ФАКУЛЬТЕТ ІНОФОРМАЦІЙНО-КЕРУЮЧИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Кафедра обчислювальної техніки та систем управління

ОСНОВИ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Конспект лекцій

Частина 1

Харків – 2018

Основи об’єктно-орієнтованого програмування: Конспект лекцій / С. Є. Бантюков, Р. О. Яровий, В. С. Меркулов, І. Г. Бізюк, С. О. Бантюкова. – Харків : УкрДУЗТ, 2018. – Ч. 1. – 58 с.

Конспект лекцій розроблено відповідно до робочих програм дисциплін ″Обчислювальна техніка та програмування ″ і ″Комп’ютерна техніка та програмування″ спеціальності ″Залізничний транспорт″.

Метою лекцій є опанування студентами алгоритмічної мови високого рівня, набуття необхідних знань для організації обчислень у середовищі VB, використання теоретичних засад об’єктно-орієнтованого підходу до розроблення програмних проектів.

Рекомендується для студентів спеціальності ″Залізничний транспорт″ усіх форм навчання.

Іл. 5, табл. 8, бібліогр.: 9 назв.

Конспект лекцій розглянуто та рекомендовано до друку на засіданні кафедри обчислювальної техніки та систем управління 26 лютого 2018 р., протокол № 7.

Рецензент

проф. О. В. Устенко

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Конспект лекцій

Частина 1

Відповідальний за випуск Бізюк І. Г.

Редактор Третьякова К. А.

Підписано до друку 05.04.18 р.

Формат паперу 60х84 1/16. Папір писальний.

Умовн.-друк.арк. 2,0. Тираж 50. Замовлення №

Видавець та виготовлювач Український державний університет залізничного транспорту,
61050, Харків-50, майдан Фейєрбаха, 7.
Свідоцтво суб’єкта видавничої справи ДК № 6100 від 21.03.2018 р.

ЗМІСТ

ЗАГАЛЬНІ ПОНЯТТЯ І ВИЗНАЧЕННЯ

Програма (Program) — комбінація комп'ютерних інструкцій і даних, що дозволяє апаратному забезпеченню обчислювальної системи виконувати обчислення або функції управління.

Додаток (Application) — прикладний комп'ютерний сервіс, який має набір певних функцій і є одним із компонентів програмного забезпечення. Простіше кажучи, це програма, яка виконує деякі дії, щоб полегшити життя користувачам або вирішити ту чи іншу проблему.

Терміни ″програма″ і ″додаток″ часто використовуються як синоніми. Однак термін ″додаток″ найкраще підходить до Windows-програми, оскільки Windows-програми зазвичай складаються із декількох файлів. Ці файли працюють разом у формі проекту.

Проект (Project) — набір створюваних файлів, які становлять Windows-додаток. **Visual Basic** (далі — **VB),** подібно до інших сучасних засобів розробки додатків, є візуальною системою. Додаток не програмується, а проектується. Розробник проектує (малює) різні об'єкти, пов'язуючи їх при необхідності кодом.

Проект фактично є обліковим записом всіх складових додатка. При створенні будь-якого, навіть найпростішого додатка, VB створює проект так, щоб із розширенням функціональних можливостей цього додатка проект міг відстежувати всі файли, пов'язані з ним. Проект описує два важливих аспекти VB-додатка: зовнішні імена файлів (на магнітному диску) і внутрішні імена, які використовуються усередині проекту.

Проект розглядають з точки зору внутрішньої його структури, а додаток — з точки зору його функціонування.

Можна сказати, коли мова йде про розроблення або модифікацію додатка, про нього говорять як про проект, розглядаючи всі його компоненти. Коли говорять про програму — йдеться про закінчений додаток, і в деталі його структури не вникають, розглядаючи його як єдине ціле. Іншими словами, проект — це додаток у стадії проектування або коригування.

Об'єкт (Object) — практично все, з чим доводиться мати справу в VB (й у Windows в цілому), і що є об'єктом проектування

у VB-формі (вікна, командні кнопки, лінії, команди меню, прапорці, перемикачі, бази даних, принтери і т.п.). Опис всього, що відбувається на екрані і ″всередині″ самого середовища здійснюється з точки зору поведінки об'єктів. Кожен об'єкт характеризується набором **властивостей, методів** і вироблюваних **подій.**

Властивість (Property) об'єкта визначає його зовнішній вигляд і поведінку. Наприклад, властивість **Caption** визначає текст напису на об'єкті.

Подія (Event) пов'язана з певними діями користувача і може викликати код VB — процедуру обробки події.

Метод (Method) — будь-яка дія, яку можна виконати над даним об'єктом для зміни тієї чи іншої властивості.

Продемонструвати ці поняття можна на прикладі умовного об'єкта ″телефон″. Дзвінок телефону — це подія. Ми реагуємо на нього, піднімаючи трубку (обробка події). Щоб зателефонувати кому-небудь, ми застосовуємо метод ″набрати″. А властивості визначають зовнішній вигляд апарата, наприклад — колір.

Клас (Class). Об'єкти об'єднуються у класи. До одного класу належать об'єкти з однаковим набором властивостей, подій і методів. Наприклад, всі, створені в додатку прапорці відносяться до одного і того ж класу, незважаючи на те, що значення окремих властивостей (наприклад, підписи) можуть відрізнятися. Головне, що вони мають однаковий набір властивостей, подій і методів. Командна кнопка не може бути віднесена до класу прапорців, тому що набір її властивостей, подій і методів відрізняється від відповідного набору для класу прапорців.

Всі об'єкти, що використовуються у VB можна розділити на кілька рівнів[1, 9].

1. Системний об'єкт (System object) — це об'єкт, що безпосередньо визначається VB і знаходиться за межами області видимості додатка. Зумовленими (системними) об'єктами VB є:

Application — описує поточний додаток;

Clipboard — буфер обміну;

Debug — об'єкт налагодження, є вікном **Immediate**;

Printer — системний принтер;

Screen — екран користувача.

2. Форма (Form) — основний об'єкт додатка. Являє собою вікно, в якому запущено програму, що виводить повідомлення, або виконуються інші дії. Розміри вікна, як правило, можна змінювати. Його можна переміщати по екрану. За допомогою форм здійснюється ″спілкування″ додатка з користувачем. У формі розташовуються всі інші об'єкти, тобто форма служить **контейнером** для інших об'єктів. Додаток може містити кілька різних форм, однак не менше однієї. Для кожної форми створюється свій файл форми.

3. Елемент управління (Control). Так у VB називають об'єкти, що розташовуються у формі або іншому контейнері. На відміну від системних, що є зумовленими, тобто які не потребують опису, елементи управління необхідно створити (описати), помістити їх у форму. Доки елемент управління не буде поміщений у форму, він просто не існує як об'єкт. Працюючи з елементами управління, користувач впливає на них, ініціюючи певні події, і в результаті управляє додатком. Звідси і термін — ″елемент управління″.

Проектування програми починається з його збирання з елементів управління. Зазвичай при цьому використовується **панель елементів управління (Toolbox)**. Це набір прототипів, на основі яких будуються конкретні об'єкти, що мають індивідуальні значення властивостей. Деякі з них вже мають якісь значення, як, наприклад, колір, форма, інші — ні.

Кожному прототипу в панелі елементів управління відповідає свій клас. Всі об'єкти, що створюються за допомогою одного і того ж прототипу, відносяться до одного і того ж класу.

Нижче наведені деякі найбільш прості елементи управління.

Командна кнопка (Command Button) — виклик коду для виконання будь-яких дій.

Напис (Label) — відображення на екрані постійного тексту.

Текстове поле (TextBox) — поле для введення даних.

Прапорець (CheckBox) — установлення або зняття опції.

Перемикач (OptionButton) — вибір однієї позиції у групі.

Список (ListBox) — вибір зі списку.

Поле зі списком (ComboBox) — комбінація текстового поля і списку.

Основна подія для цих елементів — **Click** — швидке натискання на елемент управління, або **Change** — зміна вмісту текстового поля. Для ″пожвавлення″ додатка з елементами управління слід пов'язати код.

Код (Code) — набір команд, який вказує як маніпулювати даними. За допомогою коду зв'язуються елементи управління, виконуються обчислення, маніпулювання даними.

Код підрозділяється на **модулі (Modules)** або **блоки (Blocs)** — секції коду, що є єдиним цілим.

За способом використання у VB розрізняють два види кодів: код, який управляє і реагує на елементи управління форми (**код обробки подій**), і **код загального призначення**, не пов'язаний безпосередньо з елементами управління і формою, що виконує функції, загальні для декількох елементів управління.

За способом зберігання також розрізняють два види кодів. Кожна форма містить власний код — **модуль форми (Form module),** який містить коди обробки подій для форми її елементів управління. У модуль форми може бути включений і код загального призначення, доступний, проте тільки в межах форми. Модуль форми зберігається у файлі форми. Кожна програма має не менше ніж одну форму. Тому воно містить принаймні один модуль форми. При додаванні у проект додатка форми разом з цією формою додається і новий модуль форми.

Компонент проекту типу **Modules** є окремим файлом, в якому зберігаються коди загального призначення, доступні з будь-якого місця проекту. Такий файл називається **стандартним модулем (Standards Module)** або **модулем загального призначення** і знаходиться поза файлом форми.

Для виклику кодів, звернення до об'єктів, їх властивостей і методів, а також маніпулювання даними використовуються **ідентифікатори (імена),** що складаються з літер, цифр і деяких спеціальних символів. Деякі імена (наприклад, імена властивостей, подій, методів) у VB є зумовленими, деякі (імена об'єктів, даних і т.п.) визначає розробник проекту на свій розсуд, а деякі (імена процедур обробки подій) будуються за певними правилами (див. п. 2.1).

Компілятор (Compiler) — система, яка перетворює програму (код) на мові VB у виконувані комп'ютером додатки.

Середовище розробника VB (Developer Studio) — сукупність всіх методів, правил та угод системи VB, являє собою робоче місце розробника.

1 Об’єктно-орієнтоване програмування (ООП)

1.1 Об’єктно-орієнтовані технології

Вирішування будь-якого завдання вимагає певного рівня абстракції [3]. Мови програмування високого рівня дають змогу піднятись на більш високий рівень абстракції порівняно з машинними кодами, які є основою мови асемблер. Ця можливість пов’язана з тим, що всі мови програмування високого рівня дозволяють, крім використання стандартних типів даних, створювати свої власні типи даних, які б найбільш повно відбивали характер задачі. Крім того, принцип модульності дозволяє програмну реалізацію задачі формулювати в термінах незалежних модулів, які використовують певну структуру даних задачі.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм або як перелік інструкцій комп'ютеру, об’єктно-орієнтовані програми можна вважати сукупністю об'єктів, кожний з яких здатний отримувати повідомлення, обробляти дані та надсилати повідомлення іншим об'єктам.

Принципи структурного програмування є основою для організації, з одного боку, оптимальної структури програми, а з іншого боку, для конструювання необхідної структури даних для збереження та обробки інформації у програмі. Проте дані і модулі, що їх обробляють, співіснують у структурних програмах окремо. Точніше кажучи, дані відіграють доволі пасивну роль – вони дозволяють модулям їх обробляти, будучи не в змозі захистити себе від некоректного використання.

Сучасна концепція ООП (Object Oriented Programming) визначається як технологія створення складного програмного забезпечення на основі подання програми у вигляді сукупності об’єктів, кожний з яких є екземпляром певного типу (класу), а класи утворюють ієрархію із успадкуванням властивостей.

Таким чином, кожний об’єкт може містити не лише дані, які характеризують його стан, а й методи їх обробки (функції), що демонструють поведінку об’єкта. Об’єктно-орієнтований підхід заснований на тому, що будь-який елемент оточуючого нас світу можна вважати об’єктом.

Предметну область будь-якої задачі програмування будемо вважати поданою у вигляді сукупності функціональних елементів (об’єктів), що обмінюються у процесі виконання програми повідомленнями. Такий процес називається об’єктною декомпозицією. При цьому кожний об’єкт предметної області відповідає за виконання деяких дій залежно від одержаних повідомлень і параметрів самого об’єкта.

Сукупність значень параметрів об’єкта визначає його стан або властивості, а сукупність реакцій на одержані повідомлення – поведінку або методи (які задаються як функції об’єкта). Причому властивості і методи не існують окремо, а об’єднані й утворюють єдиний об’єкт. Це принципова відміна від процедурного підходу, в якому властивості (тобто дані) існують окремо від процедур, що їх обробляють.

1.2 Основні концепції ООП

Визначимо основні концепції ООП:

- система складається із об'єктів;
- об'єкти взаємодіють між собою;
- кожний об'єкт характеризується станом і поведінкою;
- стан об'єкта задається значенням полів даних;
- поведінка об'єкта задається методами.

Детальніше розглянемо ключові принципи об’єктно-орієнтованого підходу:

1. Будь-що є об’єктом. Об’єкт можна преподавати як своєрідну змінну: він містить дані, але водночас здатний виконувати певні дії над ними. Теоретично, будь-який елемент предметної області задачі може бути поданий у програмі як об’єкт.

2. Програма – це комплекс об’єктів, які обмінюються повідомленнями. Щоб попросити об’єкт щось зробити, йому

треба надіслати повідомлення. Більш конкретно повідомлення можна подавати як виклик функції, яка є методом деякого об’єкта.

3. Можна створювати нові типи об’єктів, використовуючи вже наявні. Ця можливість реалізується завдяки механізму успадкування.

4. Кожний об’єкт має певний тип. Тип об’єкта визначається повідомленнями, які йому можна надсилати.

5. Різні об’єкти певного типу можуть отримувати однакові повідомлення, реагуючи на них по-різному.

1.3 Фундаментальні поняття ООП

Клас визначає абстрактні характеристики деякої сутності, включаючи характеристики самої сутності (атрибути або властивості) та дії, які вона здатна виконувати (поведінку, методи або можливості).

Клас – це тип, що описує устрій об'єктів – екземплярів. Класи вносять модульність та структурованість в об'єктно-орієнтовану програму. Властивості та методи класу разом називаються його **членами.**

Клас являє собою об'єктний тип даних, зовні схожий на тип даних "структура" у процедурно-орієнтованій мові QBasic. При цьому елементи такої "структури" (члени класу) можуть самі бути не тільки даними, але й "методами" (тобто процедурами або функціями).

Об'єкт. **Загальна характеристика**: об’єкт має стан (сукупність значень атрибутів), поведінку й особистість. При цьому мається на увазі, що об’єкт має внутрішні дані (стан), методи (які визначають поведінку) і кожний об’єкт відрізняється від будь-якого іншого (принаймні тим, що має унікальну адресу в пам’яті).

Об’єкт ООП – це сукупність змінних стану і пов’язаних з ними методів. Ці методи визначають, як об’єкт взаємодіє із зовнішнім світом.

Взаємодія об’єктів. Об’єкти в програмі не існують самі по собі. Програмування у термінах об’єктів має зміст лише тоді, коли можливо організувати їх взаємодію. Одним із базових принципів ООП є такий: об’єкти можуть взаємодіяти один з одним, надсилаючи повідомлення з проханням виконати деякий

належний метод або виконуючи метод у відповідь на запит іншого об’єкта. Взаємодіючи, об’єкти утворюють програму.

Об’єкт не може прямо змінити стан іншого об’єкта, він може тільки попросити його виконати деякі дії, надсилаючи повідомлення.

Будь-яке *повідомлення* складається із трьох частин:

1) *імені об’єкта*, якому воно адресоване;

2) *імені методу*, який повинен виконати об’єкт-адресат;

3) *параметрів*, необхідних для виконання методу.

1.4 Принципи ООП

Визначимо основні принципи ООП [5]:

I – Абстракція даних.

II – Інкапсуляція.

III – Ієрархія.

IV – Поліморфізм.

I. Абстракція даних – виділення істотних характеристик деякого об'єкта, що відрізняють його від всіх інших видів об'єктів.

Об'єкти являють собою спрощений, ідеалізований опис реальних сутностей предметної області.

Вибір правильного набору абстракцій для заданої предметної області – головне завдання ООП.

Ідея про об’єкти, які, хоч і є унікальними, належать до певного класу і мають спільні характеристики і поведінку, використовується в усіх об’єктно-орієнтованих мовах. Фундаментальним поняттям ООП є **абстрактні типи даних** (або класи).

За означенням, ***абстрактний тип даних*** – це група пов’язаних між собою даних і методів (функцій), які можуть здійснювати операції над цими даними.

Абстрактні типи даних дуже схожі на вбудовані типи: можна створювати змінні відповідного типу (які в ООП називаються об’єктами, а, наприклад, в мові VB – екземплярами класу) і виконувати якісь дії з цими змінними (наприклад, надсилати повідомлення, при цьому об’єкт одержує повідомлення і сам вирішує, що з ним робити).

Відмінність ООП від традиційного підходу полягає у тому, що тут програміст може визначати клас (тип даних), який є зручним для вирішення конкретного завдання, а не змушений використовувати для цього лише існуючі типи даних. Можна в рамках ООП розширювати мову програмування шляхом створення нових типів даних, необхідних для вирішення конкретного завдання. Програмна система надає цим новим типам, або класам, таку саму підтримку, яка існує для вбудованих типів.

Тип об'єкту – сукупність методів, які підтримує об'єкт, тобто інтерфейс. Інтерфейс визначає те, що може робити даний об'єкт. Проте десь повинен існувати код, в якому запрограмовані відповідні дії. Цей код, разом із прихованими даними об'єкта, складає його реалізацію.

Це просто зрозуміти за аналогією із процедурним програмуванням. Для кожного типу визначені функції, які реалізують усі можливі дії із змінними цього типу. В разі потреби виконати певну дію викликається відповідна функція. Іншими словами, в термінах ООП: об'єкту надсилається запит (повідомлення), а об'єкт вирішує як його обробити (виконує код).

II. Інкапсуляція – приховування інформації:

- відділення одне від одного елементів об'єкта, що визначають його призначення і поведінку;
- приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення.

Інкапсуляція – це механізм, який об'єднує дані і методи, що маніпулюють цими даними, і захищає те й інше від зовнішнього втручання або невірного використання. Коли методи і дані об'єднуються у такий спосіб, утворюється об'єкт.

Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Здійснюється об'єднання даних і процедур їхньої обробки в рамках однієї синтаксичної структури мови програмування.

Подібно до того як у процедурному програмуванні глобальна функція не має доступу до локальних змінних іншої функції, так і дані всередині об'єкта приховані від інших частин програми. Для маніпулювання даними об'єкта інші частини програми повинні попросити об'єкт викликати його власні

інкапсульовані методи, які і повертають інформацію про стан об’єкта (або даних) модулю, що послав запит. Реалізація класу об’єкта може бути змінена непомітно для решти програми, якщо незмінним залишається інтерфейс класу.

Часто члени класу позначаються як публічні (**public**), захищені (**protected**) та приватні (**private**), визначаючи, чи доступні вони всім класам, підкласам або лише класу в якому їх визначено.

III. Ієрархія – упорядкування абстракцій, розташування їх за рівнями.

Види ієрархій – **агрегація, спадкування.**

• Новий клас може бути створений із довільної кількості об’єктів інших класів, у будь-якому їх поєднанні.

Після того як клас створено і відтестовано, він може бути використаний неодноразово. Найпростішим способом повторно використати клас є створення об’єкта (екземпляра) даного класу. Але і сам клас може стати цеглиною при побудові нового класу. Останній може бути створений з довільної кількості об’єктів інших класів, у будь-якому їх поєднанні. Цей процес називається **агрегацією** (або композицією).

• Інший спосіб утворення нового класу – це механізм **успадкування**.

Успадкування – це процес, шляхом якого деякий клас може успадковувати властивості та методи деякого існуючого класу, додаючи до них деякі особисті риси або здатність об'єкта зберігати атрибути класу чи батьківського об’єкта.

Існує можливість породжувати один клас від іншого зі збереженням всіх властивостей і методів класу-предка і додаючи, при необхідності, нові властивості й методи.

Успадкування використовується у випадку, коли треба створити клас із функціональністю, схожою на функціональність вже існуючого класу. При успадкуванні утворюється клон існуючого (батьківського, базового) класу, який є коренем ієрархії успадкування, і цей клон (нащадок) відповідним чином модифікується. Ці два класи можуть мати спільні характеристики і поведінку, але один з них (нащадок) може мати більше властивостей та обробляти більше повідомлень (або обробляти їх інакше).

В одного батьківського класу може бути декілька нащадків, при цьому батьківський (базовий) клас містить всі властивості і методи, спільні для його нащадків.

Набір класів, пов'язаних відношенням успадкування, називають **ієрархією**.

IV. Поліморфізм – здатність об'єкта набувати різних форм.

Можна сказати, що ***поліморфізм*** – це властивість різних об’єктів по-різному відповідати на однакові повідомлення, тобто одне й те саме ім’я може використовуватись для методів різних класів.

Похідний об'єкт успадковує методи і властивості об'єкта-батька. ***Поліморфізм*** дозволяє додавати, видозмінювати, і навіть видаляти деякі особливості поведінки похідного об'єкта.

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто два або більше класів можуть реагувати по-різному на однакові повідомлення.

Основна кількість властивостей і методів об'єктів притаманні всім екземплярам класу, але деякі можуть бути змінені при необхідності, щоб характеризувати тільки даний екземпляр.

Стосовно програмування ***поліморфізм*** – явище, при якому той самий програмний код виконується по-різному, залежно від того, об'єкт якого класу використовується при виклику даного коду.

Основною концепцією ***поліморфізму*** є ідея про те, що один інтерфейс дозволяє реалізувати багато методів.

При цьому вибір конкретного методу залежно від ситуації здійснює компілятор. Механізм цього вибору такий: під час виклику певного методу класу спочатку шукається такий метод у самому класі. Якщо його знайдено – він виконується, якщо ж ні – відбувається звертання до класу, який є базовим (батьківським) для заданого класу, і пошук методу відбувається у ньому. Цей процес повторюється до тих пір, поки не знайдено корінь (верхній клас) ієрархії успадкування.

Це забезпечує збереження незмінним інтерфейсу класу-предка та дозволяє встановити зв'язок імені методу в коді з різними класами: з об'єкта якого класу здійснюється виклик, з того класу й береться метод із даним іменем.

Такий механізм називається ***динамічним*** (або пізнім) ***зв'язуванням,*** на відміну від ***статичного*** (раннього), здійснюваного на етапі компіляції програми.

2 Основні конструкції VB

Код VB складається із одного або декількох операторів, які VB перетворює у команди, зрозумілі комп'ютеру [2, 4].

Щоб VB розумів вихідний код, слід дотримуватися певних правил його написання.

У кожному рядку коду поміщається оператор, який може мати додаткові параметри. Логічний рядок (тобто оператор) можна розділяти на кілька фізичних рядків. Ознакою поділу служить символ підкреслення, наступний після пробілу.

В одному рядку можна також об'єднувати кілька операторів, розділяючи їх двокрапкою. Одна таке об'єднання є сенс робити тільки для дуже простих операторів, інакше програмний код буде нечитабельним.

У код можна вставляти **коментарі,** призначені для пояснення окремих фрагментів коду. При виконанні коду коментарі ігноруються. Для виділення коментарів можна використовувати оператор REM або символ апострофа. Їхня дія однакова. Однак REM (як оператор) повинен знаходитися в окремому рядку. Апостроф може ставитися у будь-якому місці. В обох випадках текст поміщається праворуч.

2.1 Підпрограми

У VB, як і в багатьох інших мовах, весь програмний код знаходиться всередині підпрограм [6].

Підпрограма — це послідовність оголошень та інструкцій (операторів), об'єднаних для виконання. Підпрограма має заголовний і кінцевий рядки, між якими міститься код підпрограми.

Є два різновиди підпрограм — процедури і функції.

Процедура починається із заголовка, який складається із ключового слова **Sub,** за яким іде ім'я процедури і в круглих дужках — список формальних параметрів (аргументів) із зазначенням типу кожного параметра (п. 2.2.1):

Sub <ім'я> ([<параметр> As <тип>],...)

Закінчується процедура оператором **End Sub**.

Як випливає із вищесказаного, залежно від призначення можна виділити процедури обробки подій і процедури загального призначення.

Процедура обробки події (Event procedure) призначена для обробки деякої події, пов'язаної з елементом управління (об'єктом). Ім'я такої процедури зумовлено і включає до себе ім'я об'єкта та ім'я події, розділені символом підкреслення (_). Процедура обробки події викликається автоматично при настанні події, з якою вона пов'язана.

Процедура загального призначення не пов'язана з жодною подією і її виклик розробник здійснює на свій розсуд, з інших підпрограм. Ім'я такої процедури може бути довільним і визначається розробником.

Процедура викликається як звичайний оператор VB. При цьому **формальні аргументи** замінюються еквівалентними їм — **фактичними.** Фактичні параметри в дужки не беруться.

При зверненні до процедури (виклику) виконання коду, з якого надійшов виклик (що викликає код), тимчасово припиняється і починають виконуватися оператори процедури. При цьому у всіх операторах процедури замість формальних параметрів процедури використовуються відповідні їм — фактичні. Після завершення процедури виконання викликаючого коду поновлюється із перерваного місця.

Процедуру можна викликати багаторазово з різних місць коду. При цьому можна кожен раз ставити інші значення фактичних параметрів: процедура буде виконувати подібні дії над іншими даними (фактичними параметрами).

Функція побудована так само, як і процедура. Однак є і відмінності. Як і в математиці, результатом роботи функції є значення, що повертається. Тому ім'я функції використовується одночасно і як змінна. Це означає, що змінна з іменем функції містить значення, яке повертається, що слід привласнити у функції. Виклик функції також дещо відрізняється від виклику процедури. По-перше, фактичні аргументи вказуються у дужках. По-друге, повернене значення має бути присвоєно будь-якій змінній або використано у виразі для обчислення значення.

Синтаксис функції **Function:**

Function <ім'я> ([<параметр> As <тип>,...]) As <тип>

<оператори>

End Function

де **Function** — ключове слово;

ім'я — назва (ідентифікатор) функції та значення, яке вона повертає;

параметр — ідентифікатори формальних параметрів (п. 2.2.1), параметри частіше називаються аргументами функцій;

тип — тип формальних параметрів (п. 2.2.1);

Оскільки функція повертає значення, після дужки, що закриває список формальних параметрів, вказується тип значення (тип функції).

2.2 Дані

2.2.1 Види і типи даних

Програма здійснює обробку даних. Дані бувають декількох видів.

Константи — дані, значення яких, будучи визначеними перед виконанням програми, не змінюються в процесі виконання.

Літерали — значення, вказані безпосередньо у коді. Літералами можуть виступати числа, рядки (текстові значення), логічні значення.

Змінні — дані, значення яких можуть змінюватися у процесі виконання програми.

Імена функцій — значення, які повертаються функціями (п. 2.1).

Параметри (аргументи) процедури — дані, що визначаються в іншому коді і передаються процедурі для обробки при зверненні до неї як фактичні параметри, заміщаючи їх формальне визначення (п. 2.1).

Властивості елементів управління також доступні для обробки у VB-програмах. Їх, як і змінні, можна аналізувати або встановлювати, використовувати у виразах.

Кожне дане відноситься до певного типу. Під **типом даного** розуміється безліч його допустимих значень. Тип даного визначає також безліч допустимих над ними дій (таблиця 1).

Таблиця 1 — Типи даних VB

Тип даних	*Об'єм занятої пам'яті, байт*	*Діапазон значень*	*Пре-фікс*	*Су-фікс*	*Приклад*
1	2	3	4	5	6
Цілі числа					
Byte (однобай-тове ціле число)	**1**	**Позитивне число від 0 до 255**	**byt**		**bytImage**
Integer (коротке ціле число)	**2**	**Від -32768 до 32767**	**int**	**%**	**intQuantity**
Long (довге ціле число)	**4**	**Від -2147483648 до 2 147483648**	**lng**	**&**	**ingTotal**
Boolean (логічне значення)	**2**	**True (ІСТИНА) або False (ХИБНІСТЬ)**	**bin**		**binSuccess**
Числа з плаваючою точкою					
Single (речовинне число із плаваючою точкою (нормальне, одинарне))	**4**	**Від'ємні числа: від -3.4E+38 до -1.4E-45** **Додатні числа: від 1.4E-45 до 3.4E+38**	**sng**	**!**	**sngLength**
Double (речовинне число з плаваючої точкою подвійної точності (довге))	**8**	**Від'ємні числа: від -1.8D+308 до -4.9D-324** **Додатні числа: від 4.9D-324 до 1.8D+308**	**dbl**	**#**	**dblSum**

Продовження таблиці 1

1	2	3	4	5	6
String (рядок змінної довжини)	**10 байтів + довжина рядка (1 байт на кожний символ)**	**Від 0 до 2 млрд. символів**	**str**	**$**	**strLastname**
String *довжина (рядок фіксованої (постійної) довжини))	**довжина рядка (1 байт на кожний символ)**	**Від 1 до ~65400**		**$**	
Об'єктні типи					
Object (посилання на об'єкт)	**4**				
Невизначені типи					
Variant (числові типи)	**16**	**Довільне числове значення**	**vnt**		**vntValue**
Variant (символьні типи)	**22 байти + довжина рядка**	**Довільне символьне значення**	**vnt**		
Інші типи					
Currency (грошова величина-число з фіксованою точкою)	**8**	**Ціла частина числа до 15 цифр, дробова – до 4 цифр Від - 922337203685477.5808 до 922337203685477.5807**	**cur**	**@**	**curPrice**
Date (дата/час)	**8**	**Діапазон дат від 1 січня 100 року до 31 грудня 9999 року Діапазон часу**	**dtm**		**dtmFinish**

		від 00:00:00 до 23:59:59			

Значення даних типу String (рядок) беруть в лапки. Самі лапки в значення не входять.

Тип даних Variant встановлює тип даних залежно від вмісту. Змінна типу Variant змінює свій тип під час виконання програми.

Слід мати на увазі, що дані типу Variant через часті перетворення займають більше пам'яті і довше обробляються, ніж аналогічні дані, оголошені із зазначенням типу.

Елементи управління майже завжди представляють тип даних Variant. Отже, коли програма отримує якесь значення від елемента управління, це дане має тип Variant.

2.2.2 Змінні

Змінна (Variable) — це пойменована область пам'яті, призначена для зберігання даних. Таким чином, для доступу до змінної досить знати її ім'я. Тип даних визначає формат і кількість пам'яті.

VB допускає *неявний опис* змінних. Це означає, що змінна може оголошуватися автоматично, як тільки вона зустрічається. При *явному описі* змінна має бути попередньо описана, наприклад, оператором **Dim.** Щоб заборонити неявний опис, слід використовувати опцію **Explicit,** помістивши на початку коду рядок **Option Explicit**.

Після установки цієї опції VB видаватиме помилку компіляції при виявленні в коді неоголошеної змінної.

Для явного опису змінних використовується оператор **Dim,** що має синтаксис:

Dim<ім'я_змінної> [As <тип_даних>]

Приклад.

```
Dim Grup As String
Dim ocenka As Integer
```

Перший оператор визначає змінну **Grup** рядкового типу, а другий **ocenka** — цілого типу.

Область визначення змінної.

Локальна змінна визначається і доступна всередині підпрограми;

Змінна контейнера визначається (Declarations) у секції (General) і доступна всередині відповідного контейнера (форми, модуля, класу).

Глобальна змінна визначається (Declarations) у секції (General) модуля. При цьому замість оператора **Dim** використовується **Public.** Доступна у всіх модулях і процесах проекту.

Час життя змінної. Локально оголошені змінні при виході з підпрограми видаляються із пам'яті, а при новому виклику цієї підпрограми ініціюються заново. Їхній вміст при цьому не зберігається.

VB дає можливість оголосити змінні як статичні. При виході з підпрограми вміст статичної змінної зберігається і може бути використаний при новому вході в неї. Статична змінна зберігається протягом усього часу, поки існує у пам'яті форма або модуль.

Для оголошення змінної як статичної потрібно просто замість оператора **Dim** використовувати оператор **Static:**

Static <ім'я_змінної> [As <тип_даних>]

2.2.3 Масиви

Масив — це набір елементів певного типу, кожен з яких має свій порядковий номер — індекс. Індексів може бути кілька. Розрізняють *статичні* і *динамічні* масиви.

Межі статичного масиву встановлюються на етапі розроблення і можуть змінюватися тільки в новій версії програми.

Динамічні масиви змінюють свої межі в ході виконання програми. З їх допомогою можна динамічно задавати розмір масиву відповідно до конкретних умов.

Для оголошення статичного масиву використовується оператор **Dim** із зазначенням у круглих дужках після імені масиву його максимальних індексів (розмірів):

Dim <ім'я_масиву> (<розмір1,..., розмір k>) As <тип>

Динамічний масив створюється у два етапи. Спочатку він оголошується (Declarations) у секції (General) контейнера (форми,

модуля, класу) без вказівки розміру. Потім за допомогою оператора **ReDim** встановлюється фактичний розмір масиву.

Кількість індексів **k** в описі масиву визначає розмірність масиву: при **k** = 1 масив одновимірний, при **k** = 2 — двовимірний і т.д.

Для звернення до елемента масиву необхідно вказати ім’я масиву та в круглих дужках — індекси елемента. Слід мати на увазі, що у VB індексація починається з нуля. Індексацію з одиниці можна задати за допомогою оператора **Option Base 1**.

Приклад.
Dim ocenka (5) As Integer
Dim M (5,8) As Integer

У першому випадку визначили масив ocenka з шести елементів цілого типу, а в другому — таблицю (двовимірний масив) з шести рядків і дев'яти стовпців.

2.2.4 Властивості об'єктів

Як вже зазначалось, кожен об'єкт має свій набір **властивостей**. VB дозволяє звертатися до властивостей не тільки на етапі проектування програми, але і під час виконання: можна оперативно перевіряти і змінювати окремі властивості.

Головне, що необхідно знати при роботі з властивостями об'єктів, — те, що до них можна звертатися як до змінних, привласнюючи значення певним властивостям і змінюючи їх. Синтаксис звернення до властивості об'єкта має такий вигляд:

[<Ім'я форми>.] <Ім'я елементу управління>. <Властивість>

При зверненні до елементів управління, що належать поточній формі, ім’я форми вказувати необов'язково.

Для більшості об'єктів у VB передбачені стандартні властивості, які використовують у тому випадку, якщо ім’я властивості явно не задано.

Кожна властивість є ніби внутрішньою змінною об'єкта, значення якої можна як встановлювати, так і зчитувати, наприклад:

Form1. Control1. Caption = ″Виконати розрахунок″

Напис = Form1. Control1. Caption

Перший приклад встановлює напис командної кнопки Control1 у формі Form1, а другий зчитує цей напис у змінну Напис.

2.2.5 Константи

Основна відмінність констант від змінних полягає у тому, що їх значення неможливо змінювати у процесі виконання програми.

Область видимості визначається так само, як і для змінних.

Для оголошення константи використовується ключове слово **Const**. Глобальна константа оголошується як **Public**.

[Public|Private]Const<ім'я_константи>[As<тип_константи>]=<значення>

Приклад.

Const Grupa As string = ″5/3-Вс-1″

2.2.6 Літерали

Літерал — це, як і константа, таке значення, яке не змінюється. Однак це певне значення даних. На відміну від константи літерал не має імені і використовує власне значення. Наприклад: 127, 3.25, ″стіл″.

Іменований літерал — це особливе іменоване фіксоване значення. У VB є велика кількість убудованих іменованих літералів, використання яких покращує розуміння тексту.

2.3 Оператор присвоювання

Оператор присвоювання (Assignment statement) використовується для обчислення значення виразу і присвоєння йому змінної або властивості елемента управління. Має формат

<Ім'я> = <вираз>

Вираз формується із змінних, констант, літералів, властивостей об'єктів, функцій (як убудованих, так і визначених розробником) формальних параметрів підпрограм, а також знаків операцій у круглих дужках. Дужки використовують для зміни

пріоритетів операцій. Тип виразу визначається типом даних, які беруть участь у його формуванні.

Арифметичні вирази формуються на основі даних арифметичного типу. Знаки арифметичних операцій: ″+″ (додавання), ″-″ (віднімання), ″*″ (множення), ″/″ (ділення), ″^ ″ (ступень).

Приклад.
Y = X ^ 2 + SIN (x) + SQR (x)

Рядкові вирази утворюються з операторів рядкового типу (String), над якими допускається тільки одна операція — **зчеплення (конкатенація)**, що позначається символом ″+″ або ″&″.

Приклад.
Str = naimf &″факультет″

До змінної naimf (містить назву факультету) додається через пробіл слово ″факультет″ (пробіл входити в літерал ″факультет″).

Логічний вираз (умова) будується з операндів логічного типу та відношень за допомогою логічних операторів **And** (і), **Or** (або), **Not** (не).

Відношення призначено для порівняння двох величин (вони повинні бути порівняних типів). Результат порівняння має логічний тип. Допустимі операції відношення: ″=″ (дорівнює), ″< >″ (не дорівнює), ″>″ (більше), ″<″ (менше), ″> =″ (більше або дорівнює), ″< =″ (менше або дорівнює).

Приклад.
Bool = (A <10) and (A> 0)

Змінна Bool отримує значення TRUE (істина), коли A знаходиться в інтервалі $0 < A < 10$, і FALSE (хибність) в іншому випадку.

2.4 Вбудовані функції

VB надає розробнику велику кількість вбудованих функцій. Для зручності всі функції поділяють на групи, пов'язані не тільки з типом функції, але й із її застосуванням і типом аргументів таблиці 2,3).

Таблиця 2 — Стандартні математичні функції

Запис на VB	***Функція***	***Приклад***
SIN(X)	синус X	**sinX, аргумент у радіанах**
COS(X)	косинус X	**CosX, аргумент у радіанах**
TAN(X)	тангенс X	**tgX, аргумент у радіанах**
ATN(X)	арктангенс X	**ArctgX, -π/2<X<π/2,**
ABS(X)	модуль X	**|X|**
SQR(X)	квадратний корінь X	**√X, X>0**
EXP(X)	експонента X	**eX**
LOG(X)	натуральний логарифм X	**lnX, X>0**
FIX(X)	скорочення до цілого (відкидання дробової частини)	**FIX(-5.6)=-5** **FIX(24.07)=24.00**
INT(X)	знаходження найбільшого цілого, що не перевищує X	**INT(15.5)=15** **INT(-6.2)=-7** **INT(-6.7)=-7**
RND(X)	генерація псевдовипадкових чисел від 0 до 1	
ROUND	округлення до заданого числа десяткових знаків	**Round(3.44,1)=3.4** **Round(3.44)=3.4**
SGN(X)	знак числа X	**SGN(X) = -1, якщо X < 0** **SGN(0) = 0, якщо X = 0** **SGN(X) = +1, якщо X > 0**

Таблиця 3 — Формули для обчислення деяких математичних функцій

Функція	***Розрахункова формула***
Секанс	Sec(x)=1/Cos(x)
Косеканс	Cosec(x)=1/Sin(x)
Котангенс	Cotan(x)=1/Tan(x)
Арксинус	Arcsin(x)=Atn(x/Sqr(-x*x+1))
Арккосинус	Arccos(x)=Atn(-x/Sqr(-x*x+1))+2* Atn(1)
Логарифм на підставі N	LogN(x)=Log(x)/Log(N)

Стандартні функції обробки символьної інформації

Значеннями числових функцій із символьними аргументами є числа, а аргументами — символьні вирази або функції, які повертають символьні значення. Значеннями рядкових функцій є ланцюжки символів (див. таблицю 4,6).

Таблиця 4

Функція	***Призначення***
Asc	Повертає ASCII-Код символу
Chr	Перетворює ASCII-Код у символ
InStr, InStrRev	Здійснюють пошук одного рядка в іншому
LCase	Змінює регістр букв вихідного рядка на нижній
Left	Повертає зазначену кількість символів з початку рядка
Len	Повертає кількість символів у рядку
LTrim, RTrim, Trim	Видаляють пробіли, розташовані відповідно на початку, наприкінці й по обидва боки символьного рядка
Mid	Повертає задану кількість символів з довільного місця рядка
Right	Повертає зазначену кількість символів з кінця рядка
Str, CStr	Перетворюють числовий вираз у рядок
StrReverse	Змінює порядок проходження символів у рядку на зворотний
StrConv	Змінює регістр букв символьного рядка
Val	Перетворюють рядок у числовий вираз
UCase	Змінює регістр букв вихідного рядка на верхній
Space	Повертає рядок, що складається із зазначеної кількості пробілів
Split	Перетворить рядок в одновимірний масив, що нумерується з нуля
Join	Перетворить масив у рядок
String	Повертає рядок, що складається із зазначеної кількості повторень того самого символу
StrComp	Повертає результат порівняння двох рядків
Replase	Знаходить і замінює у рядку підрядок іншим підрядком

Всі ці функції в основному використовуються для перетворення даних (таблиця 5), що поставляють оператори введення-виведення й обробки символьної інформації.

Якщо необхідно, щоб результатом виразу було число, то й усі вихідні дані в цьому виразі повинні бути числами. І також

якщо результат — рядок, то й усі вихідні дані мають бути рядками. У протилежному випадку можливі або помилки в програмі, або результат не буде відповідати очікуваному.

Таблиця 5 — Функції перетворення типів даних

Функція	***Опис***	***Приклад***	
		Аргументи	***Результат***
Val(рядок)	Перетворює рядок у число. Якщо у рядку немає цифр, то результат буде 0	**Val(″25″)** **Val(″Світ″)** **Val(″01рахунок″)**	**25** **0** **1**
Str(число)	Перетворює число в рядок	**Str(5)** **Str(-5)**	**5** **-5**

Функції для роботи з датою й часом наведено в таблиці 6.

Таблиця 6

Функція	***Призначення***
Date	Повертає поточну системну дату як **дд.мм.рр.**
DateAdd	Додає або віднімає зазначений інтервал до дати. Позитивні значення додають, негативні — віднімають
DateDiff	Повертає число заданих інтервалів часу, що лежать між двома зазначеними датами
Now	Як і Date повертає поточну дату, але при цьому вказується і поточний час
Time	Повертає поточний системний час як **hh.mm.ss.**

2.4.1 Функція Format

Для подання значень у форматі число, дата і час використовується функція **Format**, яка перетворює числову інформацію у символьну, і при цьому можна визначити, в якому вигляді потрібно виводити цю інформацію.

Синтаксис функції **Print Format**:

Print Format (значення, ″шаблон″)

де параметр **значення** — число, яке необхідно перетворити;

параметр **шаблон** — вид перетвореного числа. Цей параметр є або ім'ям стандартного формату, або складається із управляючих символів (таблиця 7).

Таблиця 7

Управляючий символ	***Опис***
0	Цифровий символ-заповнювач; друкує замикаючий або провідний нуль у поточній позиції
#	Цифровий символ-заповнювач; ніколи не друкує замикаючих або провідних нулів
.	Символ-заповнювач десяткового розподілювача
,	Символ-заповнювач для роздільника тисяч
%	Знак відсотків, число буде помножено на 100
- + $ () space	Символи відображаються точно так, як вони набрані у форматному рядку

Приклади використання шаблонів наведено в таблиці 8.

Таблиця 8

Синтаксис формату	***Результат***
Format**(8315.4, ″**00000**.00″)**	**08315.40**
Format**(8315.4, *″*#####**.**##*″*)**	**8315.4**
Format**(8315.4, *″*##,##**0.00***″*)**	**8,315.40**
Format**(315.4,*″*$##**0.00***″*)**	**$315.40**
Format**(0.25,*″*####**.**##**%***″*)**	**25%**

2.4.2 Функції InputBox() і MsgBox()

Вікно введення (*InputBox*) і **вікно повідомлення *MsgBox*()** використовуються при необхідності поставити користувачеві питання, ввести дані або відобразити повідомлення про помилки та запронувати можливі дії. Вікно введення при цьому завжди надає користувачеві місце для відповіді на питання.

Виводить на екран діалогове вікно, що містить повідомлення, встановлює режим очікування натискання кнопки користувачем, а потім повертає значення типу Integer, що вказує, яка кнопка була натиснута.

Синтаксис функції **MsgBox**:

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

де **prompt** — обов'язковий рядковий вираз, що відображається як повідомлення в діалоговому вікні. Максимальна довжина рядка prompt становить приблизно 1024 символи і залежить від ширини використовуваних символів. Рядкове значення prompt може містити декілька фізичних рядків. Для поділу рядків допускається використання символу повернення каретки — Chr(13), символу переведення рядка — Chr(10) або комбінація цих символів — Chr(13) & Chr(10);

buttons — необов'язковий числовий вираз, що являє собою суму значень, які вказують число і тип відображуваних кнопок, тип використовуваного значка, основну кнопку і модальність вікна повідомлення. Значення за замовчуванням цього аргументу дорівнює нулю. Аргумент може набувати одного із таких значень:

vbOKOnly = 0 — відображується лише кнопка ″OK″.

vbOKCancel = 1 — відображуються кнопки ″OK″ і ″Скасувати″ (Cancel).

vbAbortRetryIgnore = 2 — відображуються кнопки ″Перервати″ (Abort), ″Повторити″ (Retry) і ″Пропустити″ (Ignore).

vbYesNoCancel = 3 — відображуються кнопки ″Так″ (Yes), ″Ні″ (No) і ″Скасувати″ (Cancel).

vbYesNo = 4 — відображуються кнопки ″Так″ (Yes) і ″Ні″ (No).

vbRetryCancel = 5 — відображуються кнопки ″Повторити″ (Retry) і ″Скасувати″ (Cancel).

vbCritical = 16 — використовується значок ″Критичне повідомлення″.

vbQuestion = 32 — використовується значок ″Попереджувальний запит″.

vbExclamation = 48 — використовується значок ″Попередження″.

vbInformation = 64 — Використовується значок ″Інформаційне повідомлення″.

title — необов'язковий рядковий вираз, що відображається у рядку заголовка діалогового вікна. Якщо цей аргумент опущений, в рядок заголовка поміщається назва програми;

helpfile — необов'язковий рядковий вираз, який визначає ім'я файла довідки, що містить довідкові відомості про дані у діалоговому вікні. Якщо цей аргумент вказано, необхідно вказати також аргумент context;

context — необов'язковий числовий вираз, що визначає номер відповідного розділу довідкової системи. Якщо цей аргумент вказано, необхідно вказати також аргумент helpfile.

Приклад.
Var = msgbox ("нечислові дані", Vbcritical, "помилка введення")

На екрані буде виведено повідомлення (рисунок 1).

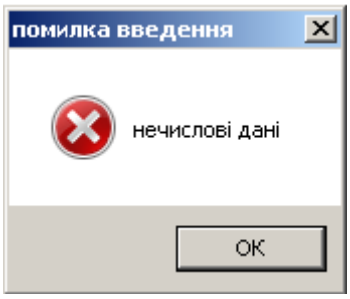


Рисунок 1

Виводить на екран діалогове вікно, що містить повідомлення і поле введення, встановлює режим очікування введення тексту користувачем або натискання кнопки, а потім повертає значення типу String, що містить текст, введений у поле**.**

Синтаксис функції **InputBox ()** :
InputBox (prompt [, title] [, default] [, Xpos] [, Ypos] [, helpfile, context])
де **prompt** — обов'язковий рядковий вираз, аналогічний функції **MsgBox**;

title — необов'язковий рядковий вираз, аналогічний функції **MsgBox**;

default — необов'язковий рядковий вираз, що відображається у полі введення за умовчанням, якщо користувач не введе інший рядок. Якщо цей аргумент опущений, поле введення зображується порожнім;

Xpos — необов'язковий числовий вираз, що задає відстань по горизонталі між лівою межею діалогового вікна і лівим краєм екрана (у твіпах). Якщо цей аргумент опущений, діалогове вікно вирівнюється по центру екрана по горизонталі;

Ypos — необов'язковий числовий вираз, що задає відстань по вертикалі між верхньою межею діалогового вікна і верхнім краєм екрана (у твіпах). Якщо цей аргумент опущений, діалогове вікно поміщається по вертикалі приблизно на одну третину висоти екрана;

helpfile — необов'язковий рядковий вираз, аналогічний функції **MsgBox**;

context — необов'язковий числовий вираз, аналогічний функції **MsgBox**.

Приклад.

Var = Inputbox ("введіть прізвище", "введення даних")

На екрані з'явиться вікно для введення прізвища (рисунок 2).

Рисунок 2

2.4.3 Функція RGB

У VB колір задається цілим значенням типу Long у стандарті **RGB** (червоний — зелений — синій). Функція RGB дозволяє визначити колір як комбінацію його трьох складових.

Синтаксис функції **RGB**:

RGB (red, green, blue)

де **red, green, blue** — обов'язкові аргументи, які задають відносну інтенсивність (значення типу Integer в інтервалі від 0 до 255 включно) відповідно червоного, зеленого та синього компонентів кольору.

Приклад.

```
Private Sub Form_Load()
BackColor = RGB(255, 255, 255)
End Sub
```

2.5 Оператори управління порядком виконання команд

Дозволяють оцінити реальну ситуацію й адекватно реагувати на неї, приймаючі рішення про подальші дії [7, 8].

2.5.1 Умовний оператор If... Then

Може мати просту однорядкову або блокову структуру.

Синтаксис однорядкової структури умовного оператору:

If <умова> Then <оператор> [Else <оператор>]

де **<умова>** — логічний вираз або логічне відношення. Коли умова істинна, тобто результат дорівнює True, то виконується оператор, наступний за **Then**, в іншому випадку — оператор, наступний за ключовим словом **Else**.

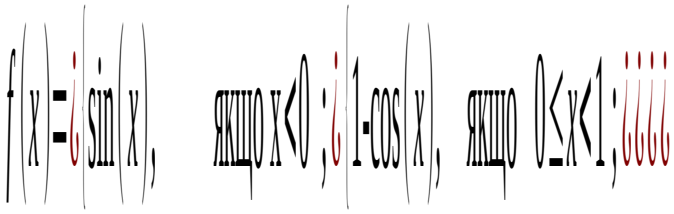
Синтаксис блокової структури умовного оператору:

```
If  <умова 1> Then
[оператори 1]
ElseIf  <умова 2> Then
[оператори 2]
…
Else
[оператори 3]
End If
```

У принципі блоковий запис надає такі ж можливості, як і однорядковий. Але якщо залежно від умов необхідно виконати не просто команду, а групу операторів, слід використовувати блоковий синтаксис. Крім того, блокова структура з **Else If** дозволяє аналізувати кілька умов.

Використання блокового синтаксису покращує читабельність програми.

Приклад. Запишемо функцію для обчислення значення



```
Function f (x As Single) As Single
 If x <0 Then
        f = sin (x)
 ElseIf x <1 Then
        f = 1-cos (x)
 Else
f = exp (sin (x))
 End If
End Function
```

2.5.2 Оператор вибору варіанта Select Case

Оператор вибору варіанта дозволяє виконувати одну з декількох груп операторів залежно від значення умови.

Синтаксис оператора **Select Case:**

Select Case <перевірна змінна>
[Case < значення 1 >
[оператори 1]]
[Case < значення 2 >
[оператори 2]]
…
[Case Else
[оператори 3]]]
End Select

Для блока **Case** можна вказувати не тільки одне значення, а й декілька, розділених комою. Можна також визначати області порівняння **<значення 1> To <значення 2>** або скористатися відносним порівнянням **Is > = <значення>.**

Блок **Case Else** виконується, коли жодна з попередніх умов не є істинною.

Коли умові в **Select Case** відповідає кілька блоків, то виконується перший з них.

Приклад. Для обчислення значення функції з попереднього прикладу можна записати

```
Function f (x As Single) As Single
 Select Case x
 Case Is <0
        f = sin (x)
 Case Is <1
        f = 1-cos (x)
 Case Else
f = exp (sin (x))
 End Select
End Function
```

2.5.3 Оператори організації циклів

Цикл з лічильником дозволяє організувати багаторазове виконання одного або декількох операторів, надаючи можливість заздалегідь визначати кількість повторів циклу.

For <лічильник> = поч_знач> To <кін_знач> [Step <крок>]
[оператори 1]
[Exit For]
[оператори 2]
Next [< лічильник>]

На початку лічильник встановлюється у початкове значення. При кожному проході лічильник збільшується на величину кроку, або на 1, якщо крок не задано. Якщо ця змінна перевищує (становиться менше при від'ємному кроці) кінцеве значення, цикл завершується і виконуються оператори, наступні після **Next**.

Різниця між кінцевим і початковим значеннями, поділена на величину кроку, складає кількість повторень, зменшену на 1.

Приклад.
Дана функція обчислює значення факторіала **n!**

```
Function Factorial (n As Integer) As Long
 Dim i As Integer
 Factorial = 1
 For i = 1 To n
       Factorial = Factorial * i
 Next i
End Function
```

Цикл з умовою використовується, якщо кількість проходів залежить від деяких умов.

Цикл, керований на початку (цикл з передумовою)

Do [{While | Until} < умова >]
[оператори 1]
[Exit Do]
[оператори 2]
Loop

Цикл, керований в кінці (цикл з постумовою)

Do
[оператори 1]
[Exit Do]
[оператори 2]
Loop [{While | Until} < умова >]

В обох випадках тіло циклу виконується невизначену кількість разів, поки умова не спонукає вихід з циклу. У першій конструкції перевірка умови здійснюється до виконання операторів між **Do** та **Loop** (тіла циклу), а у другій — після.

<Умова> може бути будь-яким виразом, який VB може оцінити як **True** або **False**. При використанні **While** (поки) виконання тіла циклу триває до тих пір, поки умова має значення **True.** Якщо задано **Until**, виконання тіла циклу припиняється, як тільки умова набуде значення **True.**

Для альтернативного виходу з циклу можна використовувати оператор **Exit DO.** Зазвичай він використовується в оператором **If...Then**.

Приклад. Для обчислення суми ряду $y = \sum_{k=1}^{\infty} \frac{1}{x^k}$ із заданою точністю ε (тобто поки черговий доданок не стане менше ε) можна використовувати таку функцію:

```
Function Sum (x As Single, e As Single) As Single
Sum = 0
k=1
a = 1 / x^k
Do While a > = e
Sum = Sum + a
k=k+1
a = 1/ x^k
Loop
End Function
```

3 Властивості. Події. Методи

Кожен об'єкт має характерний його класу набір властивостей, подій і методів.

3.1 Основні властивості

Властивості визначають зовнішній вигляд та поведінку об'єкта. З ними можна працювати як зі змінними, дотримуючись певного синтаксису (див. п.2.2.4).

Нижче розглядаються основні властивості, якими володіє більшість об'єктів. Їх значення встановлюються в основному на етапі проектування, найчастіше візуально.

Ім’я об'єкта. Властивість **Name** відіграє особливу роль. Ім'я об'єкта є його ідентифікатором. Всі дії з об'єктом здійснюються

через його ім’я. Тому завжди спочатку слід визначити ім’я об'єкта, а вже потім писати для нього код обробки події.

Позицію об'єкта в контейнері визначають чотири властивості: **Left**, **Top**, **Height** і **Width**.

Як одиниці вимірювання розміру використовується твіп (twip) — незалежна від пристрою одиниця, яка дорівнює 1/20 точки принтера, що гарантує незалежність відображення об'єктів від дозволу дисплея 567 твіпов = 1 см.

Властивості **Top** і **Left** задають координати верхнього лівого кута об'єкта. Властивості **Height** і **Width** — його висота і ширина. Відлік у системі координат ведеться зверху вниз (Y) і зліва направо (X).

Колір. Управління колірнім оформленням об'єктів здійснюється за допомогою властивостей **BackColor** (колір фону), **ForeColor** (колір для відображення тексту і графіки) і **FillColor** (колір заповнення).

Параметри шрифту. Для вибору й установлення параметрів шрифту в VB існує об'єкт **Font.**

Зовнішній вигляд. Більшість об'єктів має властивість **Appearances**, що відповідає за відображення об'єкта (без візуальних ефектів або в тривимірному вигляді).

Властивість **BorderStile** визначає наявність і вигляд рамки біля об'єкта. Для форми ця властивість визначає також тип вікна та можливість зміни його розмірів.

Властивість **Caption** дозволяє задавати текст, що ідентифікує об'єкт на екрані при виконанні програми (текст на кнопці, в заголовку вікна і т.д.).

Надання підказки користувачеві. Властивість **ToopTilText** дозволяє визначити текст підказки, тобто текст, який з'являється на екрані як підказка **ToopTil** при наведенні покажчика миші на об'єкт у час виконання програми.

Зовнішній вигляд покажчика миші. Форма покажчика миші при його наведенні на об'єкт визначається властивістю **MousePointer**.

Доступність і видимість об'єкта визначаються за допомогою властивостей **Enabled і Visible**.

Якщо **Enabled = True**, об'єкт доступний. При **Enabled = False** об'єкт можна побачити на екрані, але звернутися до нього не можна.

Якщо **Visible = True**, об'єкт відображається на екрані, в іншому випадку (**Visible = False)** об'єкт з екрана зникає (стає невидимим). Ці властивості часто змінюються програмно (на етапі виконання) залежно від ситуації: робочий об'єкт бачимо (або доступний) або приховуємо.

Порядок отримання фокуса за Tab (Shift + Tab). Індекс об'єкта який визначає порядок переходу при натисканні клавіші Tab або Shift + Tab, встановлюється властивістю **TabIndex.**

Фокус — поняття етапу виконання (**RunTime)**. Для обслуговування безлічі елементів управління є тільки одна клавіатура. Тому система повинна визначити, якому елементу передається управління для введення з клавіатури інформації. У зв'язку з цим у Windows використовується поняття ″**фокус**″. Якщо говорять, що елемент управління має фокус (або активний), це означає, що введення інформації з клавіатури відноситься саме до цього елемента.

Елемент, що має фокус, можна розпізнати за різними візуальними ознаками. Найчастіше це миготливий курсор (у текстовому полі) або пунктирна рамка біля елемента управління.

Слід ще раз підкреслити, що фокус пов'язаний з можливістю вікна або елемента управління отримувати натискання клавіш. Для миші фокус не потрібен. Де б користувач не натиснув, викликається процедура обробки того елемента управління, на якому був курсор у момент натискання, незалежно від того, який елемент управління мав фокус перед цим натисканням.

Фокус може встановлюється у результаті натискання на елемент управління. Після цього елемент може отримувати інформацію з клавіатури.

Інший спосіб встановлення фокуса — натискання клавіші Tab або Shift+Tab.

Всі елементи управління, для яких можливо встановити фокус, мають властивість **TabIndex**, яка управляє послідовністю переходу при натисканні клавіші Tab. Елемент управління, для якого властивість **TabIndex = 0,** отримує фокус одразу після завантаження форми. Після натискання клавіші Tab

фокус переходить до елемента з наступним значенням **TabIndex** (Shift + Tab — до попереднього).

Послідовність переходів за Tab (**фокусний порядок)** слід створювати таким чином, щоб користувач міг простежити її логічно, а не шукати активний елемент управління. Для цього слід коректно привласнювати значення властивості **TabIndex**.

VB автоматично встановлює значення властивості **TabIndex** по мірі додавання елементів управління у форму: кожен новий елемент управління отримує значення цієї властивості на одиницю більше, ніж у попереднього. Цю послідовність можна змінити, змінивши значення властивостей **TabIndex** у необхідного елемента управління. При цьому VB автоматично змінить значення індексів інших елементів управління.

Tag — властивість, яка призначена для зберігання будь-яких додаткових даних, необхідних розробникові.

3.2 Основні події

Події (Events) пов'язані з певними діями користувача над об'єктом і можуть викликати код (процедуру) обробки події.

Події миші. Є дві події, що викликаються натисканням миші: **Click** (натискання) і **DblClick** (подвійне натискання).

Події клавіатури. При натисканні клавіші для активного елемента управління генерується подія **KeyPress**, яка повертає код натиснутої клавіші.

VB дозволяє обробляти дві події, пов'язані з передачею фокуса: **LostFocus** (втрата фокуса) і **GotFocus** (отримання фокуса).

3.3 Методи

Коли необхідно виконати деякі дії над об'єктами, застосовують **метод об'єкта.** За допомогою методу можна замінити ті чи інші властивості об'єкта, створити/видалити в ньому нові елементи або сам об'єкт, перемістити його на інше місце і т. д.

Загалом метод можна розглядати як деяку вбудовану до об'єкта процедуру, активізувати яку можна, викликавши відповідний метод. Багато методів мають аргументи, що дозволяють уточнювати, як вони повинні діяти на об'єкт. Синтаксис звернення до методу нагадує звернення до властивості:

[<Форма>.] <Ім’я_елемента_управління>. <Метод> [<аргументи>]

Межі між методами і властивостями часом розпливчасті. Так, у багатьох об'єктів є метод **Move**, що дозволяє змінити позицію об'єкта. Однак властивості **Top** і **Left** виконують фактично ту ж саму функцію. Основна відмінність між методами і властивостями полягає у тому, що з властивостями можна працювати як під час розроблення проекту, так і при виконанні програми, тоді як методи доступні тільки при виконанні. Слід зазначити, що деякі властивості можуть бути також недоступні при розробленні програми.

При зверненні до властивостей об'єктів поточної форми ім’ям цієї форми можна знехтувати.

4 Відображення результатів розрахунків

Результати невеликих розрахунків можна вивести на екран за допомогою елементів управління. Для відображення великих обсягів структурованої інформації як текстового так і графічного характеру VB надає такі об'єкти:

1) системний об'єкт **Printer**;

2) форма (об'єкт **Form**);

3) елемент управління **PictureBox.**

Всі три об'єкти є контейнерами, тобто в них можна помістити будь-які інші об'єкти. Крім того, використовуючи відповідні методи, можна безпосередньо в цих об'єктах виконувати рисування як тексту, так і графіки.

Більшість методів і властивостей даних об'єктів дуже схожі. Тому розглянемо їх разом, роблячи акценти в тих місцях, де вони відрізняються.

4.1 Властивості

Основні властивості, якими найчастіше користується більшість об'єктів, розглянуті в розділі 3. Однак ці об'єкти мають і деякі специфічні, притаманні тільки їм властивості.

Розглянуті раніше властивості **Height і Width** містять фізичні розміри контурів об'єкта — аркуша, вікна та ін.). Виведення інформації здійснюється у досить вузької області — робочій зоні: частина площини об'єкта виділяється на поля, зону заголовка вікна, меню і т.д. Тому для визначення робочої зони використовують такі властивості:

- **ScaleLeft —** відступи зліва (ліве поле);
- **ScaleTop —** відступи зверху (верхнє поле);
- **ScaleWidth —** ширина робочої зони;
- **ScaleHeight —** висота робочої зони.

Всі координати при поданні інформації у ці об'єкти задаються відносно робочої зони: верхній лівий кут має координати (0,0). Розміри вказуються зазвичай у твіпах (див. п. 3.1). При необхідності одиницю вимірювання можна змінити за допомогою властивості **ScaleMode**.

Виведення інформації починається з точки виведення, тобто точки, в якій знаходиться курсор. Для зміни позиції точки виведення (установлення у потрібне місце курсора) використовують властивості **CurentX і CurentY.**

Слід зазначити, що в процесі виведення інформація за межі кордонів об'єкта, вона просто буде скорочуватися і втрачатися.

Властивість **BorderStile** для форми має більш розширене значення, ніж для інших об'єктів, і визначає вид рамки та вікна:

0 (vbBSNone) — вікно без рамки. Розмір змінити неможливо. Переміщувати неможливо. Нема рядка заголовка;

1 (vbFixedSingle) — фіксована рамка. Є заголовок, але розмір вікна змінити неможливо;

2 (vbSizable) — допускаються всі зміни вікна. Це значення за замовчуванням.

3 (vbFixedDialog) — товста рамка. Зміна розміру неможлива. Максимізація та мінімізація можливі, але із системного меню.

Зовнішній вигляд заголовка вікна визначається властивостями **CorelBox** (наявність системного меню), **MaxButton** (наявність кнопки максимізації) і **MinButton** (наявність кнопки мінімізації).

4.2 Події

Слід зазначити одну особливість — ім’я процедури обробки події містить **Form** замість конкретного імені форми.

Подія **Load** виникає при завантаженні форми, а подія **Unload** — при її видаленні з екрана. Під час обробки події **Load** форму ще не видно на екрані.

Подія **Resize** виникає при зміні розмірів форми. У тому числі ця подія виникає і при завантаженні форми, тому що при цьому її розміри змінюються від нуля до заданих.

4.3Методи

Мабуть, найбільш важливими методами для об'єктів, які відображають інформацію (**Printer**, **Form** і **PictureBox)** є ті, що здійснюють виведення даних (тексту, графічних зображень).

У зв'язку з тим, що при зверненні до поточної форми її ім’я необов’язково вказувати, виклик методів форми дуже нагадує використання операторів VB. Фактично, щоб викликати метод для поточної форми, досить вказати ім’я цього методу і через пробіл — його аргументи.

Розглянемо деякі методи.

1 Очищення екрана (CLS). Здійснює видалення з екрана (вікна форми або **PictureBox)** всієї інформації, що виводилась раніше за допомогою методів об'єкта. Елементи управління, що знаходяться в об'єкті, не змінюються.

2 Виведення даних в об'єкт. Синтаксис оператору:

Print [Spc (n) | Tab (n)] <список_виведення>

<**Список_виведення**> — вирази, значення яких обчислюються і виводяться на екран або принтер. Роздільником у списку є крапка з комою або кома. Крапка з комою

використовується у тому випадку, коли таке значення слід вивести одразу за попереднім, праворуч від нього. Кома вказує на необхідність виведення наступного значення у наступній зоні друку. **Зона друку (print zona)** становить кожні 14 стовпців на екрані або аркуші. Кінцева крапка з комою затримує курсор на екрані або встановлює його в кінці рядка для подальшого виведення із цієї ж позиції. Тому наступний метод **Print** виведе дані в тому ж рядку, правіше попереднього виведення.

Метод **Print** підтримує вбудовані функції **Spc(n)** і **Tab(n)** для додаткового управління виведенням даних. **Spc(n**) створює при виведенні рядок пропусків, кількість яких задається аргументом **n**. Функція **Tab(n**) визначає, в якому стовпці (**n)** з'явиться наступний перший друкований символ, тобто задає номер зони друку.

3 Виведення точки. Синтаксис оператору:

Pset [step] (X, Y) [, <колір>],

де X, Y — координати точки на екрані;

step — вказує, що координати беруться як зміщення відносно поточного положення курсору (точки виведення);

<**колір**> **—** колір точки в стандарті **RGB**; якщо колір не вказується, використовується поточний основний колір (**CoreColor**).

4 Рисування лінії, прямокутника. Синтаксис оператору:

Line [[step] (X1, Y1)] — [step] (X2, Y2) [, [, < колір >] [[, B [F]]]

де **(X1, Y1)** — координати початку лінії (якщо знехтовано — початок у поточній позиції);

(X2, Y2) — координати кінця лінії;

step — вказує на відносні координати, тобто координати обчислюють як зміщення відносно останньої точки;

<**Колір**> — колір лінії у стандарті RGB (за замовчуванням **CoreColor)**;

B — опція рисує прямокутник з координатами верхнього лівого кута (X1, Y1) і нижнього правого (X2, Y2);

BF — опція рисує зафарбований вказаним кольором прямокутник.

5 Рисування еліпса, кола або дуги. Синтаксис оператору:
Circle [step] (X, Y), <радіус> [, [<колір>] [, <початок>]
[[, <кінець>] [, <коефіцієнт>]]]

де **(X, Y)** — координати центра;

step — вказує, що (X, Y) — відносний центр від поточної позиції курсора;

<**радіус**> — радіус;

<**колір**> — атрибут кольору контура в стандарті **RGB,** за замовчуванням основний (**CoreColor)**;

<**початок>**, <**кінець**> — використовується для рисования дуг. Їхні значення від -2*PI до + 2*PI, де PI = 3,14.... За замовчуванням початок = 0, кінець = 2 * PI;

<**коефіцієнт**> — коефіцієнт стискання — відношення довжини півосей еліпса (радіуса ″ Y ″ до радіуса ″ X ″). За замовчуванням коефіцієнт дорівнює 1 (для рисования кола).

Приклад. У наведеному нижче прикладі в центрі робочої області поточної форми червоного кольору рисується коло, діаметр якого дорівнює половині мінімального розміру форми, і проводяться дві вертикальні лінії синього кольору.

```
Sub Prim
 If ScaleWidth <ScaleHeight Then
       R = ScaleWidth / 4
 Else
       R = ScaleHeight / 4
 End If
    Circle (ScaleWidth / 2, ScaleHeight / 2), R, vbRed
    Line (ScaleWidth / 2,0) - (ScaleWidth / 2, ScaleHeight), vbBlue
    Line (0, ScaleWidth / 2) - (ScaleWidth, ScaleHeight / 2), vbBlue
End Sub
```

5 Огляд основних елементів управління (об'єктів)

Створення Windows-додатків практично неможливо без використання елементів управління, оскільки саме вони дозволяють користувачеві взаємодіяти з цим додатком.

5.1 Командна кнопка (CommandButton)

Призначення. Почати, перервати або запустити будь-якій процес.

Властивості. Властивість **Default** визначає, що кнопка є активною за замовчуванням. У цьому випадку натискання Enter автоматично генерує подію **Click** для цієї кнопки. Аналогічно властивість **Cancel** забезпечує перехоплення кнопки ESC і виклик події **Click** для відповідної кнопки. Ці властивості можуть бути присвоєні лише одній з кнопок форми.

Події. Головна подія — **Click**.

5.2 Напис (Label)

Призначення. Відображення постійного тексту, який користувач не може змінити з клавіатури.

Властивості. Найважливіша — **Caption**, що містить текст, який відображується. **BorderStyle** задає наявність і тип рамки. **AutoSize=True** встановлює розмір напису відповідно до довжини тексту. **WordWrap=True** задає перенесення слів у рядку на наступний рядок.

Події. Зазвичай не використовують.

5.3 Текстове поле (TextBox)

Призначення. Основний елемент управління для запровадження даних.

Властивості. Найважливіша — **Text,** що містить текст, який відображується у полі.

У текстовому полі можна виділяти фрагмент тексту за допомогою властивостей **SelStart** (початкова позиція) та **SelLength** (кількість виділених символів).

Наприклад, для виділення всього тексту при отриманні фокуса:

```
Private Sub FLD_GotFocus ()
FLD.SelStart=0
FLD.SelLength=Len(FLD.Text)
Ens Sub
```

За допомогою властивості **SelText** можна прочитати або змінити виділений фрагмент тексту.

Події. Change — зміна вмісту текстового поля. Відбувається кожен раз при введенні, видаленні або заміні символу. Наприклад, при введенні слова Hello ця подія виникає п'ять разів. Для аналізу введеного тексту найкраще підходить подія **LostFocus.** Дана подія викликається після втрати фокуса елементом управління (передачі фокуса іншому елементу після завершення введення або коригування даних). Однак якщо поле єдине у формі, воно не може втратити фокус. Щоб видалити або форматувати вміст текстового поля використовується подія **GotFocus,** що виникає, коли користувач ″входить″ у текстове поле (поле отримує фокус).

5.4 Прапорець (CheckBox)

Призначення. Вибір одного з двох значень: Так / Ні, Увімкути / Вимкнути і т.п. Це елемент управління, який можна відмічати (ставити ″галочку″). Можливий і третій стан — відмічений, але недоступний.

Властивості. Найважливіша властивість прапорця — його значення (**Value)**, може набувати значень:

0 — невідмічений;

1 — відмічений;

2 — відмічений, але недоступний.

Події. Головна подія — **Click.**

5.5 Перемикач (OptionButton)

Призначення. Установлення тільки однієї позиції з групи. Кожна група перемикачів поміщається в окремий контейнер, наприклад, **Frame.** Форма є контейнером за замовчуванням.

Властивості. Значення перемикача **Value,** що може бути **True** (увімкнений) або **False** (вимкнений).

Події. Головна — **Click**.

5.6 Список (ListBox)

Призначення. Вибір зі списку одного або декількох елементів. У будь-який час у список можна додати нові елементи або видалити існуючі. При відображенні у списку може з'явитися смуга прокручування.

Властивості. Найпростіши спосіб отримати текст вибраного елемента — властивість **Text.** Індекс обраного елемента — властивість **ListIndex**. Доступ до потрібного елемента списку можна здійснити за його індексом за допомогою властивості **List ().** Поточна кількість елементів в індексі визначається властивістю **List Count.** Отримати обраний елемент можна також, комбінуючи властивості **List() і ListIndex**:

```
Str = LstBox.List (LstBox.ListIndex)
```

Це те саме, що і:

```
Str = LstBox.Text
```

Якщо не вибрано жоден елемент, значення властивості **ListIndex** = 1, а властивості **Text** — порожній рядок.

Властивість **Sorted** (сортування) визначає спосіб розташування елементів в списку. Якщо встановити цю властивість, то всі елементи будуть сортуватися за алфавітом.

Властивість **ItemData ()** дозволяє для кожного елемента списку задати якесь число (код, індивідуальний номер і т.п.), що не залежить від порядку розташування елементів у списку (сортування). При цьому для отримання індексу останнього доданого елемента можна скористатися властивістю **NewIndex**.

Наприклад, додати до списку співробітника Іванова з індивідуальним номером 52431:

```
LstPersonal.AddItem = ″ Іванов ″
LstPersonal.ItemData (LstPersonal.NewIndex) = 52431
```

Властивість **MultiSelect** надає можливість користувачеві вибрати одночасно кілька значень (множинний вибір):

0 — множинний вибір неможливий (можна вибрати тільки один елемент);

1 — простий множинний вибір. Елементи списку вибираються натисканням мишу або на пробіл;

2 — розширений множинний вибір. При виборі можливо використовувати клавіші Shift і Ctrl.

При множинному виборі властивість **Text** містить текст останнього вибраного елемента. Для обробки всіх обраних елементів слід скористатися властивістю **Selected (),** наприклад:

```
If LstPersonal.Selected (2) Then
Str = LstPersonal.List (2)
End If
```

Події. Основна — **Click,** що виникає при виборі елемента в списку (за допомогою миші або клавіші управління).

Методи. Для додавання нових елементів у список використовується метод **AddItem**, а для видалення — **RemoveItem**. При додаванні слід вказати значення елемента, що додається, і його індекс — місце вставки у список. Індексація починається з нуля. Якщо індекс не завдано — вставка здійснюється у кінець списку. Для видалення досить задати тільки індекс. Як правило, заповнення списку здійснюється при завантаженні форми

```
Private Sub Form_Load
LstBox.AddItem ″Вс-01-1″
LstBox.AddItem ″Вс-01-2″
LstBox.AddItem ″Вс-01-3″
End Sub
```

Приклад. Видалення третього елемента списку:

LstBox. RemoveItem 2

Для видалення всіх елементів — метод **Clear** :
LstBox. Clear

5.7 Поле зі списком (ComboBox)

Призначення. По суті це комбінація списку і текстового поля. У такому списку потрібне значення можливо вибрати або ввести у текстовому полі.

Властивості. Цей об'єкт має багато рис текстового поля **TextBox** і списку **LstBox,** виняток — властивість **MultiLine**. Особливо слід виділити властивість **Style,** що визначає зовнішній вигляд і функціонування поля зі списком:

0 — текстове поле і список (за замовчуванням);

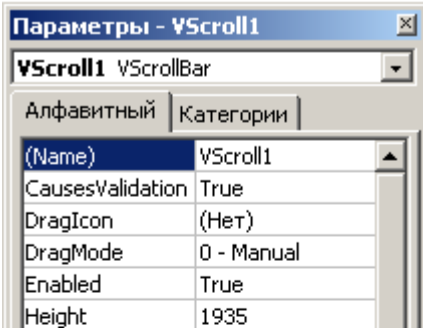
1 — текстове поле і постійно розкритий список;

2 — текстове поле і список без права введення тексту в текстовому полі.

Події. Основні — **Click** — вибір елемента списку і **Change** — зміна тексту в текстовому полі.

5.8 Смуга прокручування (Scrollbar)

Призначення. Створення смуг (лінійок) прокручування. Вона не виконує автоматично будь-яких дій, тобто її поведінку необхідно програмувати. Крім свого прямого призначення (прокручування документа у вікні перегляду), вона може бути використана для задавання або зміни значення будь-якої величини. Замість вказівки конкретного значення у цьому випадку задається позиція бігунка всередині діапазону значень, яка змінюється переміщенням миші. **Бігунок** — це рухомий прямокутник на лінійці прокручування (нагадує кабінку ліфта). Панель елементів включає до себе як горизонтальну (**Horizontal**), так і вертикальну (**Vertical)** смуги прокручування.



Властивості. Перед використанням смуги прокручування потрібно

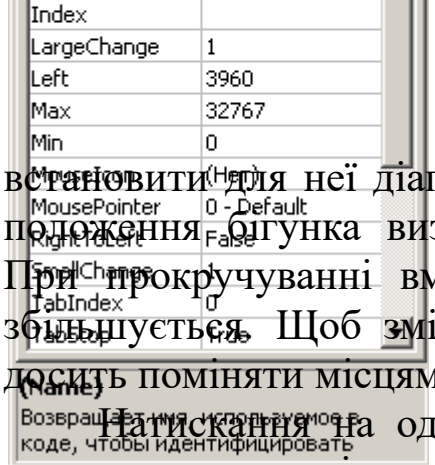


Рисунок 3 — Властивості смуги прокручування

встановити для неї діапазон — властивості **Min і Max**. Поточне положення бігунка визначається значенням властивості **Value**. При прокручуванні вмісту вікна зверху вниз значення **Value** збільшується. Щоб змінити напрямок зміни властивості **Value,** досить поміняти місцями значення властивостей **Min** і **Max**.

Натискання на одну з двох кнопок зі стрілками на смузі прокручування змінює значення властивості **Value** на величину, яка визначається властивістю **SmallChange**. Натискання в області між бігунком і кнопкою змінює значення Властивості **Value** на величину, яка визначається властивістю **LargeChange** (рисунок 34).

Події. Найважливіша подія — **Change,** що виникає після зміни позиції бігунка, тобто властивості **Value**.

5.9 Таймер (Time)

Призначення. Запуск або завершення будь-яких процесів, додатків у певні моменти часу. Під час проектування таймер відображується у формі, але при виконанні додатка він стає невидимим.

Властивості. Для установлення інтервалу часу служить властивість **Interval,** значення якої встановлюється у мілісекундах. Для відключення таймера слід надати цій властивості нульове значення, а властивості **Enabled** значення **False.**

Події. Єдина подія — **Timer**, яка викликається після закінчення встановлення часового інтервалу.

Якщо обробка події **Timer** триває більше часу, заданого у властивості **Interval**, нова подія не викликається, поки Visual Basic не обробить цю подію.

5.10 Рамка (Frame)

Призначення. Це один з елементів — контейнерів. Його призначення — об'єднання у групу кількох елементів управління. Об'єкти, об'єднані за допомогою рамки, можна як єдине ціле переміщати, активувати, робити скритими.

Деякі елементи самі потребують контейнерів, наприклад, всі перемикачі об'єднуються у групу.

Для об'єднання об'єктів у групу (тобто розташування їх у рамці) спочатку створюють елемент-контейнер (тобто рамку), а потім у ньому — об'єкти. Якщо ж просто перемістити у рамку елемент управління з іншого місця у формі, то він не буде в рамці, а буде розташований зверху рамки. VB не розглядатиме такі елементи, які об'єднані рамкою. Щоб додати нові управляючі елементи у рамку, в якій вже є елементи управління, потрібно виділити один з елементів рамки і тільки після цього додавати новий.

Властивості. Особливих властивостей немає.

Події. Зазвичай не аналізуються, тому що найчастіше проектувальник працює тільки з елементами управління, які знаходяться у рамці.

5.11 Вікно з рисунком (PictureBox)

Призначення. Відображення рисунків та інших графічних об'єктів. Є елементом-контейнером, тому можна використовувати для об'єднання інших об'єктів.

Властивості. Властивість **Picture** містить графічний об'єкт. Це може бути растрове зображення (*. BMP), піктограма (*. ICO), метафайл (*. WMF) або розширений метафайл (*. EMF), а також файли GIF і JPEG.

При виконанні програми зображення можна скачувати за допомогою функції **LoadPicture**:

Picture1.Picture = LoadPicture (″C: \ Picture.bmp″)

або зберегти, використовуючи функцію **SavePicture** :

SavePicture Picture1.Picture, ″ Picture.bmp ″

Події. Зазвичай не обробляються.

Методи. Методи **PictureBox** дозволяють нарисувати точку, лінію і коло, а також вивести текст (див. п. 4.3):

```
PicDisplay.Print ″Hell про World″
PicDisplay.Line (0,0) - (100.500), VBRed
PicDisplay.Circle (300,300), 250, VBBlue
```

5.12 Зображення (Image)

Призначення. Відображення зображення. Але на відміну від **PictureBox** не є контейнером: не допускає рисування і групування елементів. Однак використовує менше ресурсів і перерисовує швидше.

Властивості. Головна властивість — **Picture.** З її допомогою можна визначити рисунок, який відображується елементом управління (як і в елементі управління PictureBox)**.**

Події. Зазвичай не аналізуються.

5.13 Меню (Menu)

Призначення. Створення операційного меню. Процес проектування меню здійснюється за допомогою спеціального інструмента — редактора меню (рисунок 4), викликати який можна за допомогою команд Tools⇒Menu Editor... або клавіш Ctrl + E.

Меню має ієрархічний характер і складається із декількох (не більше 6) рівнів. Верхній рівень — це рядок меню.

Для визначення рівнів служать кнопки зі стрілками у вікні проекту меню. Кнопка зі стрілкою вправо зміщує елемент меню на рівень нижче, а зі стрілкою вліво — на рівень вище. Кнопки з вертикальними стрілками змінюють позицію окремих елементів у меню, тобто переміщують їх у списку вгору або вниз.

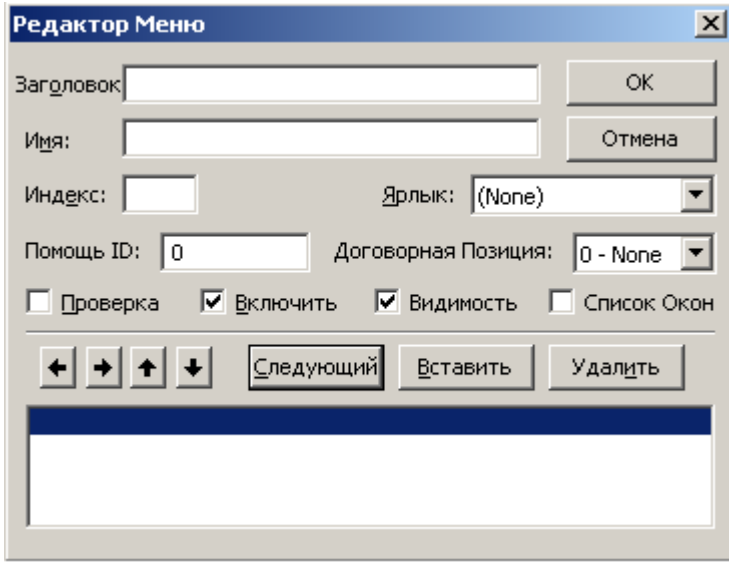


Рисунок 4 — Вікно проекту меню

Властивості. Як і будь-який елемент управління, меню також має властивості, які можна встановити прямо у вікні проекту меню (рисунок 4).

Властивість **Caption** містить текст, який буде видно в рядку меню. Для визначення ″гарячих клавіш″ (що забезпечують швидкий доступ до команд меню) досить перед відповідною буквою поставити амперсант (**&**). Крім того, можна визначити клавіші для швидкого доступу до окремих елементів меню за допомогою властивості **Shortcut.** Властивість **Visible** дозволяє відображати або приховувати окремі пункти меню; **Enabled** — зробити пункт недоступним для користувача; **Checked** — відмітити вибрані елемент ″галочкою″.

Події. Обробляється подія **Click** — вибір пункту меню. Для створення процедури обробки цієї події (тобто виконання команди меню) слід під час проектування обрати відповідний пункт у рядку меню. VB відобразить спадне меню, показуючи, що буде, коли користувач вибере цей пункт під час виконання програми. Слід пройти по всіх пунктах ієрархії меню. При натисканні на пункті найнижчого рівня відкриється вікно **Code** для введення нової процедури обробки події для цього пункту.

5.14 Лінія (Line)

Призначення. Зображення на формі ліній різної ширини, довжини і стилю.

Властивості. Основні властивості задають зовнішній вигляд лінії:
BorderColor — колір лінії;
BorderStyle — стиль лінії;
BorderWidth — ширина лінії у твіпах.

Події. Зазвичай не обробляються.

5.15 Фігура (Shape)

Призначення. Рисування різних фігур, що мають різні властивості.

Властивості. Основні властивості зміни вигляду фігур:

BackColor — колір фону;

BackStyle — 0 — фігура прозора (тобто видно фон), 1 — фон не видно;

BorderColor — колір границі фігури;

BorderStyle — стиль границі;

BorderWidth — ширина границі;

FillColor — колір внутрішніх ліній;

FillStyle — стиль внутрішнього рисунка;

Shape — вигляд фігури.

Вигляд фігури може прийняти одне з 6 значень:

0 — Rectangle — прямокутник;

1 – Square — квадрат;

2 – Oval — овал;

3 – Circle — коло;

4 – RoundedRectangle — прямокутник із закругленими кутами;

5 – RoundedSquare — квадрат із закругленими кутами.

Події. Зазвичай не обробляються.

5.16 Генерація стандартних діалогових вікон (CommonDialogBox)

Призначення. За допомогою цього елемента можна встановлювати різні стандартні діалогові вікна, які широко використовуються у різних додатках Windows, як наприклад, вікно діалогу **Відкриття файла** або **Закриття файла.**

Хоча **Common Dialog Box** і є стандартним елементом управління, але щоб він з'явився у панелі елементів управління, його слід встановити, виконавши команду **Project⇒Components**. Потім у вікні діалогу **Components** встановити прапорець навпроти стрілки **Microsoft Common Dialog Control 6.0** і натиснути OK. До речі, тут же можна вибрати й інші елементи

управління, наприклад **Microsoft Calendar,** якщо необхідно відображати дані календаря.

Після переміщення у форму цього елемента управління він на етапі виконання у формі не з'являється (також як і елемент управління **Timer**). Виклик відповідного вікна здійснюється у коді за допомогою методів цього елемента за необхідності.

Елементи **CommonDialogBox** — це оболонки, що надають користувачам стандартні діалогові вікна, з якими вони можуть взаємодіяти в добре знайомій манері: генеровані вікна поводять себе так само, як і аналогічні вікна в інших додатках. Після того як користувач зробить свій вибір, код додатка повинен перевірити зміну значення властивостей і якимось чином прореагувати на дію користувача. Іншими словами, якщо навіть користувач у вікні **Відкриття файла** натиснув на той чи інший файл, а потім — кнопку ОК, це не означає, що обраний файл відкритий. Робота діалогового вікна полягає лише в тому, щоб надати користувачеві знайомий інтерфейс. Після закриття вікна необхідно перевірити повернені їм значення і зробити в коді відповідні дії з відкриття файла. Те ж саме стосується й інших стандартних діалогових вікон, оскільки вони лише повідомляють додаткам про вибір користувача, але не вживають жодних самостійних заходів.

Властивості. Після додавання у форму проекту елемента управління **CommonDialogBox** у вікні **Properties** з'явиться нова властивість **Custom**, яка служить для визначення різних параметрів елемента управління. Щоб відкрити вікно, де задаються всі ці параметри, потрібно натиснути правою кнопкою на значення **CommonDialogBox** і вибрати **Properties**.

На рисунку 5 показано багатосторінкове вікно **Properties Pages,** яке з'являється після подвійного натискання властивості **Custom**. Тут немає можливості описати всі властивості, показані тільки основні.

Окремі пункти стандартних вікон можна встановити в коді підпрограми.

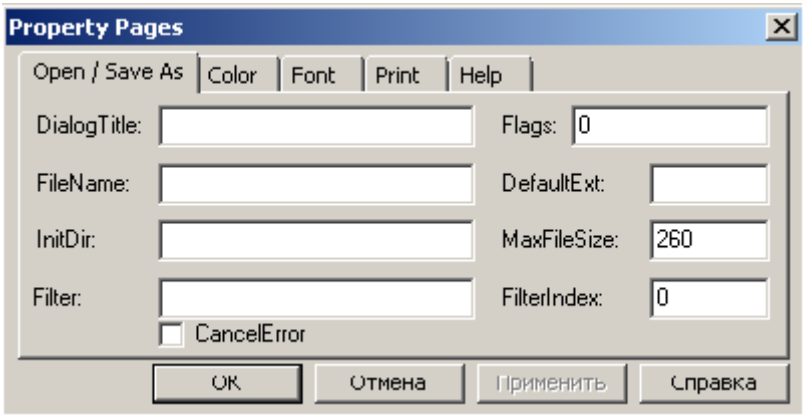


Рисунок 5 — Багатосторінкове вікно

Події. Зазвичай не використовують.

Методи. Для виведення на екран діалогових вікон використовують такі методи:

ShowColor — вікно **Колір**;
ShowFont — вікно **Вибір шрифту;**
SowHelp — вікно **Довідкова система**;
SowOpen — вікно **Відкриття файла**;
SowPrinter — вікно **Друк**;
SowSave — вікно **Збереження.**

Після того як користувач зробить свій вибір і закриє діалогове вікно, програма має перевірити, який вибір зроблено, шляхом аналізу відповідних властивостей.

6 Файли

Файл — послідовність логічно пов'язаних даних, які розміщуються на зовнішньому пристрої. На магнітному диску кожен файл зберігається під унікальним ім'ям у папці (каталозі). У кожен конкретний момент для обробки доступний тільки один компонент файла (поточний). Це запис. На цей компонент встановлений покажчик поточної позиції у файлі.

Запис — один рядок файла, що є одиницею обміну між додатком і файлом на диску. Процедура перенесення даних до файла на диску називається записом, а процедура вибору даних із файла — читанням або введенням.

Зазвичай всі файли вважаються файлами послідовного доступу. Це означає, що додавати записи у файл можна тільки послідовно, один за одним (у кінець файла). Те ж відноситься і до читання. Після виконання операції введення/виведення покажчик поточної позиції у файлі просувається на одну позицію.

Відкриття файла. Оператори роботи з файлами:

Open <ім'я > For <режим> AS # <номер>

де **<ім'я>** — рядок або рядкова змінна, яка містить ім'я файла;

<режим> визначає режим доступу до файла:

Input — файл відкривається для читання. Він повинен існувати;

Output — файл створюється для запису із самого початку. Якщо файл з таким ім'ям існує, він знищується;

Append — файл відкривається для запису в його кінець (тобто для поновлень);

Random — файл складається із записів фіксованої довжини і відкритий як для запису, так і для читання;

<номер> — номер файла (канал введення/виведення). Надалі звернення до файла здійснюється за цим номером.

Запис у файл. Оператори роботи з файлами:

Write # <номер>, <список змінних>

При записуванні на диск значення змінних відокремлюються одне від одного комами, а записи — символами повернення каретки і переведення рядка.

Читання з файла. Оператор роботи з файлами:
Input # <номер>, <список змінних>
Читає дані з файла і роміщує їх у змінні.

Закриття файла. Оператор ***роботи з файлами:***
Close # <номер>

Для розпізнавання кінця файла використовується функція **EOF()**. Вона застосовується з аргументом <**номер**>, що являє собою номер файла, і повертає логічне значення ″**істина**″, якщо досягнути кінця файла, і ″**хибність**″ в іншому випадку.

Список літератури

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

2 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 Меркулов, В. С. Програмування в середовищі Visual Basic 6.0 [Текст] : конспект лекцій / В. С. Меркулов, І. Г. Бізюк, О. В. Чаленко. — Харків : УкрДАЗТ, 2011. Ч. 1. — 72 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 Грэхем, И. Объектно-ориентированные методы: принципы і практика [Текст] / И. Грэхем. – 3-е изд. — М. : Вильямс, 2004. — 880 с.

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

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

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

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