі"
End Sub
Private Sub cbRun_Click()
tmrGraphicsTimer.Enabled = True
xAdd = 100
yAdd = -100
figSunMoon = 1
ImgSunMoon.Picture =
= PicSunMoon(0).Picture
End Sub
Private Sub cbStop_Click()
tmrGraphicsTimer.Enabled = False
ImgSunMoon.Left = xScaleSunMoon / 2
ImgSunMoon.Top = FormForGraphics.ScaleHeight - 500
ImgSunMoon.Picture = PicSunMoon(0).Picture
End Sub
Private Sub tmrGraphicsTimer Timer()
ImgSunMoon.Left = ImgSunMoon.Left + xAdd
```

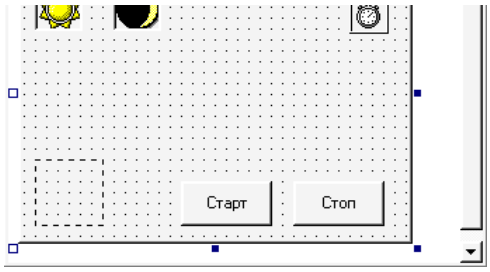

Рисунок 7.5 — Вигляд додатка MyGraphicsDay

```
ImgSunMoon.Top = ImgSunMoon.Top + yAdd
If ImgSunMoon.Left > FormForGraphics.ScaleWidth / 2 - xScaleSunMoon /
2 And figSunMoon = 1 Then
yAdd = 100
End If
If ImgSunMoon.Left < FormForGraphics.ScaleWidth / 2 -xScaleSunMoon /
2 And figSunMoon = 0 Then
yAdd = 100
End If
If ImgSunMoon.Left > FormForGraphics.ScaleWidth Then
yAdd = -100
xAdd = -100
figSunMoon = 0
ImgSunMoon.Picture = PicSunMoon(1).Picture
ImgSunMoon.Left = FormForGraphics.ScaleWidth - 500
ImgSunMoon.Top = FormForGraphics.ScaleHeight - 500
End If
If ImgSunMoon.Left < xScaleSunMoon / 2 And figSunMoon = 0 Then
ImgSunMoon.Left = xScaleSunMoon / 2
ImgSunMoon.Top = FormForGraphics.ScaleHeight - 500
tmrGraphicsTimer.Enabled = False
End If
End Sub
```

12) запустіть додаток на виконання. При натисканні кнопки **Старт** об'єкт, що нагадує сонце, переміщається вздовж форми й при досягненні правого нижнього кута міняє зображення, після чого повертається у вихідну точку (рисунок 7.6). Натискання кнопки **Стоп** переводить додаток у вихідний стан.

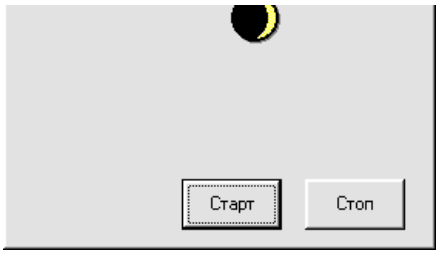

Рисунок 7.6 — Вигляд додатка MyGraphicsDay при запуску

Сполучення зображення й тексту. Іноді виникає необхідність сполучити текст і зображення в об'єктах додатка. Для цього використовується метод **print**, що дозволяє виводити текст на об'єкті. Розглянемо, як цей метод застосовується на практиці. Для створення додатка виконайте такі дії:

1) створіть новий стандартний проект;

2) присвойте проекту ім'я **MyGraphObjects**. Для цього відкрийте вікно властивостей проекту, вибравши команду

Project1Properties з меню **Project** . Після перейменування проекту ця команда буде виглядати як **MyGraphObjects Properties**;

3) присвойте формі проекту ім'я **FormForGraphics**. У властивість **caption** форми введіть заголовок **Графічні форми й методи**;

4) додайте у форму кнопку управління типу **conmandButton**, двічі клацнувши мишею кнопку **CommandButton** на панелі елементів управління. Назвіть цю кнопку **cbstart** і присвойте властивості **caption** значення **Виконати**. Створена у формі кнопка **cbstart** буде служити для відображення тексту на графічному об'єкті по події **click** цієї кнопки;

5) перейменуйте форму проекту **Form1** в **FormForGraphics** і у властивість **Caption** введіть заголовок вікна **Графічні форми й методи**;

6) додайте у форму елемент управління типу **picture**. Назвіть цей об'єкт **picpicture**. У властивість picture об'єкта вставте графічне зображення світлофора з каталогу ***\Common\Graphics\ Icons\Traffic;***

7) введіть у вікно редактора код, зазначений нижче:

```
Private Sub cbStart_Click()
picPicture.Print "Світлофор"
 End Sub
```

10) запустіть додаток на виконання. Натисніть кнопку **Виконати**. На зображенні з'явиться напис "**Світлофор**" (рисунок 7.7).

Рисунок 7.7 — Додаток зі сполученням зображення й тексту в режимі виконання

Рисунок 7.8 — Додаток із призначенням координат для тексту

Можна помістити напис під рисунком, вказавши координати точки початку напису на зображенні рисунок 7.8). У цьому випадку введіть у вікно редактора такий код:

```
Private Sub cbStart_Click()
picPicture.Current = 100
picPicture.Current = 400
picPicture.Print "Світлофор"
End Sub
```

Елемент управління Animation. Елемент управління **Animation** дозволяє використовувати при анімації відеофайли — спеціально підготовлені файли з розширенням **avi**, що являють собою послідовний набір кадрів (растрових зображень). Для використання елемента управління **Animation** необхідно підключити до проекту бібліотеку Microsoft Windows Common Control-2 6.0. Після підключення цієї бібліотеки до проекту на панелі елементів управління з'являється кнопка **Animation**. Вид елемента управління **Animation**, розміщеного у формі, показаний на рисунок 7.8.

Треба ще раз звернути увагу на те, що для застосування елемента управління ***Animation*** *потрібен відеофайл. У своєму прикладі ми скористаємося відеофайлом з каталогу* ***\Common\ Graphics\Videos.***

Основні методи елемента управління **Animation**, які дозволяють організувати перегляд відеофайлів, що випливають (таблиця 7.8):

Таблиця 7.8 — Основні методи елемента управління Animation

Метод	Призначення
Open	Відкриває відеофайл
Play	Запускає відеофайл на виконання
Stop	Зупиняє відеофайл
Close	Закриває відеофайл

Метод **Play** може мати додаткові аргументи:

Repeat — задає кількість повторів перегляду відеофайлу;

Start — задає початковий кадр;

Stop — задає останній кадр.

Для вивчення елемента управління **Animation** створимо додаток **MyVideoAnimation.** Виконайте такі дії:

1) створіть новий стандартний проект;

2) присвойте проекту ім'я **MyVideoAnimation**. Для цього відкрийте вікно властивостей проекту, вибравши команду

Project1 Properties меню **Project**. Після перейменування проекту ця команда буде йменуватися **MyVideoAnimation Properties**;

3) присвойте формі проекту ім'я **FormVideoAnimation**. У властивість **Caption** форми введіть заголовок **Перегляд відео файлів**;

4) підключіть до проекту дві бібліотеки: Microsoft Common Dialog Control 6.0 і Microsoft Windows Common Control-2 6.0. Для цього в меню **Project** виберіть команду **Components** і в діалоговому вікні, що відкрилося, встановіть відповідним іменам бібліотек прапорці.

Бібліотеку Microsoft Common Dialog Control 6.0 підключаємо для створення у формі діалогового вікна пошуку відеофайлів на диску;

5) додайте у форму кнопку управління типу **CommandButton.** Назвіть цю кнопку **cbRun** і присвойте властивості **caption** значення **Перегляд відеофайлу.** Створена у формі кнопка **cbRun** призначена для запуску відеофайлу на перегляд по події **click** цієї кнопки;

6) додайте у форму **FormForGraphics** ще одну таку ж кнопку й назвіть її **cbStop**. Присвойте властивості **Caption** значення **Стоп**. Ця кнопка буде служити для останову перегляду відеофайлу;

7) додайте у форму елемент управління **Animation**, двічі клацнувши мишею кнопку Animation на панелі елементів управління. Назвіть його **anivideo**;

10) додайте у форму елемент управління **CommonDialog**, двічі клацнувши мишею кнопку **Common Dialog** на панелі елементів управління. Назвіть створений об'єкт **cdvideo.** Вигляд створеного проекту показаний на рисуноку 7.9.

11) відкрийте вікно редактора й введіть код:

```
Private Sub cbRun_Click()
cdvideo.Filter = "avi files (*.avi):*.avi"
cdVideo.ShowOpen
anivideo.Open cdVideo.FileName
anivideo.Play
End Sub

Private Sub cbStop_Click()
anivideo.Stop
End Sub

Private Sub Form Unload(Cancel As Integer)
anivideo.Close
End Sub
```

12)запустіть додаток на виконання. При натисканні кнопки **Перегляд відеофайлу** відкривається діалогове вікно **Відкриття файлу**, що дозволяє знайти відеофайл. Знайдіть необхідний файл і натисніть кнопку **Відкрити**. Відеофайл запускається на перегляд (рисунок 7.10). Зупинити перегляд можна натисканням кнопки **Стоп**.

*Для діалогового вікна **Відкриття файлу** встановлена властивість **Filter**, щоб здійснити фільтрацію файлів з розширенням **avi**.*

Якщо метод **play** у цьому прикладі використовувати в такому вигляді:

aniVideo.Play 3, 5, 10

то три рази будуть прокручені кадри з 5 по 10.

Рисунок 7.9 — Вигляд елемента управління **Animation** при проектуванні додатка

Рисунок 7.10 — Перегляд відеофайлу у формі

СПИСОК ЛІТЕРАТУРИ

1 Браун, С. Visual Basic 6 [Текст]: учебный курс / С. Браун. — СПб: Питер, 2006. — 574 с.

2 Бутенко, В.М. Основи програмування мовами високого рівня [Текст] : навч. посібник / В.М. Бутенко, В.С. Меркулов, О.В. Казанко, О.В. Чаленко — Харків: УкрДАЗТ, 2009. — 206 с.

3 Вирт, Н. Систематическое программирование [Текст] / Н.Вирт. — М.: Мир, 1977. — 183 с.

4 Волченков, Н.Г. Программирование на Visual Basic 6 [Текст]: в 3-х ч. / Н.Г.Волченков.— М.: ИНФРА-М, 2000. — 280 с.

5 Голуб, А.И. Правила программирования С и С++ [Текст] / А.И. Голуб. — М.: БИНОМ, 2001. – 272 с.

6 Грэхем, И. Объектно-ориентированные методы [Текст] : принципы и практика, 3-е изд. / И. Грэхем. — М.: Вильямс, 2004. —880 с.

7 Дал, У. Структурное программирование [Текст] / У.Дал, Э. Дейкстра, К. Хоор. — М.: Мир, 1973. — 247 с.

8 Дейкстра, Э. Дисциплина программирования [Текст] / Э.Дейкстра. — М.: Мир, 1978. — 275 с.

9 Зелковиц, М. Принципы разработки программного обеспечения [Текст] / М. Зелковиц, А. Шоу, Дж. Гэннон.— М.: Мир, 1982. — 368 с.

10 Иванова, Г.С. Основы программирования [Текст] : учебник для вузов / Г.С.Иванова. — М.: Изд-во МГТУ им. Н.Э.Баумана, 2002. — 416 с.

11 Иванова, Г.С. Технология программирования [Текст] : учеб. для вузов / Г.С.Иванова. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. — 320 с.

12 Информатика и ИКТ. Задачник-практикум [Текст] : в 2 т. / под ред. И. Г. Семакина, Е.К. Хеннера. — М.: Лаборатория базовых знаний, 2000.

13 Інформатика, комп’ютерна техніка, комп’ютерні технології [Текст] : посібник / за ред. О.І. Пушкаря. — К.: Академія, 2001.

14 Йодан, Э. Структурное проектирование и конструирование программ [Текст] / Э. Йодан. — М.: Мир, 1979. — 415 с.

15 Керниган, Б. Практика программирования [Текст] / Б. Керниган, Р. Пайк. — СПб.: Невский диалект, 2001. — 381 с.

16 Максимов, Н.А. Азбука программирования на Visual Basic [Текст]: практикум / Н.А.Максимов. — Чебоксары, 2007. — 64 с.

17 Меркулов, В.С. Основи алгоритмізації базових обчислювальних процесів [Текст]: навч. посібник / В.С. Меркулов. В.О. Гончаров, І.Г. Бізюк, В.М. Бутенко, О.В. Головко . — Харків: УкрДАЗТ, 2008. — 163 с.

18 Райтингер, М. Visual Basic 6.0 [Текст] / М. Райтингер, Г. Муч — К.: BHV, 2000. — 288 с.

19 Семакин, И.Г. Информатика [Текст] : структурированный конспект базового курса / И.Г. Семакин, Г.С. Вараксин. - М.: Лаборатория базовых знаний, 2001.

20 Синтес, А. Освой самостоятельно объектно-ориентированное программирование за 21 день [Текст] /А.Синтес. — М.: «Вильямс», 2002. — 672 с.

21 Тассел, Д. В. Стиль, разработка, эффективность, отладка и испытание программ [Текст]: пер. с англ. — 2-е изд., испр. / Д.В. Тассел. — М.: Мир, 1985.—332 с.

22 Філіппенко, І.Г. Програмування інженерно-технічних задач в середовищі QBasic [Текст] : конспект лекцій. - Ч.3. / І.Г. Філіппенко, В.О. Гончаров, В.С. Меркулов. — Харків: УкрДАЗТ,
2007. — 134 с.

23 [http://www.alexsoft.ru Visual Basic 6.0.]

24 [http://www.ctc.msiu.ru/materials/Book/node82.html Основные концепции ООП]

25 [http://www.weblibrary.biz/c-sharp/princypy