О.Н. Романюк, А.В. Денисюк

Організація баз даних і знань

Міністерство освіти і науки України
Вінницький національний технічний університет

О.Н. Романюк, А.В. Денисюк

Організація баз даних і знань

Затверджено Вченою радою Вінницького національного технічного університету як лабораторний практикум для студентів напряму підготовки 0804 - "Комп'ютерні науки" спеціальності 7.080403 "Програмне забезпечення автоматизованих систем". Протокол № 5 від 26 грудня 2002 р.

Вінниця ВНТУ 2004

УДК 681.3
Р 96

Рецензенти:

А.Т. Теренчук, кандидат технічних наук, доцент
В.І. Месюра, кандидат технічних наук, доцент
В.П.Кожем'яко, доктор технічних наук, професор

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

Романюк О.Н., Денисюк А.В.
Р 96 **Організація баз даних і знань.** Лабораторний практикум. - Вінниця: ВНТУ, 2004. - 63с.

В практикумі розглянута сучасна СУБД Visual FoxPro, наведено теоретичні відомості і показано застосування на конкретних прикладах для використання в лабораторних роботах. Посібник розроблений у відповідності з планом кафедри та програми до дисципліни "Організація баз даних і знань".

УДК 681.3

О.Н. Романюк, А.В. Денисюк

Організація баз даних і знань

Інтерфейс, головне меню, панель інструментів, меню, редагування даних, поле, таблиця, запис, база даних, тип даних, змінна, масив, команда, вікно, синтаксис.

21.01.04

_____ (О.Н. Романюк)

_____ (А.В. Денисюк)

Навчальне видання

Олександр Никифорович Романюк
Алла Василівна Денисюк

Організація баз даних і знань

Лабораторний практикум

Оригінал-макет підготовлено Денисюк А.В.

Редактор С.А. Малішевська

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

Підписано до друку
Формат 29,7х42 1/4
Друк різографічний
Тираж прим.
Зам. №

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

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

ЗМІСТ

ВСТУП

Основна ідея сучасної інформаційної технології базується на концепції баз даних (БД). Відповідно до цієї концепції, основою інформаційної технології є дані, які організовані в БД із метою адекватного відображення реального світу і задоволення інформаційних потреб користувачів.

Збільшення об'єму і структурної складності збереження даних, розширення кола користувачів інформаційних систем висунуло вимогу створення зручних загальносистемних засобів інтеграції збережених даних і керування ними. Це привело до появи промислових систем керування базами даних (СКБД) – спеціалізованих програмних засобів, призначених для організації і ведення БД.

Завдяки досягненням в області штучного інтелекту з'являються системи, що базуються на використанні знань. Систему, яка забезпечує створення, ведення і застосування баз знань, можна розглядати як інструментальну систему або прикладну систему з конкретною прикладною базою знань. Існує тісний взаємозв'язок між технологією баз даних і систем баз даних – з одного боку, і технологією систем баз знань з іншого. Виникла тенденція „інтелектуалізації" систем БД. На зовнішньому рівні їх архітектури реалізують різноманітні систематичні моделі даних, створюють „дружні" інтерфейси для користувачів, хоча традиційні СКБД є необхідною складовою частиною інструментарію керування даними в системах баз знань.

В лабораторному практикумі приведені необхідний теоретичний матеріал для виконання лабораторних робіт з курсу „Організація баз даних і знань", завдання до лабораторних робіт та контрольні запитання.

Лабораторна робота №1

Тема: Огляд можливостей і особливостей різних СУБД**.**
Мета: Порівняння засобів розробки Microsoft.

Теоретичні відомості

Огляд можливостей і особливостей різних СУБД в англомовній комп'ютерній літературі має дуже лаконічну абревіатуру – *RAD* (*Rapid Application Development*) і все частіше зустрічається на сторінках спеціалізованих видань. Що це таке? Це черговий етап, при чому етап революційний, розвитку інформаційних технологій. Природна реакція комп'ютерної індустрії на інформаційні потреби суспільства, що швидко розвиваються .

У світі використовуються десятки мільйонів персональних комп'ютерів, і їхнє число постійно росте. Комп'ютери застосовуються в тих областях, де про них не думали ще рік тому. Комп'ютери починають витісняти навіть такі, здавалося б, непорушні атрибути цивілізації, як телевізори й іншу звичну для нас побутову техніку. А збільшення числа і розширення сфери застосування комп'ютерів викликає потребу у збільшенні кількості програмного забезпечення. Єдиний шлях, що відведе від необхідності перетворити все працездатне людство в програмістів, – різке підвищення ефективності засобів розробки програм.

Ця ідея і втілюється в сучасних версіях пакетів програм для створення систем автоматизації обробки даних, що відповідають вимогам *RAD*. Можна виділити такі особливі риси цих засобів розробки:

- Наявність об'єктно-орієнтованої мови програмування, що дозволяє дуже ефективно використовувати модульний принцип складання програм.
- Візуальні засоби розробки, що надають можливість замінити написання програмного коду рисуванням користувацького інтерфейсу і заданням необхідної функціональності діалоговими засобами.
- Підтримка стандартних протоколів обміну даними між додатками, що дозволяє розробляти багаторівневі додатка, що не залежать від джерела даних. Тут же закладена можливість застосування компонентної технології створення додатків.
- Можливість створення додатків клієнт-сервер, що дозволяє розробляти додатки необмеженої складності і забезпечувати потреби цілого підприємства в обробці даних.

Жоден будівельник не побудує будинок швидше маляти, що складає його з кубиків. Задача сучасного засобу розробки – дати нам багато різних кубиків. Задача програміста – взяти потрібний кубик і поставити його в потрібне місце. Це основна ідея *RAD*.

Перелік сучасних засобів розробки систем автоматизації обробки даних, у яких закладені ідеї *RAD*, дуже велика. Майже кожен місяць з'являються нові версії цих продуктів тієї чи іншого фірми-виробника програмного забезпечення. Вони включають усе нові і нові можливості, що полегшують працю програміста. «Чому ж Microsoft ?»

- Microsoft – найбільша і на даний момент найбільш вдала фірма-виробник програмного забезпечення;
- Програмами Microsoft користуються десятки мільйонів людей в усьому світі;
- Засоби розробки Microsoft відмінно інтегровані між собою, підтримують усі сучасні протоколи обміну даними, і тому завжди можна використовувати найбільш ефективний у конкретній ситуації пакет програм.

Розглянемо таблицю 1.

Таблиця 1 - Порівняння засобів розробки Microsoft

Назва продукту	Основні переваги	Основне призначення
Access	Простота освоєння. Можливість використання непрофесійним програмістом. Має потужні засоби підготовки звітів з БД різних форматів.	Створення звітів довільної форми на підставі різних даних. Розробка некомерційних додатків.
SQL - Server	Високий ступінь захисту даних. Потужні засоби роботи з даними. Висока продуктивність.	Збереження великих масивів даних. Збереження даних, що вимагають дотримання режиму таємності або недопустимості їхньої втрати.
Visual Basic	Універсальність. Можливість створення компонентів OLE. Невисокі вимоги до потужності ПЭВМ.	Створення додатків середньої потужності, не зв'язаних з великою інтенсивністю обробки даних. Розробка компонентів OLE. Створення додатків для інтеграції компонентів Microsoft Office.
Visual C++	Універсальність. Найбільша швидкість роботи додатку. Необмежена функціональність.	Створення компонентів додатка для виконання критичних за швидкістю процесів або за безпечення функціональності, недосяжної в інших засобах розробки.
Visual FoxPro	Високий рівень об'єктивної моделі. Висока швидкість обробки даних. Інтеграція об'єктно-орієнтованої мови програмування з xBase і SQL. Багатоплатформність.	Створення додатків масштабу підприємства. Створення додатків для роботи на різних платформах (Windows 3.X, Windows 95, Macintosh і т.д.)

Звичайно, при спільному використанні різних засобів розробки додатків нас більше всього будуть цікавити дані. У таблиці 2 наведено перелік типів даних, доступних у розглянутих засобах розробки. Прочерки в двох останніх колонках таблиці означають, що для цього типу даних задання конкретних величин не потрібне.

Таблиця 2 - Типи даних

Тип даних	Visual FoxPro	Access і Visual Basic	MS SQL Server	Довжина (у байтах)	Число десяткових розрядів	Займаний обсяг
Binary Image	Немає	db Long Binary	вynary (n)	n	–	до 1,2 Гбайт
Byte	Немає	db Byte	tinyint	1	–	1 байт
Character Text	C	db Text	char (n), varchar (n)	n	–	4 байти
Count	Немає	db Long	Немає	–	–	4 байти
Currency	Y	db Currency	money	–	–	8 байт
Date	D	Немає	Немає	–	–	8 байт
Date Time	T	db Date	datetime	–	–	8 байт
Logical (Yes/No)	L	db Boolean	bit	–	–	1 байт
Numeric	N	Немає	float	n	d	від 1 до 20 байтів
Integer Integer	I	db Integer db Long	smallint int	– n	– –	2 байти 4 байти
Double	B	db Double	float	–	d	8 байт
Float	F		float	n	–	від 1 до 20 байтів
General (OLE Object)	G	db Long Binary	image	–	–	4 байти
Memo	M	db Memo	text	–	–	4 байти
Single		db Single	real	–	–	4 байти
Character (binary)	C	Немає	Немає	n	–	1 байт на символ
Memo (binary)	M	Немає	Немає	–	–	4 байти

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Дати порівняльний аналіз засобів розробки Microsoft.
3. Ознайомитися з типами даних, доступних у розглянутих засобах розробки.
4. Підготувати звіт по лабораторній роботі.

5. Зробити висновки по лабораторній роботі, захистити лабораторну роботу.

Контрольні питання

1. Які особливі риси засобів розробки програм?
2. Чому використовують засоби розробки Microsoft?
3. Дайте порівняльну характеристику засобам розробки Microsoft?
4. Які основні типи даних, доступні у засобах розробки Microsoft ?

Лабораторна робота №2

Тема: Вивчення головного вікна **Visual FoxPro**.

Мета: Набуття навички роботи з головним вікном **Visual FoxPro**.

Теоретичні відомості

Visual FoxPro – не просто наступна версія однієї з найбільш швидких СУБД для персональних комп'ютерів. Це зовсім нова програма, що дозволяє легко зробити те, що в попередніх версіях здавалося дуже складним чи було просто недоступно.

Інтерфейс **Visual FoxPro** відповідає уявленням про сучасне графічне середовище, нагадуючи інтерфейс інших програм Microsoft, робить роботу інтуїтивно зрозумілою. Основна робота з даними в **Visual FoxPro** виконується за допомогою різних інструментальних засобів, тому команди меню часто мають допоміжний характер і їхній склад гнучко змінюється в залежності від того, який засіб активний у даний момент.

Дамо основну характеристику командам меню.

Головне вікно **Visual FoxPro** включає:

1) розташоване ліворуч угорі – головне меню;
2) розташовану праворуч угорі – стандартну панель інструментів;
3) розташований ліворуч унизу – статус-рядок для висновку оперативної інформації;
4) вікно команд, розташування і розмір якого можна змінювати за допомогою миші.

Головне меню містить у собі:
File, Edit, View, Format, Tools, Program, Window, Help.

Меню ***File*** включає основні функції для роботи з файлами:

- **New** – створює новий файл. Тип створюваного файлу можна вибрати з діалогового вікна, що з'являється. Розглянемо його:
  File Type – вибір створюваного файлу чи компонента:

Project – проект;
Database – база даних;
Table – таблиця;
Query – запит;
Connection – зв'язок;
View – перегляд;
Remote View – віддалений перегляд;
Form – форма;
Report – звіт;
Label – ярлик, мітка;
Program – програма;
Class – клас;
Text File – текстовий файл;
Menu – меню.

NewFile – створення файла.
Wizard – виклик майстра.
Cancel – скасування.
Help – виклик довідки.

- **Open** – відкриває існуючий файл.
- **Close** – закриває активне вікно. Якщо ви утримуєте клавішу ***Shift***, закриваються усі відкриті вікна.
- **Save** – зберігає зміни, зроблені в активному файлі.
- **Save as** – зберігає активний файл з іншим ім'ям.
- **Revert** – скасовує всі зміни, зроблені в активному файлі з моменту останнього збереження.
- **Import** – дозволяє імпортувати дані в таблицю **Visual FoxPro**.
- **Export** – дозволяє експортувати дані з **Visual FoxPro** в інші формати.
- **Page Setup** – викликає діалогове вікно для встановлення параметрів сторінки звіту чи ярлика.
- **Print Preview** – дозволяє переглянути на екрані результат виконання звіту чи ярлика.
- **Print** – виводить на друк чи у файл текстовий файл, програму, звіт, ярлик, вміст вікна ***Command*** буфера чи обміну (***Clip-board***).
- **Send** – дозволяє відправити електронну пошту за наявності на комп'ютері відповідних засобів.
- **Exit** – закриває **Visual FoxPro**.

Меню ***Edit*** дозволяє виконувати редагування програмного коду чи будь-якого іншого тексту, а також полегшує роботу з елементами керування й об'єктами у формах, звітах і т.д.

- **Undo** – скасовує останню виконану дію.
- **Redo** – відновлює останню скасовану дію.
- **Cut** – видаляє і записує в буфер обміну виділений фрагмент чи елементи керування.
- **Copy** – записує в буфер обміну виділений фрагмент чи елементи керування.
- **Paste** – копіює з буфера обміну дані, що там зберігаються, у місце розташування курсора.
- **Paste Special** – зв'язує чи вбудовує з буфера обміну OLE - об'єкти, що там зберігаються, в місце розташування курсора.
- **Clear** – видаляє виділений фрагмент чи елементи керування.
- **Select All** – виділяє весь текст чи елементи керування в активному вікні.
- **Find** – виводить діалогове вікно пошуку фрагмента тексту.
- **Replace** – виводить діалогове вікно пошуку фрагмента тексту і його заміни на інший об'єкт.

- **Go To Line** – виконує швидкий перехід на зазначений номер рядка в текстовому файлі.
- **Insert Object** – виводить діалогове вікно зі списком об'єктів, які можна вмонтувати в активну форму чи поле типу, що ***редагується,*** General.
- **Object** – виводить список дій, підтримуваних активним об'єктом (наприклад, при перегляді вмісту поля типу ***General***).
- **Links** – виводить діалогове вікно для редагування чи видалення зв'язку з активним об'єктом.

Меню ***View*** дозволяє керувати появою на екрані допоміжних засобів, наприклад, панелей інструментів, і виводити на екран дані. За відсутності на екрані будь-яких засобів розробки в цьому меню доступна тільки одна команда – **Toolbars**, що виводить діалогове вікно для вибору розташованих на екрані панелей інструментів.

Меню ***Format*** з'являється завжди, коли на екрані є активне вікно, і дозволяє змінювати зовнішній вигляд відображуваних даних за допомогою таких команд:

- **Font** – виводить діалогове вікно зміни шрифту і його характеристик.
- **Enlarge Font** – збільшує розмір відображуваного в активному вікні тексту.
- **Reduce Font** – зменшує розмір відображуваного в активному вікні тексту.
- **Single Space** – встановлює одиничний міжрядковий інтервал для відображуваного в активному вікні тексту.
- **1½ Space** – встановлює полуторний міжрядковий інтервал для відображуваного в активному вікні тексту.
- **Double Space** – встановлює подвійний міжрядковий інтервал для відображуваного в активному вікні тексту.
- **Indent** – дозволяє зрушити вправо лінію або кілька ліній тексту на один інтервал табуляції у вікні редактора чи вікні ***Command.***
- **Remove Indent** – дозволяє зрушити вліво лінію чи кілька ліній тексту на один інтервал табуляції у вікні редактора чи вікні ***Command.***

Меню ***Tools*** дозволяє виконати різні допоміжні дії:

- **Wizards** – запускає один з наявних Майстрів.
- **Spelling** – запускає програму перевірки правопису для вмісту текстового файла чи поля приміток (можна використовувати для перевірки правильності написання команд).
- **Macros** – дозволяє присвоїти клавіатурній комбінації виконання деякої дії чи набору дій.
- **Class Browser** – виводить на екран утиліту роботи з класами.

- **Trace Window** – виводить на екран вікно для візуального відображення виконуваного програмного коду.
- **Debug Window** – виводить на екран вікно для відображення поточних значень у процесі виконання програми.
- **Options** – виводить на екран діалогове вікно для установки параметрів конфігурації середовища розробки.

Меню ***Program*** містить команди, пов'язані з виконанням програм:

- **Do** – запускає програму на виконання.
- **Cancel** – закінчує виконання.
- **Resume** – продовжує виконання програми з рядка, на якому вона була припинена командою ***Suspend***.
- **Suspend** – припиняє виконання програми без вивантаження її з пам'яті, залишаючи можливим продовження її роботи командою ***Resume***.
- **Compile** – компілює програму в псевдокод.

Меню ***Window*** містить команди керування вікнами:

- **Arrange All** – розташовує усі відкриті вікна на екрані так, щоб кожне було видиме.
- **Hide** – ховає активне вікно.
- **Clear** – стирає вміст активного вікна.
- **Circle** – виконує перехід до наступного відкритого вікна.
- **Command Window** – робить активним чи відкриває вікно ***Command***. Серед розглянутих засобів розробки це вікно є унікальним, тому що дозволяє негайно виконувати майже всі команди **Visual FoxPro** і, відповідно, бачити результат їхньої роботи.
- **View Window** – робить активним чи відкриває діалогове вікно ***View***, що містить основний інструментарій для роботи з даними.

Меню ***Help*** містить команди, що дозволяють швидко одержати необхідну інформацію про роботу з **Visual FoxPro.**

Особливості **Visual FoxPro** можна описати в такий спосіб:

1. Забезпечення можливості швидкої розробки прикладної програми базується на включенні засобів, що дозволяють підвищити швидкість роботи програміста. У першу чергу це засоби об’єктно-орієнтованого програмування, що дозволяють користувачу формувати компоненти свого проекту (об'єкти), що потім можуть багаторазово використовуватися.

У зв'язку з цим, традиційна ***xBase*** мова в **Visual FoxPro 3.0** значно розширена, що дозволяє створювати справжні об'єкти, класи і підкласи. Крім того, об'єкти можуть бути створені за допомогою візуальних засобів і багаторазово використані в будь-який час.

2. Забезпечення повного набору засобів для керування подіями. Традиційно в ***xBase*** програмісту доводилося писати власний драйвер для обробки необхідного набору подій чи покладатися на READ-стан чекання, що моделює обробку події системою. У Windows число подій, до яких може звертатися користувач, дуже велике, і, отже, обробка подій є непростою задачею. **Visual FoxPro 3.0** має істинно керовану подіями модель, так що за замовчуванням система раніш, ніж користувачі, обробляє об'єктні події.

Крім того, програміст тепер має повний доступ до набору стандартних, основаних на функціонуванні Windows, подій (наприклад, за допомогою миші допускається перетягування об'єктів).

3. Забезпечення потужного інструментального набору засобів для програміста. Розроблювачі систем автоматизації обробки даних крім могутнього набору візуальних засобів проектування можуть використовувати широкі можливості з інтеграції систем збереження даних і доступу до серверів даних за допомогою технології ODBC (Open Database Connectivity – Відкритий Доступ до Баз Даних).

Основні нововведення – це розширення вбудованої мови SQL, можливість відновлення даних на сервері через редагування курсорів, вбудований механізм забезпечення транзакцій, можливість звертання до сервера на тому діалекті SQL, що підтримує сервер. Наявність словника даних робить більш швидкою розробку структури баз даних і полегшує її подальшу експлуатацію і підтримку.

4. Забезпечення повної інтеграції **Visual FoxPro 3.0** у сімейство прикладних програм ***Microsoft***. Єдиний інтерфейс із найбільш популярними прикладними програмами ***Microsoft*** робить роботу в інтерактивному режимі інтуїтивно зрозумілою. Підтримка правої кнопки миші дозволяє уникнути довгих подорожей по системі меню і значно полегшує вивчення нових можливостей СУБД. Просто виберіть курсором об'єкт і натисніть праву кнопку миші! На деяких діалогових вікнах, що часто використовуються в роботі на смузі заголовка, з'явиться перемикач у вигляді анімаційної піктограми (push pin), що дозволяє легко включити режим, при якому це вікно буде завжди розташовано на передньому плані.

Visual FoxPro забезпечує повну підтримку ***OLE 2.0***, що полегшує взаємодію з іншим програмним забезпеченням у середовищі ***Windows***.

Крім залишеної можливості завантаження зовнішніх функцій за допомогою ***команди SET LIBRARY***, з'явилася можливість звертання до функцій динамічних DLL-бібліотек Windows за допомогою ***команди DECLARE***.

5. Сумісність з раніше розробленим програмним забезпеченням у середовищі **Visual FoxPro**.

У **Visual FoxPro** система організації баз даних найбільш близька до теоретичних основ реляційної моделі і дозволяє більш природно виконувати операції реляційної алгебри. Основна одиниця збереження

даних – це таблиця, у стовпцях і рядках якої зберігаються дані, як це і було раніш у DBF-файлі. Таблиця зберегла розширення файлу DBF і має пряму сумісність з «старими» DBF-файлами. Таблиці об’єднуються в базу даних, у якій можна описати всі зв'язки, що встановлюються між полями окремих таблиць, правила перевірки, що будуть визначати реакцію системи на внесені зміни, додавання чи видалення даних і правила перевірки цілісності даних у БД.

Файли баз даних мають розширення DBC і при відкритті автоматично підтримують усі перераховані установки для таблиць, що до неї входять. При необхідності можна мати і таблиці, що не входять у БД, – вільні таблиці. **Visual FoxPro** забезпечує підтримку значень ***NULL*** і виконання операцій з цими даними у відповідності зі стандартом *ANSI*. Це полегшує задачу представлення невідомих даних і взаємодію з MS Access і БД.

Крім формування структури представлення інформації, він виконує функції словника даних за рахунок підтримки наступних функціональних можливостей:

- Допускаються довгі імена таблиць.
- Кожному полю і таблиці можна давати коментарі.
- Допускається використання довгих імен полів.
- Для полів крім ідентифікаторів можна використовувати заголовки, що можуть використовуватися як у вікні ***Browse***, так і як заголовки для колонок таблиці в об'єкті ***Grid***.
- Введено значення за замовчуванням для полів.
- Передбачено правила перевірки для полів і записів при зміні і введенні нових даних.
- Маються тригери для підтримки цілісності даних.
- Підтримуються постійні зв'язки між таблицями, розміщеними в БД.
- Маються процедури БД для описання складних умов правил перевірки.
- Можна використовувати з'єднання для зв'язку з зовнішніми джерелами даних.
- Підтримуються локальні і зовнішні перегляди.

Основними засобами редагування даних в оболонці **FoxPro** є повноекранні засоби ***Browse*** і ***Edit*** (***Change***). ***Browse*** дозволяє редагувати дані в найбільш звичному для користувача вигляді – табличному, а ***Edit*** – у вигляді колонки полів. Для перегляду даних з таблиці, відкритої в будь-якій робочій області, відкривається окреме вікно. Для перегляду чи редагування даних у таблиці досить відкрити потрібну таблицю і в меню ***View*** вибрати команду ***Browse***. Цей спосіб візуалізації даних дуже зручний, тому що дозволяє переглядати відразу кілька записів, але, як правило, рідко коли всі поля таблиці містяться одночасно у вікні, навіть розкритому на весь екран.

Перегляд даних у вигляді ***Edit*** дозволяє працювати відразу з усіма даними в одному записі.

Переключатися між цими двома видами перегляду даних можна за допомогою відповідних команд у меню ***View***. При цьому робота з даними не переривається, змінюється тільки вид подання інформації. Для введення і редагування даних можуть використовуватися прийоми, звичайно застосовувані під час роботи з даними в програмах для Windows. Використовуйте можливості запропоновані в меню ***Edit***. Натискання на клавішу ***Tab*** чи ***Enter*** приводить до переміщення курсора в наступне поле, а для повернення в попереднє зручно використовувати сполучення клавіш ***Shift + Tab***.

Врахуйте, що при досягненні курсором останнього символу в полі під час введення даних, **Visual FoxPro** за замовчуванням подає звуковий сигнал і переводить курсор у наступне поле.

Вікна ***Browse*** чи ***Edit*** є могутніми засобами перегляду і редагування даних. Додаткові зручності для досягнення найвищої зручності роботи закладені в меню ***Table***. Що пропонують нам команди цього меню?

- **Properties** – виводить на екран діалогове вікно, що дозволяє встановити характеристики для таблиці, що відкрита в даній робочій області.
- **Font** – викликає стандартне діалогове вікно Windows, що дозволяє вибрати зручний шрифт і підібрати його характеристики.
- **Go To Record** – дозволяє швидко перейти до потрібного запису, вибравши одну з таких опцій:

  Top – на перший запис (вершина таблиці);
  Bottom – на останній запис;
  Next – на наступний після поточного запису;
  Previous – на попередній перед поточним записом;
  Record # – на запис із зазначеним номером.

  Не забудьте, що дані у вікні перегляду розташовуються в порядку їхніх номерів, тільки якщо ви не використовуєте якийсь індекс. Опція ***Locate*** дозволяє знайти необхідний запис за його вмістом, указавши відповідний вираз для пошуку.

- **Append New Record** – додає в таблицю один новий запис.
- **Toggle Deletion Mark** – позначає для видалення поточний запис чи забирає цю відмітку, якщо поточний запис вже позначений для видалення.
- **Append Records** – дозволяє перейти в режим додавання записів, в якому новий запис буде автоматично додаватися після введення даних у поточний. Доданий запис буде збережений, якщо ви ввели в нього хоча б один символ. Ми наполегливо рекомендуємо додавати дані в таблицю саме цим методом, тому

що він дозволяє уникнути появи в таблиці великої кількості непотрібних порожніх записів.

- **Delete Records** – виводить діалогове вікно, що дозволяє вказати запис, який необхідно позначити для видалення.
- **Recall Records** – виводить діалогове вікно, що дозволяє вказати записи, в яких необхідно забрати позначки для видалення.
- **Remove Deleted Record** – запускає команду **PACK** для фізичного видалення позначених для цього записів.
- **Replace Field** – дозволяє вказати записи, дані в який потрібно замінити на зазначене значення.
- **Size Field** – дає можливість змінити ширину колонки у вікні для виведення даних з поточного поля. При цьому ви змінюєте ширину колонки, а не довжину поля в таблиці. При досягненні потрібної ширини колонки необхідно натиснути клавішу ***Enter***. Якщо ширина колонки менша, ніж довжина поля в таблиці, дані будуть прокручуватись при переміщенні курсора всередині колонки.
- **Move Field** – дозволяє перемістити поточну колонку у вікні.
- **Resize Partitions** – дозволяє змінити розміри частин вікна перегляду чи розбити це вікно на дві частини, якщо це не було зроблено раніше. Розбиття вікна дозволяє залишити на екрані будь-яке поле чи поля при горизонтальному прокручуванні даних. Це дуже зручно під час роботи з таблицями, що мають велику кількість полів.
- **Link Partitions** – дозволяє синхронізувати переміщення по записах таблиці незалежно від того, у якій частині вікна ми переміщаємо записи. Під час скасування цієї тригерної команди в кожній частині вікна перегляду записи будуть переміщатися незалежно.
- **Change Partitions** – забезпечує перехід з однієї частини вікна в іншу без використання мишки.
- **Rebuild Indexes** – дозволяє в разі потреби привести індекси відповідно до даних у таблиці.

Для роботи з даними, розташованими в полях приміток, досить два рази клацнути мишкою в потрібній ділянці вікна чи перемістити туди курсор і натиснути клавіші ***Ctrl + PgDn***.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Практично перевірити можливості роботи з головним меню.
3. Створити базу даних з 10 записів і 5 полів.

4. Виконати заповнення бази даних.
5. Відредагувати наявні записи.
6. Підготувати звіт по лабораторній роботі.
7. Зробити висновки по лабораторній роботі, захистити лабораторну роботу.

Контрольні питання

1. Що містить головне вікно **Visual FoxPro** ?
2. Що містить головне меню **Visual FoxPro** ?
3. Які особливості **Visual FoxPro** ?
4. Охарактеризуйте систему організації баз даних у **Visual FoxPro** ?
5. Які основні функціональні можливості **Visual FoxPro** ?
6. Які засоби редагування даних в оболонці **Visual FoxPro** ?

Лабораторна робота №3

Тема: Основи мови програмування **Visual FoxPro**.

Мета: З’ясувати з чого складається мова програмування **Visual FoxPro** і як зберігають дані, як виконувати дії з даними, як працювати зі змінними і масивами.

Теоретичні відомості

Мови програмування мають досить багато загальних рис, але одна з них – багата історична спадщина – привела до того, що сучасні об'єктно-орієнтовані властивості в них співіснують із традиційними структурними складовими. Причому, число команд і функцій, що складають структурну основу мови вже більше тисячі. У **Visual FoxPro** можна символьні значення вказувати не тільки в лапках, але й у квадратних дужках.

Мова програмування являє собою набір команд, що послідовно обробляються *інтерпретатором і перетворюються ним у машинний код*, у свою чергу оброблюваний мікропроцесором. За допомогою команд ми виконуємо деякі дії, аналогічно вибору команди в меню. Типова структура команди:

COPY TO File Name [FIELDS FieldList][Scope][FOR/Expression].

Назва команди є ключовим елементом для її ідентифікації. У зв'язку з тим, що розроблювачі мови намагалися дати назвам команд максимальне змістовне навантаження для більш легкого їх запам'ятовування, у ряді випадків команди вийшли досить громіздкими. У **Visual FoxPro** назви команд у більшості випадків можна скоротити до чотирьох символів. Якщо в якомусь випадку так робити не можна через небезпеку втратити унікальність ідентифікації, про це обов'язково буде написано в довідковому файлі **Visual FoxPro** при описанні даної команди. Опції служать для уточнення дії, виконуваної командою. У даному прикладі можна використовувати опцію ***Fields*** для вказівки списку тих полів, що будуть копіюватися в новий файл.

Частина команди, позначена словом ***Scope***, дозволяє задати діапазон записів, на які буде впливати команда. Якщо ми використовуємо цю можливість, то в команді замість слова **Scope** треба використовувати один з перерахованих варіантів:

ALL – усі записи в таблиці;

NEXT n Records – указує число записів після поточного (включно з поточним);

RECORD n Record Number – запис із зазначеним номером;

REST – записи від поточного до кінця таблиці.

Умова виконання команди ***FOR*** дозволяє визначити виконання команди в залежності, наприклад, від змісту полів. Для обробки інформації ми можемо використовувати дані, що зберігаються в таблицях чи у пам'яті

комп'ютера. В останньому випадку для даних у пам'яті, що називаються змінними, ми повинні визначити ідентифікатор, посилаючись на який можна однозначно встановити, які дані ми маємо на увазі.

Для збереження в пам'яті однорідних даних використовуються масиви.

Таблиця 3 - Способи подання даних

Спосіб подання	Опис
Константи	Одиничні елементи даних, записані в програмному коді і незмінні в процесі роботи
Змінні	Одиничні елементи даних, що зберігаються в оперативній пам'яті (ОЗП)
Масиви	Безліч елементів даних, що зберігаються в ОЗП
Записи в таблицях	Безліч рядків, що містять заздалегідь визначені поля, кожне з визначеним фрагментом даних, що зберігаються у файлі таблиці.

Константи часто використовуються для включення у вираз якихось визначених даних, що не змінюються під час роботи програми. Як константи ми можемо використовувати дані різних типів:

- Символьні дані записуються в лапках. Наприклад, ми можемо запам'ятати слово *ім'я*, використовуючи його вказівку в лапках: "*Ім'я"*.
- Дані типу "*дата"* чи "*дата і час"* у **Visual FoxPro** записуються у фігурних дужках: *{10/10/95}*/
- Числові дані використовуються без будь-яких роздільників.
- Логічні дані як константи можуть приймати одне з двох значень. У **Visual FoxPro** під час записування вони обмежуються крапками:

  .*T.* - відповідає істині (True),

  .*F.* - відповідає хибності (False).

Для роботи з даними використовуються оператори, перелік яких приведений у таблиці 4.

Пам'ятайте, що з **одним оператором** необхідно використовувати **дані одного типу**! Багато операторів з приведених у таблиці 4 знайомі нашим читачам зі шкільної лави. А для деяких з цих операторів потрібне докладне пояснення.

- Оператор **$** використовується для пошуку символу чи набору символів у будь-якому символьному виразі чи полі приміток. Шуканий символ вказується зліва від цього оператора.
- Оператор = = аж ніяк не той же саме, що оператор =. Якщо останній можна ідентифікувати словом "дорівнює", то для першого більше підходить назва "ідентичне". Наприклад, якщо ми будемо шукати дані за умовою cName = «Іван», то **Visual FoxPro** буде вважати правильним

результат, якщо знайде значення «Іванов», «Іваненко», так само як і зазначене слово «Іван». Якщо ж у вираженні для пошуку вказати оператор «= =», то до правильного результату приведе тільки знайдене слово «Іван». Це розходження дуже зручно використовувати для пошуку даних за неповною відповідністю, коли точно невідомо шукане значення.

Таблиця 4 - Знаки операцій.

Оператор	Припустимі типи даних	Приклад	Результат дії оператора в прикладі	Примітки
=	Усі	n=7	Число 7 запам'ятовується в змінній n	
+, –	N, C, D	{10/10/95}+10 “Fox” + “Pro”	{20/10/95} “FoxPro”	
^, *, /, ()	N	5*5 2^3 (4–3)+56	25 8 57	
NOT, AND, OR, ()	L	NOT .T. .F. OR (.T. AND .F.)	.F. .F.	Дужки – оператор групування
<, >, <=, >=, <>	Усі	23<54	.T.	Повертає логічне значення перевірки відносин
$	C	“m”$cName	.T., якщо буква m є в рядку, що міститься в змінній cName	Тільки в Visual FoxPro
= =	C	cTurn = = «Андрій»	Порівняння на повну тотожність двох рядків	Тільки в Visual FoxPro

Поля, змінні, константи, функції та оператори є елементами під час складання виразів. Під час написання виразів необхідно враховувати пріоритет виконання операцій:

1. Піднесення до степеня.
2. Множення і ділення.
3. Додавання і віднімання.
4. Додавання символьних виразів.
5. Операції порівняння.
6. NOT
7. AND
8. OR

Якщо необхідно змінити порядок дій, то варто використовувати дужки. Змінні в програмі мають чітко визначені області дії і за цією ознакою поділяються на три типи.

- ***Локальні змінні*** **(private)** існують тільки під час роботи процедури чи програмного файла (модуля), у якому вони були визначені. За замовчуванням усім змінним, обумовленим у програмі, присвоюється статус локальних. Явно визначити статус змінної в **Visual FoxPro** можна командою:

  PRIVATE MemVarList | PRIVATE ALL [LIKE Sceleton | EXCEPT Sceleton].

Важлива особливість локальних змінних полягає в тому, що якщо вони були визначені з тими ж іменами, що були оголошені раніше (у програмі більш високого рівня), то після завершення роботи програми "старі" значення будуть відновлені. Таким чином, локальні змінні, створені в будь-якій процедурі, будуть діяти і у всіх процедурах і функціях викликаних з неї, але перестануть бути доступними, як тільки ми вийдемо з процедури, що їх створила, на більш високий рівень. Опції команди в **Visual FoxPro** дозволяють не приводити повний список усіх змінних, а використовувати для їхньої ідентифікації шаблон, що за допомогою символів **?** і ***** вкаже поширення дії команди на змінні, імена яких відповідають шаблону (**LIKE**), чи на змінні, імена яких не відповідають приведеному шаблону (**EXCEPT**). Нагадаємо, що шаблон зазвичай включає загальну частину імені і знаки заміщення ? і *, де * – заміняє будь-яку кількість символів імені, а ? – заміняє один символ імені. Це дозволяє знаком * задавати усі імена, а декількома знаками ? імена відповідної довжини.

При великій кількості однотипних змінних більш ефективно використовувати замість них ***масиви***, тобто одно- чи двовимірні таблиці змінних під загальним ім'ям. Звертання до елементів масиву відбувається за допомогою вказівки після імені масиву в круглих чи квадратних дужках потрібних номерів рядків і (чи) стовпців. Перед використанням масивів у **Visual FoxPro** їх треба оголосити командою:

DIMENSION ArrayName1 (nRows1 [,nColumns1]) [, ArrayName2 (nRows2 [,nColumns2])]...

Кожен масив може містити не більш 65000 елементів, тобто гранична розмірність двовимірного масиву може скласти 254×255, 1000×65 і т.п. В останньому випадку nRows1=1000 і nColumns1=65, якщо значення nColumns не визначене, то масив одновимірний. Після оголошення масиву за замовчуванням усім його елементам присвоюється логічний тип даних зі значенням .F.

Для ***задання типу даних***, а це відноситься і до змінних в **Visual FoxPro**, необхідно використовувати команду:

STORE Expression TO MemVarList / ArrayList

чи використовувати присвоєння:

MemVar / Array = Exoression.

Якщо вказується ідентифікатор масиву без аргументів, то зазначене значення **Expression** присвоюється всім елементам масиву. Наприклад:

```
DIMENSION aMembers (4,2)
STORE 5 TO aMembers.
```

Для ***визначення змінної типу дати*** в **Visual FoxPro** можна використовувати команду, що одночасно присвоє їй нульове значення:

```
STORE CTOD ('. .') TO dDate
```

або

```
STORE {} TO dDate.
```

Для присвоєння конкретного значення краще скористатися варіантом з фігурними дужками, наприклад:

```
SET DATE GERMAN
STORE {01.01.91} TO dDate.
```

Для форматування значення дати в **Visual FoxPro** зручно використовувати установочну команду **SET DATE**.

- ***Глобальні змінні*** **(public)** будуть зберігати свої значення у всіх програмних файлах і викликаних ними процедурах. Для оголошення змінних і елементів масиву глобальними в **Visual FoxPro** використовується команда:

```
PUBLIC MemVarList | [ARRAY] ArrayName1
  (n Rows1 [, nColumns1])
  [, Array Name2 (n Rows2 [, nColumns2])]...
```

- ***Внутрішні змінні*** **(local)** діють тільки в межах процедури чи функції, у яких були створені. До них не можна звернутися з програми чи функції ні більш високого, ні більш низького рівня. Оголосити змінні внутрішніми в **Visual FoxPro** можна командою:

```
LOCAL MemVarList | [ARRAY] Array Name1
  (n Rows1 [n, Columns1])[, Array Name2
 (n Rows2 [n, Columns2])]...
```

- ***Регіональні змінні чи масиви*** **(regional)** можуть використовуватися тільки в **Visual FoxPro**. Вони подібні локальним і з'являються за допомогою команди:

```
REGIONAL MemVarList
```

Перед командою міститься директива **#REGION nRegionNumber** із вказівкою номера регіону **nRegionNumber** (від 0 до 31), у якому діють змінні, перераховані в списку **MemVarList**. У регіоні з іншим номером та ж змінна може мати інше значення.

Таблиця 5 - Типи основних файлів у **Visual FoxPro**

Тип файлу	Розширення файлу	Розширення файлу після компіляції
Користувальний додаток, що включає в себе окремі програмні файли	–	APP
База даних	DBC	–
Поля приміток у БД	DCT	–
Індексний файл	DCX	–
Таблиця	DBF	–
Індекс	IDX	–
Складений індекс	CDX	–
Поля приміток таблиці	FTP	–
Текстовий файл із повідомленнями про помилки компіляції	ERR	–
Виконувана програма, що створена на основі файла-додатка APP	–	EXE
Файли макрокоманд	FKY	–
Звіт	FRX	–
Поля приміток звіту	FRT	–
Програма	PRG	FXP
Ярлик	LBX	–
Поля приміток ярлика	LBT	–
Поля приміток меню	MNT	–
Меню	MNX	–
Згенерована програма меню	MPR	MPX
Файл елементів керування Active	–	OCX
Проект	PJX	–
Поля приміток проекту	PJT	–
Згенерований файл запиту	QPR	QPX
Форма	SCX	–
Поля приміток форми	SCT	–
Текстові файли	TXT	–
Візуальна бібліотека класів	VCX	–
Поля приміток візуальної бібліотеки класів	VCT	–
Файл конфігурації Visual FoxPro	FPW	–

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Розробити невелику програму, що ілюструвала б описані теоретичні положення.
3. Продемонструвати виконану роботу викладачу.
4. Підготувати звіт по лабораторній роботі. Зробити висновки.
5. Захистити лабораторну роботу.

Контрольні питання

1. Способи подання даних?
2. Які основні оператори для роботи з даними ?
3. Який пріоритет виконання операцій ?
4. Які є основні типи змінних ?
5. Які типи основних файлів у **Visual FoxPro** ?

Лабораторна робота №4

Тема: Написання програми. Розробка найпростіших програм у **Visual FoxPro**.

Мета: Навчитися складати дуже прості програми, що будуть працювати в СУБД **Visual FoxPro**.

Теоретичні відомості

Писати навіть найпростішу програму непросто. Найчастіше найважчим буває перший крок. Не будемо шукати труднощів і просто уникнемо цього першого кроку. Доручимо це самій СУБД, нехай сама напружує свій “електронний мозок”.

Зверніть увагу, що під час виконання будь-яких дій в інтерфейсі **Visual FoxPro** у вікні **Command** автоматично записується відповідна команда. Якщо *кілька* послідовних *команд перенести* в *програмний файл*, то вийде ***програма.*** Якщо її запустити, кілька дій, які ми до цього робили за допомогою меню і відповідних діалогових вікон, будуть виконані відразу без нашого безпосереднього втручання.

Якщо ми захочемо із таблиці (заздалегідь створеної) **SPPOS.DBF**, що містить список постачальників, прочитати дані, що відносяться до фірми з назвою «Комп'ютерне утворення», ми повинні з меню ***File*** вибрати команду ***New,*** встановити тип файлу ***Program***. Відкриється вікно редактора **FoxPro**. У ньому можна набрати зазначені рядки чи скопіювати їх з вікна ***Command***. Під час закриття вікна редагування необхідно вказати ***ім'я файла***, що надалі і буде ім'ям програми. Для зміни чи доповнення програми необхідно її відкрити за допомогою команди ***Open*** у меню ***File***.

- Якщо ви взялися за розробку програм, то вам часто доведеться виконувати цю операцію і працювати з ***редактором*** **Visual FoxPro**. Тому приділимо цьому небагато уваги. Почнемо з конфігурації, адже від того, наскільки зручно встановлені ті чи інші параметри роботи редактора, залежить продуктивність вашої праці. Відкрийте будь-який файл із програмою чи створіть новий файл. Виберіть команду ***Options*** у меню ***Tools***. У діалоговому вікні, що з'явилося, знайдемо *вкладку* ***Edit***. Розглянемо доступні установки для редактора ***Edit Options for Program***:

Drag-and-Drop Editing – використання технології перетягування;

Word Wrap – перенесення частини рядка вниз, якщо її довжина перевищує ширину вікна редактора;

Automatic Indent – автоматичне збереження відступу під час переходу на новий рядок;

Alignment (напр. left) – вид вирівнювання;

Tab Width *(characters)* – число символів, що пропускаються під час натискання клавіши ***Tab***;

Show Line / Column Position – показує номер рядка і колонки;

Make Backup Copy – створення резервної копії;
Compile Before Saving – компіляція перед збереженням;
Save with End-of-File Marker – запис символу кінця файла;
Save with Line Feeds – запис символу кінця рядка;
Save Preferences – зберегти зроблені установки;
Set as Default – використовувати ці установки для всіх програмних файлів.

Set as Default	- зберегти.

Ok	- вибрати до виходу із СУБД.

Для роботи з редактором ми можемо використовувати команди, що розташовані в меню ***Edit*** і ***Format***. Ще зручніше використовувати клавіатурні комбінації, перелік яких приведений у таблиці 6.

Таблиця 6 - Комбінації клавіш, що використовуються в роботі з редактором

Комбінація клавіш	Пункт меню	Виконувана дія
Ctrl+A	Select All	Виділити весь текст у вікні редактора
Ctrl+C	Copy	Скопіювати виділений текст
Ctrl+F	Find	Знайти
Ctrl+L	Replace	Замінити
Ctrl+R	Redo	Повернути скасовану дію
Ctrl+V	Paste	Вставити фрагмент тексту з буфера в місце перебування курсора
Ctrl+W	–	Закрити редактор і зберегти всі зроблені зміни
Ctrl+X	Cut	Вирізати виділений текст
Ctrl+Z	Undo	Скасувати зроблену дію
Ctrl+Home	–	Перемістити курсор у початок тексту
Home	–	Перемістити курсор у початок рядка
Ctrl+End	–	Перемістити курсор у кінець тексту
End	–	Перемістити курсор у кінець рядка

Під час першого запуску програми й надалі під час її зміни, **Visual FoxPro** ***компілює*** вихідний код програми в так званий ***препроцесорний*** **код**, чи "**р-код**", що виконується в багато разів швидше зінтерпретованої програми. Файли з "р-кодом" одержують інше розширення. Зверніть увагу на дуже зручну можливість компіляції програми перед її збереженням. Включення цієї опції дозволяє відразу визначити синтаксичні помилки, що виявляються на цьому етапі, - недолік чи відсутність ком, неправильне описання ключових слів. У зв'язку з тим, що редактор **Visual FoxPro** використовується не тільки для роботи з файлами програм, але і взагалі

для редагування текстових даних, наприклад у полях приміток, то перелік доступних опцій у діалоговому вікні **Options** залежить від типу вікна, що був активним перед його викликом.

Під час набору програми дуже зручно в редакторі використовувати *макроси*.

Макросом називається попередньо записана послідовність команд, що запускається під час його запуску. Для запуску макросу найчастіше використовуються клавіатурні комбінації.

У **Visual FoxPro** макрос можна записати за допомогою відповідної команди в меню ***Tools***. У своїй роботі ви можете використовувати кілька наборів макросів, зберігаючи їх у відповідних файлах. Для збереження і завантаження необхідних макросів треба використовувати кнопки, розташовані внизу діалогового вікна (***Save, Restore, set Default, Clear All***). Діалогове вікно праворуч має кнопки: ***Record, New, Edit, Clear, Ok***.

Для запису макросу натисніть кнопку ***New***. З'явиться діалогове вікно для створення і редагування макросів. Натисніть клавіатурну комбінацію, за допомогою якої ви хочете надалі викликати даний макрос.

Як приклад ми з вами створимо макрос для автоматичного набору будь-якої складної команди. Нехай це буде команда задання циклу **FOR...NEXT**. Такі структурні команди найчастіше спричиняють помилки, тому що тіло циклу може займати десятки рядків і, розбираючись в його перипетіях, програміст забуває про необхідність закриття циклу. Присвоїмо цьому макросу клавіатурну комбінацію ***Ctrl+F***. Тоді за замовчуванням він одержить **ім'я ctrl f**. Не слід намагатися присвоїти макросу клавіатурну комбінацію, що уже використовується в **Visual FoxPro**. Тепер у поле **Macro Contents** наберемо такий текст:

FOR {ENTER} {ENTER} NEXT.

У фігурних дужках ми вказали назву клавіш, натискання яких нам треба імітувати в даний момент. У цьому прикладі ми імітуємо дворазове натискання клавіші *Enter* для переходу на наступний рядок і тим самим створюємо усередині команди порожній рядок для запису тіла циклу. Як правильно записати в макросі назви інших клавіш і їхніх сполучень можна довідатися, вивчивши довгу таблицю в довідці до команди ***ON KEY LABEL*** після виклику контекстної допомоги.

Тепер, працюючи в редакторі, ми завжди можемо натиснути клавіші ***Ctrl+F***, і в програмі в нас автоматично з'являться такі три рядки:

1. **FOR**

2.

3. **NEXT**

Програма в **Visual FoxPro** – це текстовий файл, що містить набір команд, написаних відповідно до синтаксичних правил мови. Програма може мати підпрограми (процедури), у яких містяться часто повторювані фрагменти коду, розташовані після основного тексту чи програми в окремому файлі. Підпрограма починається з ключового слова

PROCEDURE Procedure Name і виконується, поки не буде виконана одна з таких умов:

- Ще раз зустрінеться слово ***PROCEDURE***.
- Буде виявлена команда ***RETURN*** – повернення в попередню програму.
- Буде видана команда ***CANCEL*** – переривання роботи програми.
- Буде видана команда ***QUIT*** – вихід із СУБД.
- Зустрінеться нова команда ***DO*** для запуску іншої програми.
- Буде досягнутий кінець файла.

У **Visual FoxPro** аналогічно підпрограмі трактується поняття ***користувальної функції***, що починається з ключового слова ***FUNCTION, Function Name*** і на відміну від процедури може повернути необхідні значення у викликану програму.

Маємо чотири способи **виклику** функції:

1. Присвоїти значення змінній. Наприклад, наступний рядок коду запам'ятовує поточну системну дату в змінній **dToday**:
   dToday=DATE( )
2. Включити виклик функції в команду. Наприклад, наступна команда встановлює за замовчуванням каталог, ім'я якого повертає функція ***GETDIR( )***:
   SET DEFAULT TO GETDIR( ).
3. Надрукувати значення, що повертається, в активне вікно:
   ? TIME( )
4. Викликати функцію без запам'ятовування значення, що де-небудь повертається:
   = SYS(2002)

Для **переривання** виконання програми служить оператор

RETURN [Expression | TO MASTER| TO Program Name]

він передає керування програмі, що викликана, і в ній виконується наступна команда після команди виклику, якщо зазначена опція ***TO MACTER***, то керування повертається на найвищий рівень, а ця ж програма з опцією ***TO Program Name*** передає керування в зазначену програму. Під час використання оператора у функції команда автоматично повертає **.T.**, якщо не зазначений інший вираз на місці ***Expression***.

RETRY. Діє подібно команді ***RETURN***, але після повернення керування у програму, що викликає, буде повторюється виконана остання команда.

Створіть новий програмний файл і наберіть у редакторі приведений нижче текст. У цій програмі з'являється масив, елементи якого приймають значення від 1 до 10. Кожне присвоєння значення супроводжується появою на екрані повідомлення з вказівкою присвоєної величини. Подивіться на повідомлення, виведені цією програмою на екран.

Приклад найпростішої програми:

```
DIMENSION aSampleArray(10)
FOR nItem=1 TO 10
        ASampleArray(nItem)=nItem
        =Append_proc( )
NEXT
FUNCTION Append_proc
? “У масив додане нове значення”
?? nItem
```

Якщо ця програма здалася вам занадто примітивної, то без сумніву ви маєте рацію. Вам для закріплення даного теоретичного матеріалу потрібно розробити і налагодити свою програму.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Розробити і налагодити найпростішу програму, за вказівкою викладача.
3. Підготувати звіт по лабораторній роботі. Зробити висновки.
4. Захистити лабораторну роботу.

Контрольні питання

1. Які основні доступні установки для редактора ***Edit*** ?
2. Комбінації клавіш, використовуваних в роботі з редактором?
3. Що таке макрос?
4. Що таке підпрограма ?
5. Які є основні чотири способи виклику функції ?
6. Як діють команди ***RETRY*** та ***RETURN*** ?

Лабораторна робота №5

Тема: Огляд основних команд і функцій **Visual FoxPro**.

Ціль*:* Ознайомитися з правилами використання найосновніших команд і функцій **Visual FoxPro**.

Теоретичні відомості

Поки ми вчимося програмувати, нам дуже допоможе найпростіша команда виведення даних у **Visual FoxPro**:

? | ?? Expression

Ми даємо тут не повний її синтаксис, тому що ця команда навряд чи знадобиться вам для чогось віртуознішого, ніж виведення потрібного значення в процесі розробки і налагодження прикладної програми. Її дуже зручно набирати у вікні **Command.** Один знак питання "**?**" завжди виводить значення виразу ***Expression*** з нового рядка, два знаки питання "**??**" – на тому ж рядку.

Для запуску програми чи для передачі керування іншій програмі в **Visual FoxPro** дайте команду ***Do*** з меню ***Program*** і виберіть файл із потрібним ім'ям, чи у вікні **Command** наберіть команду

DO ProgramName1 | ProcedureName [WITH ParameterList][IN ProgramName2].

Якщо в **ProgramName1** розширення не вказується, то **Visual FoxPro** буде намагатися запустити цю версію програми в такій послідовності розширень для файлу з тим самим ім'ям:

- *EXE* – версія, що виконується;
- *APP* – прикладна програма;
- *FXP* – відкомпільована версія;
- *PRG* – програма.

Опція ***WITH*** використовується для передачі параметрів (заздалегідь визначених даних) у програму (число параметрів не повинно перевищувати 27). За замовчуванням параметри передаються за посиланням, для передачі за значенням необхідно встановити *SET UDF PARMS TO VALUE*. В ProgramName2 можна вказати файл, в якому розміщується програма, що викликається.

Раніше ми навчилися визначати в програмі потрібні нам зміні з необхідним типом даних. Під час обробки даних програмісту в силу різних причин часто доводиться перетворювати дані з одного типу в інший. У мові програмування **Visual FoxPro** для цього існує велика кількість функцій, з яких найбільш вживаними можна назвати такі:

STR (n Expression [, n Length [, n DecimalPlaces]]) – перетворить числовий вираз *n Expression* у рядок символів. Необов'язкові параметри *n Length* і *n DecimalPlaces* дозволяють задати довжину і число десяткових розрядів відповідно.

VAL (c Expression) – перетворить у число рядок символів, що складається з набору цифр.

Важко уявити собі програму обробки даних, в якій програмісту не довелося б з цими даними хоч щось робити. З числами все просто. У цій області ми постійно тренуємо себе під час відвідування магазинів. Суми ростуть, тренування стають усе більш напруженими. Складніше із символьними даними. Розглянемо основні функції для роботи з рядками.

У **Visual FoxPro** за частотою вживання можна виділити такі функції:

- **SUBSTR (c Expression, n StartPosition [, n CharactersReturned])**
  Повертає фрагмент рядка символів з рядкового виразу *c Expression*, що починається з позиції *n StartPosition* і довжиною *n CharactersReturned*.
- **LEFT (c Expression, n Expression)**
  Повертає з рядкового виразу *c Expression* фрагмент, що включає перші *n Expression* символів.
- **RIGHT (c Expression, n Expression)**
  Повертає з рядкового виразу *c Expression* фрагмент, що включає останні *n Expression* символів.
- **ALLTRIM (c Expression)**

Виключає всі початкові і кінцеві пропуски з рядкового виразу.

Під час роботи з даними нам постійно доводиться думати про їх "свіжість", тому ми ніяк не можемо залишити без уваги функції, пов'язані з визначенням поточної дати і часу.

У **Visual FoxPro** для цього можна використовувати такі функції:

- Для визначення поточної дати:
  DATE( )
- Для визначення поточного часу:
  TIME([n Expression])
- Для визначення поточної дати і часу у форматі дата і час:
  DATETIME( )
- Якщо у функції ***TIME( )*** як вираз задати будь-яку числову величину, значення поточного часу буде повернуто із сотими частками секунди, хоча тоді краще вже звернутися до функції ***SECOND( )***.

В тому випадку, якщо необхідно багаторазово виконувати будь-який блок команд, поки виконується задана умова, може бути використана команда, що у **Visual FoxPro** виглядає от так:

DO WHILE / Expression

```
    Command
    [LOOP]
    [EXIT]
ENDDO
```

Щоразу, коли програма досягає рядка з командою ***DO WHILE***, вона перевіряє значення виразу /***Expression***, і якщо воно дорівнює ***.T.***, то виконуються команди всередині структури, якщо ***.F.***, то керування передається рядку, що йде за ***ENDDO***. Опція ***LOOP*** дозволяє після її вказівки повернути керування рядку ***DO WHILE***, а опція ***EXIT*** – припинити виконання циклу, незважаючи на значення /***Expression***.

Найчастіше як умову використовують вираз:

- **DO WHILE . NOT . EOF( )** – до вичерпання записів у таблиці;
- **DO WHILE n Min < 10 . AND . n Max > 100** – поки змінна *n Min* менша 10 і змінна *n Max* більша 100;
- **DO WHILE .T.** – нескінченно виконуваний цикл (вихід тільки по ***EXIT***).

Природно, що у випадку використання ***LOOP*** і (чи) ***EXIT*** перед ними повинні бути записані умови їхнього виконання (найчастіше команда ***IF ... ENDIF)***. Приведемо простий приклад організації циклу для реалізації реакції на дії користувача.

```
DO WHILE . T .
    CLEAR
WAIT “Наберіть цифру 1 чи 2.” TO c_Number
DO CASE
   CASE c_Number = “1”
    ? “Ви набрали цифру 1”
   CASE c_Number = “2”
    ? “Ви набрали цифру 2”
   OTHER WISE
    ? “Помилка: Введене значення ні 1, ні 2!”
ENDCASE
WAIT “Хочете спробувати ще раз (Y/N)” TO c_Yes No
IF UPPER (c_Yes No) < > “Y”
    EXIT
ENDIF
ENDDO
```

У цьому прикладі для забезпечення введення користувачем даних використовується найпростіший варіант команди ***WAIT***, що призупиняє роботу програми і після натискання користувачем будь-якої клавіші присвоює значення цієї клавіші змінній, зазначеній в опції ***TO***.

Якщо ми запишемо в програму деяку послідовність команд, то вони будуть виконуватися рядок за рядком від початку до кінця програми.

Навряд чи в такий спосіб ми зможемо написати потужну програму, що повинна обробляти дії користувача в залежності від тих чи інших умов. Нам знадобляться засоби для керування послідовністю виконуваних команд. Найпростіше розгалуження програми в залежності від заданої умови можна забезпечити командою, синтаксис якої в **Visual FoxPro** записується в такому вигляді:

```
IF / Expression
        Commands
[ELSE
        Commands]
ENDIF
```

Як умову можна використати будь-який вираз, у тому числі і логічний. На місці команд, у свою чергу, може стояти знову команда **IF** (вкладена умова). Якщо вираз дорівнює ***.T.***, то виконуються команди, вкладені між ***IF*** і **ELSE**. Якщо ***.F.***, то виконуються команди, що стоять за ***ELSE***, а в тому випадку, якщо ця необов'язкова опція ***ELSE*** у команді не використовується, виконуються команди, що йдуть за **ENDIF.**

Більш складне розгалуження, коли є кілька умов, у залежності від який треба виконати ту чи іншу групу команд, організується структурною командою, що у **Visual FoxPro** виглядає так :

```
DO CASE
   CASE / Expression 1
        Commands
   CASE / Expression 2
        Commands
   …
   [OTHER WISE
        Commands]
ENDCASE
```

Команда виконує перевірку умов, заданих в операторах ***CASE***. Вони проглядаються послідовно зверху вниз і, як тільки перша з них виявляється правильною (логічний вираз дорівнює *.T.*), виконується блок команд для цього ***CASE***, після чого керування передається рядку програми, що йде за ***ENDCASE***. Якщо жодна з умов не істинна, виконується опція ***OTHER WISE***, за відсутності цієї опції команда ігнорується.

Наприклад:

```
DO CASE
   CASE n Choice = “1”
     ? “Обрано значення 1”
   CASE n Choice = “2”
     ? “Обрано значення 2”
   OTHER WISE
```

```
    ? “Обрано неправильне значення!”
  RETURN
ENDCASE
```

Цикл, крім структури ***DO WHILE***, можна організувати і стандартним для більшості мов програмування способом.

Тому наступна команда куди популярніша попередньої:

```
FOR Mem VarName = n InitialValue TO n FinalValue [STEP n Increment]
    Commands
    [LOOP]
    [EXIT]
NEXT
```

Команда **FOR … NEXT** забезпечує виконання блоку команд для кожного значення змінної, починаючи зі значення, рівного *n InitialValue*, до значення *n FinalValue*, покрокове збільшення чи зменшення відбувається на величину, задану *n Increment* яка за замовчуванням дорівнює 1. Опція ***LOOP*** приводить до передачі керування на початок наступного циклу, а ***EXIT*** – до припинення дії команди.

У синтаксисі **Visual FoxPro** приклад організації вкладеного циклу приведений нижче. У змінну *MyString* 10 разів записуються значення від 0 до 9.

```
MyString = “   ”
FOR Words = 10 TO 1 Step-1
   FOR Chars = 0 TO 9
MyString = MyString + STR (Chars)
   NEXT
MyString = MyString + “   ”
   NEXT
? MyString
```

Навіть якщо ми навчилися складати поки дуже простенькі програми, ми повинні прагнути до високих ідеалів і намагатися створити дружній інтерфейс. Тут не обійтися без засобів, що дозволяють у гарній манері повідомляти користувачам різну інформацію, може, і не дуже приємну, запитувати в них поради про подальші дії. З огляду на важливість цієї проблеми для сучасного користувацького додатка, на перше місце поставимо функцію. У **Visual FoxPro** вона має такий синтаксис:

MESSAGEBOX (c MessageText [, n DialogBoxType [, c TitleBarText]])

Функція виводить на екран обумовлене користувачем діалогове вікно. Параметр *c MessageText* визначає текст, що з'являється в цьому вікні. Щоб перемістити частину повідомлення на наступний рядок у діалоговому вікні, використовуйте в *c MessageText* символ повернення каретки (*CHR*

(13)). Висота і ширина діалогового вікна збільшується в міру необхідності, щоб помістити заданий в *c MessageText* текст.

Параметр *n DialogBoxType* визначає кнопки і піктограми, що з'являються в діалоговому вікні, а також початково обрану, після виведення діалогового вікна на екран, кнопку.

У таблиці 7 приведені можливі значення для цих елементів.

Таблиця 7

Значення n DialogBoxType	***Елементи***
	Кнопки діалогового вікна
0	Тільки кнопка ОК
1	Кнопки ОК і Cancel
2	Кнопки Abort, Retry і Ignore
3	Кнопки Yes, No і Cancel
4	Кнопки Yes, No
5	Кнопки Retry і Cancel
	Піктограма
16	Знак “Стоп”
32	Знак “Питання”
48	Знак “Вигук”
64	Піктограма “Інформація”
	Кнопка за замовчуванням
0	Перша кнопка
256	Друга кнопка
512	Третя кнопка

Пропуск значення *n DialogBoxType* ідентичний присвоєнню йому значення 0.

Параметр *n DialogBoxType* повинен складатися з суми трьох значень – по одному з кожного розділу таблиці 7.

Параметр *c TitleBarText* визначає текст, що з'являється в заголовку діалогового вікна. Якщо ви опускаєте *c TitleBarText*, у назві вікна з'явиться заголовок “Microsoft **Visual FoxPro**”.

Значення функції, що повертається, ***MESSAGEBOX ( )*** вказує, яка кнопка була обрана в діалоговому вікні. У діалогових вікнах із кнопкою Cancel при натисканні клавіші Esc буде повертатися те ж саме значення *(2)*, як і при виборі Cancel.

Таблиця 8 містить список повернутих функцією ***MESSAGEBOX( )*** значень:

Таблиця 8

Значення, що повертається	***Кнопка***
1	OK
2	Cancel
3	Abort
4	Retry
5	Ignore
6	Yes
7	No

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Розробити і налагодити програму з використанням команд, зазначених викладачем.
3. Підготувати звіт по лабораторній роботі, зробити висновки.
4. Захистити лабораторну роботу.

Контрольні питання

1. Які основні команди ?
2. Які є функції для перетворення даних ?
3. Які функції використовуються для роботи з рядками ?
4. Які основні функції для визначення поточної дати і часу ?
5. Використання циклу.

Лабораторна робота №6

Тема: Об’єктно-орієнтоване програмування. Об’єктна модель і її властивості.

Мета: Одержати основні відомості про принципи ООП, вивчити можливості об’єктної моделі **Visual FoxPro**.

Теоретичні відомості

Програмування не таке вже й приємне зайняття, як можна видатися на перший погляд. Регулярно програмісти зіштовхуються з двома подіями, що сильно діють на нерви: вони то годинами мучаться в роздум, як запрограмувати те чи інше функціональне рішення, то їх долає жахлива нудьга при багаторазовій реалізації давно відпрацьованих рішень. Спосіб боротьби з цими неприємностями має назву „Об’єктно-орієнтоване програмування”(ООП).

З різних причин ООП, як невід’ємна частина сучасних систем RAD, не відразу стала надбанням широкого кола програмістів, довгий час залишаючись долею професіоналів , що пишуть на мовах, подібних C++. Серед розглянутих у даній книзі засобів розробки найбільш розвинуті засоби ООП має СУБД **Visual FoxPro.**

Об’єктна модель і її властивості

В структурному програмуванні послідовно виконуються оператори, які записуються програмістом відповідно до логіки розв’язання поставленої перед ним задачі. А тепер уявіть інтерфейс сучасної прикладної програми, що-небудь типу Windows-98. Уявили це розмаїття вікон, діалогів, задач? І все це то відкривається, то закривається, то переміщується, то змінює колір. Ризикнемо припустити, що такий інтерфейс написати засобами стандартної структурної мови програмування просто неможливо.

Об’єктно-орієнтоване програмування робить акцент не на програмні структури, а на об’єкти.

Майже усе, що має для вас інтерес і, можливо, навіть не існує візуально, може бути об’єктом. Об’єктом може бути вікно, поле для введення даних, користувач вашої програми, сама програма і т.д. Тоді будь-які дії ми можемо прив’язати до цього об’єкта, а також описати, що повинно статися з ним під час певних дій користувача (наприклад, при закритті вікна). Корисний, багаторазово використовуваний об’єкт можна зберегти в якості типового і застосувати його в декількох програмах для забезпечення подібної функціональності.

Таким чином, в ООП програміст створює потрібні об’єкти, а потім описує дії з ними і їхню реакцію на дії користувача. При такому підході будь-яку програму можна написати, включаючи туди той чи інший набір об’єктів, що забезпечують виконання тих чи інших функцій.

Більш детальне знайомство з принципами ООП почнемо з термінології, тому що деякі поняття тут є не тільки новими, але і не такими простими, як може здаватися на перший погляд.

Абстрактним типом даних називається такий тип даних, що описаний програмістом, але не підтримується у використовуваній мові програмування.

У мовах програмування, що не підтримують ООП, таких як, наприклад, стандартний С, абстрактний тип даних може бути створений із існуючих типів даних. Гарним прикладом такого підходу використання ***структур***; у мовах, що не підтримуються ООП на понятті „абстрактний тип даних", оснований опис **класів,** здійснюваний у програмі на високому рівні. У процесі роботи програми цей опис використовується для створення об'єктів, що виконують реальні функції. У **Visual FoxPro** такий тип даних підтримується програмно за рахунок використання команди DEFINE CLASS і візуально під час використання Конструктора класів (Class Designer). Об'єкти, створені під час роботи програми, доступні для керування за допомогою спеціальних змінних, які служать як покажчики на ці об'єкти.

Абстракцією називається метод, використанням якого можна ігнорувати неістотні в даний момент деталі і функції програми для того, щоб одержати можливість зосередитися на інтерфейсі програми, обміні повідомленнями з користувачем і т.д.

Абстракція підтримує ідею „чорної скриньки", коли частина чи навіть вся інформація про поводження об'єкта може бути схована. Такий підхід забезпечує можливість використання в додатку користувача додатку об'єктів із заздалегідь визначеними властивостями, функціональність яких не може опускатися нижче визначеного рівня. У той час можливе надання таким об'єктом додаткових властивостей і функцій, зміна зовнішнього вигляду і т.д.

Класом називають шаблон, що описує методи і властивості, використовуванні для визначеного типу об'єктів.

На підставі опису класу під час роботи програми створюються об'єкти, що і забезпечують виконання заданої функції. З опису одного класу може бути створено скільки завгодно об'єктів. Такі об'єкти будуть називатися ***екземплярами одного класу***. Описати клас програмно можна за допомогою команди DEFINE CLASS і візуально використовуючи Конструктор класу.

У **Visual FoxPro** ***класи поділяються на візуальні і невізуальні.*** **Візуальні** класи служать ***прообразами*** об'єктів, що будуть видимі і, відповідно, стануть основою інтерфейсу користувача майбутньої програми. **Невізуальні** класи можна буде побачити тільки в момент проектування на їхній основі об'єктів, що виявляються невидимими при роботі програми.

Як правило, об'єкти, основані на невізуальних класах, створюються і керуються програмою, за допомогою відповідних команд і функцій. Класи зберігаються у файлі бібліотеки класів, що полегшує до них доступ. Цей файл має розширення VCX і може містити один чи кілька класів.

Об'єктом називається програмно зв'язана колекція ***методів*** (функцій) і властивостей, що виконують одну функціонально зв'язану задачу.

Наприклад, об'єкт – це вікно для визначення даних – має набір властивостей, що описують його зовнішній вигляд, і набір методів для керування його поводженням на екрані. Програмно об'єкт може бути створений за допомогою функції ***CREATEOBJECT*** ( ).

Властивість – це характеристика, за допомогою якої описується зовнішній вигляд і робота об'єкта. Наприклад, заголовок, тип шрифту для написання, вид ліній і колір у рамці, що обрамляє, джерело даних і т.д.

Подія – це дія, що пов'язана з об'єктом. Подія може бути ініційована користувачем, викликана програмою чи операційною системою.

Подія є основним інструментом для описання необхідної реакції об'єкта на дії користувача. Усі можливі події містять за замовчуванням закладену в них реакцію, яку програміст може змінити, записавши в процедуру події визначений програмний код. Ці процедури можуть бути захищені від подальшої зміни чи навпаки – легко доступні для модифікації користувачем. Деякі дії в програмі викликають цілу низку подій, що відбуваються у визначеній послідовності. Наприклад, завантаження екранної форми викликає події, послідовність яких не може бути змінена.

Метод – це функція чи процедура, що буде керуватися роботою **об'єкта**.

Момент виконання методу визначається винятково програмістом, тому що метод визначається тільки за його явного вказування, наприклад OBJECT.DRAW.

Метод може бути захищений від подальшої зміни. Як у будь-яку функцію, в нього можна передати параметри й одержати значення, що повертається. Однак ні метод, ні подія не можуть мати вкладених процедур. Варто також врахувати, що метод не може мати ім'я, що збігається за назвою з властивістю.

Ієрархія класів – це деревоподібна структура класів, що відображає взаємозв'язок між використовуваними в додатку класами одного типу.

Розібратися в понятті „ієрархія класів" нам допоможе рис.1. Класи, що знаходяться на більш низьких рівнях ієрархії, називаються ***підкласами***. Будь-який з класів, що знаходяться на більш низьких рівнях в ієрархії, у порівнянні з базовим класом, буде ***підкласом.*** Класи, що знаходяться на більш високому рівні, на підставі яких описані класи, що стоять нижче в ієрархії, називаються ***батьківськими*** чи ***суперкласами.*** Наприклад, на рис.1 базовий клас розроблювача буде батьківським відносно класу кнопок для панелі інструментів. На підставі одного батьківського класу може бути

створено скільки завгодно підкласів наступного рівня. Такі підкласи називаються ***екземплярами даного класу***. На рис.1 такими екземплярами будуть класи для виконання визначених дій, створені на основі батьківського класу графічних кнопок.

Ієрархія класів може включати стільки рівнів, скільки може побажати розробник, однак, розробляючи системи класів для додатка, краще не перевищувати п’яти рівнів ієрархії.

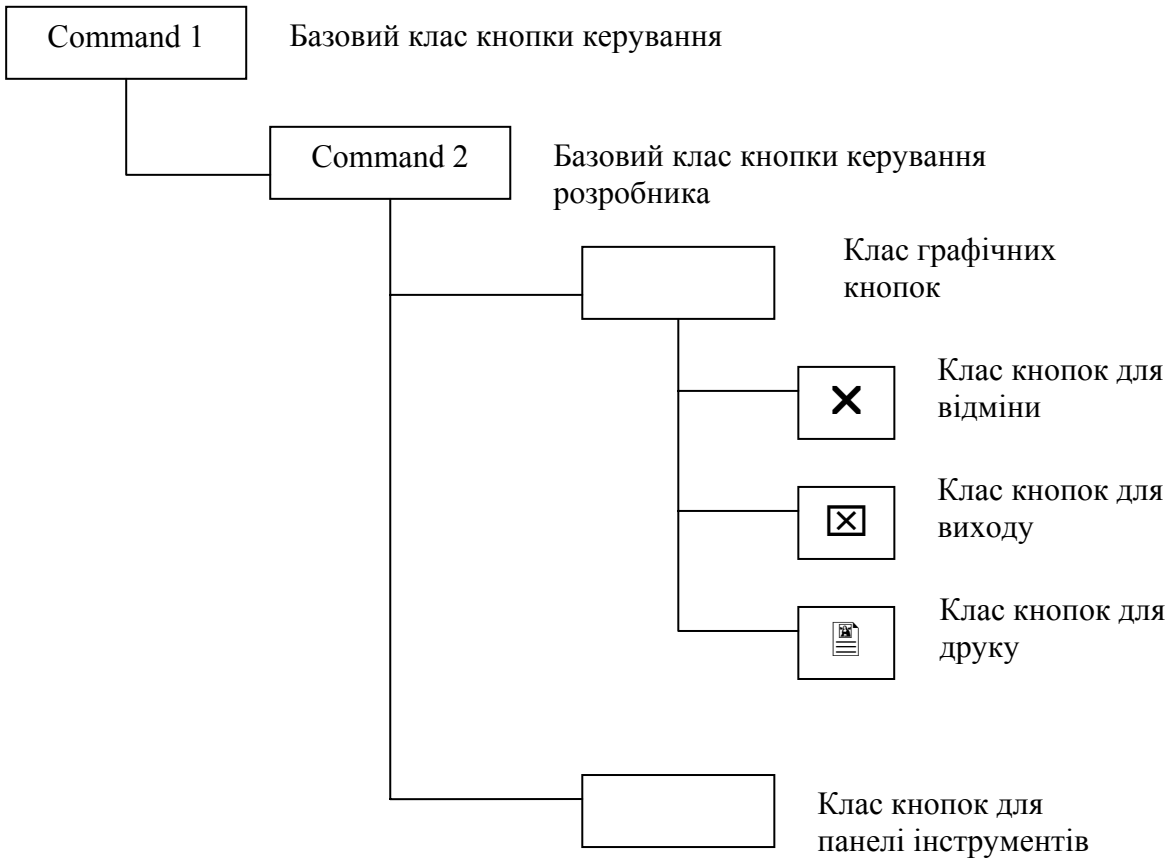

Рисунок 1 – Приклад ієрархії класів

Базовим класом називається клас, що знаходиться на вершині ієрархії класів, використовуваних у додатку.

У **Visual FoxPro** базові класи мають дуже серйозну особливість. Вони вбудовані в саму СУБД і, отже, їхнє описання не може бути змінене. У зв’язку з цим корисно ввести поняття „базових класів розробника”. Базові класи розробника є дублерами базових класів **Visual FoxPro,** стоять на наступному рівні ієрархії після них, але є класами більш високого рівня для всіх інших класів, використовуваних у додатку, як це видно на рис. 1 на прикладі класів для кнопок керування.

Усього в **Visual FoxPro** програміст може використовувати близько 30 базових класів. Їхня класифікація приведена на рис.2, а призначення описане в табл.9.

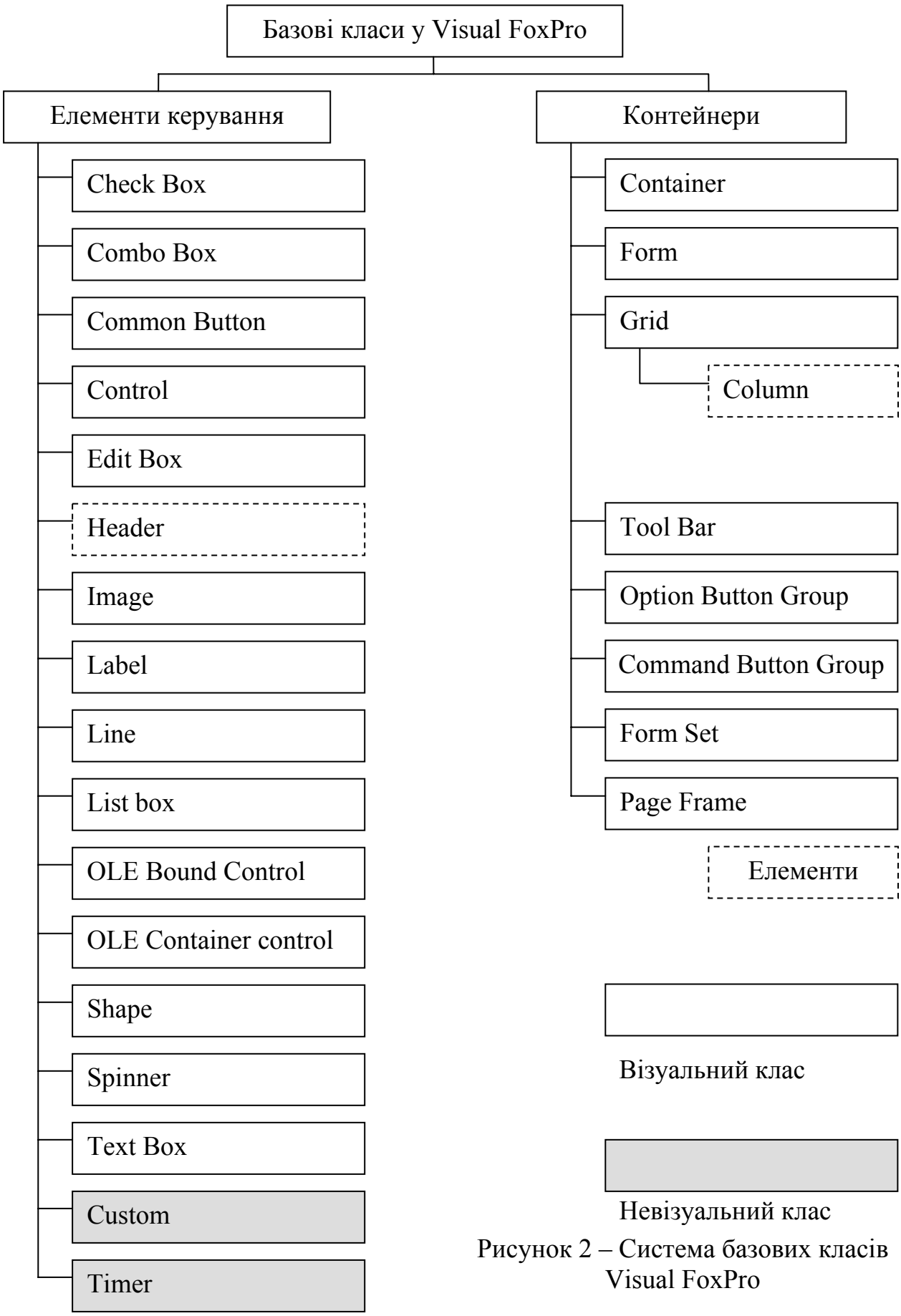

Рисунок 2 – Система базових класів Visual FoxPro

Таблиця 9 – Базові класи в Visual Fox Pro

Ім’я класу	Опис
Check Box	Створює поле перевірки, що використовується для переключення між двома станами
Column	Створює стовпець в об’єкті Grid. Стовпець може містити дані з поля в таблиці чи вираз, а також містити в собі будь-які інші елементи керування
Combo Box	Створює список, з якого можна вибрати один елемент. Сполучити можливості елементів керування List Box, Text Box, тому що крім вибору зі списку дозволяє вводити дані
Command Button	Створює одиночну кнопку керування. Кнопка звичайно використовується, що активізувати подію, подібно до закриття форми, переміщення курсору в інший запис, друкування звіту і т.д.
Command Group	Створює групу кнопок керування
Container	Створює об’єкт, що може містити інші об’єкти. Об’єкти Container можуть містити об’єкти і дозволяють доступ до об’єктів, що містяться в середині них
Control	Створює об’єкт елемента керування, що може містити інші захищені об’єкти. Об’єкти Control можуть містити в собі інші об’єкти, але на відміну від об’єктів Container не дозволяють здійснювати доступ до об’єктів, що знаходяться всередині них
Custom	Створює обумовлений користувачем об’єкт на основі класу користувача. Обумовлені користувачем класи – це класи з властивостями, подіями і методами, але без візуального подання
Edit Box	Створює область редагування. Використовуйте елемент керування Edit Box для символьних полів чи полів приміток великої довжини
Form	Створює форму для роботи з даними і керування роботою програми. Форма – це об’єкт-контейнер, що містить у собі всі необхідні елементи керування і складає основу інтерфейсу користувача
Form Set	Створює об’єкт-контейнер, що містить набір форм
Grid	Створює об’єкт Grid. Grid – це об’єкт-контейнер, що відображає дані в рядках і стовпцях і за зовнішнім виглядом схожий на вікно Browse, але має розширену функціональність, тому що має повний контроль над кожним елементом Grid за

	рахунок окремого набору властивостей
Header	Створює заголовок для стовпця в Grid. Об'єкт Header дозволяє відповідати на події, тобто може змінювати своє значення в процесі роботи програми
Image	Створює елемент керування, що показує зображення у форматі BMP
Label	Створює мітку, що відображає текст
Line	Створює елемент керування, що відображає список пунктів, з яких ви можете вибрати один чи декілька. Може застосовуватися для введення даних коли користувач повинен ввести тільки заздалегідь визначені значення
List Box	Створює зв'язаний елемент керування OLE. Зв'язаний елемент керування OLE дозволяє вам додавати об'єкти, що включаються OLE з інших прикладних програм типу Microsoft Word і Microsoft Excel, що підтримують стандарт OLE 2.0
OLE Container Control	Створює елемент керування OLE. OLE-об'єкти містять елементи керування Active X (файли з розширенням OCX) OLE-об'єкти, що включаються з інших прикладних програм Microsoft Word і Microsoft Excel. На відміну від елементів керування Active X OLE-об'єкти, що включаються, не мають власного набору подій. Крім того, елементи керування Active X не прив'язані до поля типу General у таблиці FoxPro, як зв'язані елементи керування OLE
Option Button	Створює одиночну кнопку вибору. Така кнопка може бути додана тільки до групи кнопок вибору
Option Group	Створює групу кнопок вибору. Група кнопок вибору – це контейнер. який містить окремі кнопки вибору. Дозволяє користувачу вибирати одну дію зі списку можливих варіантів, запропонованих набором кнопок
Page	Створю сторінку в сторінковому блоці. Об'єкт Page дозволяє з легкістю створювати багатосторінкові форми чи діалоги, шляхом переміщення у вікно форми набору сторінок, що переключаються за допомогою вкладок
Page Frame	Створює сторінковий блок, у якому містяться сторінки форми. Сторінковий блок визначає глобальні характеристики сторінки форми: розмір і

	положення на екрані, вид рамки, активну сторінку і т.д.
Shape	Створює елемент керування форми, що відображає геометричну фігуру (прямокутник, коло чи еліпс)
Spinner	Створює лічильник для фіксованої зміни числового значення
Text box	Створює текстове поле, у якому можна редагувати вміст змінної, елемента чи масиву поля. Текстове поле – це один із най широко використовуваних елементів керування для введення і редагування заздалегідь не визначених величин
Timer	Створює невидимий під час роботи програми об'єкт, що забезпечує контроль за часом виконання усієї програми чи окремих її фрагментів. Керування за допомогою таймера є корисним для фонової обробки. Типове використання для таймера – це одержання системного часу для визначення моменту, коли необхідно виконати будь-яку програму чи перервати деяку дію
Tool Bar	Створює панель користувача інструментів. Панель інструментів являє собою набір об'єктів, що об'єднані в одному вікні. Панель інструментів може бути прикріплена до верхньої, нижньої чи бічної рамок головного вікна. Якщо панель не прикріплена, вона поводиться аналогічно формі.

Базові класи поділяються на ***контейнери*** й ***елементи керування*** . Варто пояснити трохи докладніше відмінність класів, на основі яких створюються майбутні об'єкти – елементи керування, від класів-контейнерів. Об'єкти-контейнери можуть містити всередині себе інші об'єкти, у той же час допускаючи маніпуляції з цими внутрішніми об'єктами. Їх можна назвати складеними об'єктами, при цьому окремі складові частини не втрачають свого „суверенітету". Об'єкти – елементи керування, основані на неконтейнерних класах, хоча також можуть складатися з декількох складових частин, допускають маніпуляції з ними тільки, як з єдиним компонентом.

Деякі базові класи займають особливе положення в зв'язку з тим, що самі є складовою частиною іншого базового класу і тому не можуть бути основою для створення підкласу (на рис.2 виділені пунктирною лінією).

Спадкуванням називається здатність передачі властивостей і методів, що належать класу, на основі якого створюється підклас, новостворюваного класу.

Новостворюваний підклас автоматично успадковує усі властивості і методи батьківського класу, але завжди можна змінити якісь з цих властивостей чи методів для виконання спеціалізації даного підкласу.

Спадкування підтримується не тільки при створенні підкласу, але і надалі. Таким чином, усі зроблені вами зміни в батьківському класі відразу відіб'ються на його підкласах.

Як видно з рис.1, клас кнопок для виведення даних на друк успадковує свій зовнішні вигляд від класу графічних кнопок, але має спеціалізоване зображення і, мабуть, специфічну реакцію на натискання. Якщо ми вирішимо змінити розмір кнопки, в класі графічних кнопок зміняться розміри усіх використовуваних у додатку кнопок, що стоять нижче в ієрархії класів.

Оператор вказівки діапазону дозволяє викликати метод батьківського класу з більш низького рівня, у межах описаного підкласу. Це дозволяє розширити функціональність об'єкта без необхідності написання додаткового програного коду. Створюючи підкласи він автоматично успадковує всі методи класу. Ми можемо змінити успадкований метод, і в той же час, виконати не тільки цей змінений метод для даного підкласу, але і метод батьківського підкласу.

Оператор має такий синтаксис:

c ClassName :: Method

Інкапсуляція – це можливість об'єднання зв'язаних фрагментів чи даних процесів в окремий модуль – ***<u>контейнер.</u>***

Це схоже на поняття абстракції – приховання внутрішніх даних, тобто використання принципу створення об'єкта, як „чорної скриньки". Такий об'єкт буде працювати без розкриття своєї внутрішньої структури, яка забезпечує його функціональність. Елементи керування, основані не на контейнерних класах, являють гарний приклад використання принципу інкапсуляції. Використання інкапсуляції дає дві переваги програмісту:

- Простіший процес розробки програми. Під час створення об'єктів програміст може зосередитися на більш вузьких, конкретних задачах без необхідності обмірковування тих наслідків, що можуть відбутися в інших частинах програми через зроблені ним зміни.
- Безпечніший спосіб дублювання фрагментів чи коду об'єктів. Після того як об'єкт описаний і правильність його роботи в програмі перевірена, усі програми чи частини системи, що працюють за подібним алгоритмом, наприклад, доступ до даних, легко створюються як породжені об'єкти. Це включає ризик ушкодження даних під час незалежного програмування кожної частини системи і можливість виникнення при цьому великого числа помилок.

Поліморфізм – можливість здійснювати однакове звертання до різних об'єктів у випадку, якщо кожен об'єкт може виконувати таке звертання.

Поліморфізм забезпечує загальний інтерфейс для роботи зі створюваними об'єктами. Наприклад, виклик методу Draw для об'єктів Box1і Box2 може викликати зовсім різні наслідки, тому що кожен з цих об'єктів може мати власний метод Draw. Для програміста поліморфізм забезпечує більш просте і гнучке керування для групи зв'язаних об'єктів.

Звертання – це інструкція від одного об'єкта до іншого для виконання одного з методів того об'єкта, до якого звертаються.

Розрізняють три частини звертання: ***ім'я об'єкта***, що одержує повідомлення, ***ім'я методу*** і, можливо, ***передача параметрів***, якщо того вимагає виконуваний метод.

У звертанні ім'я об'єкта відокремлюється спеціальним оператором – крапкою (.). Наприклад, якщо треба зробити недоступним об'єкт-список з ім'ям ***MyList,*** необхідно виконати команду:

MyList. Enabled = .F.

Це викличе зміну властивості, що контролює доступність даного об'єкта для роботи з ним користувачу. Список змінить свій зовнішній вигляд і для повернення його в робочий стан потрібна команда:

MyList. Enabled = .T.

Звертання до об'єкта для виконання будь-яких дій аналогічне звертанню для зміни його властивостей. Після зміни об'єкта ми повинні вказати ім'я методу:

MyList. Refresh ( )

У деяких ситуаціях ми не зможемо прямо звернутися до об'єкта, що нас цікавить. Наприклад, якщо ми хочемо звернутися до згаданого вище списку з іншої форми, то повинні будемо послати наше звернення не списку, а формі, що містить потрібний список:

MyForm . MyList. Enabled = .T.

Такий порядок звертання до об'єкта може здатися стомливим, особливо, якщо уявити собі достатньо довгий ланцюжок вкладених об'єктів. Однак це дозволяє не підшукувати 100 унікальних імен для кнопки виходу в кожній з 100 використовуваних у додатку форм. Усі кнопки можуть мати те саме ім'я Cmd Close, і для кожного звертання знайдеться правильний адресат, тому що в них буде використовуватись ім'я потрібної форми:

- ***THIS –*** дозволяє послатися на поточний об'єкт ;
- ***THISFORM –*** дозволяє послатися на поточну форму;
- ***THISFORMSET –*** дозволяє послатися на поточний набір форм.

У командах ці оператори можуть указуватися замість відповідних імен об'єктів, форм чи наборів форм під час виконання будь-якого методу чи зміни значення властивостей. У яких випадках які оператори відносного посилання треба використовувати, допоможе розібратися табл.10.

Таблиця 10 – Звертання до об’єкта за допомогою відносної адресації

Джерело звертання	Приклад звертання до об’єкта MyObject
Інший метод того ж самого об’єкта	THIS . Enabled = .T.
Форма, що включає об’єкт	THIS . MyObject . Enabled = .T.
Інший об’єкт тієї ж самої форми	THISFORM . MyObject . Enabled = . T.
Ззовні форми	MyForm . MyObject . Enabled = . T.
Об’єкт іншої форми, що входить в один набір форм	THISFORMSET . MyForm . MyObject . Enabled = .T.
Ззовні набору форм	MyFormSet . MyForm . MyObject . Enabled = . T.

У **Visual FoxPro** є дві властивості, що дозволяють виконувати дії щодо ***активного об’єкта*** , тобто об’єкта, з яким у даний момент працює користувач (на який вказує курсор), без необхідності знання його імені. Властивість ***Active Form*** форми чи набору форм має такий синтаксис:

FormSet . ActiveForm . Property [=Setting]

чи FormSet . ActiveForm . Method

Ця властивість дозволяє з’ясувати задану в ***Property*** властивість для активної форми чи виконати зазначений у ***Method*** метод. Якщо ми задаємо параметр ***Setting***, то зазначене значення порівнюється з установленим, і у випадку рівності повертається .T. Якщо ***Setting*** не вказується, повертається встановлене значення властивості ***Property***.

Якщо ми не використовуємо набір форм, то замість ***FormSet*** у **Visual FoxPro** ми повинні послатися на екранний об’єкт за допомогою системної змінної ***_SCREEN***. Ця системна змінна виконує в такому випадку трохи нестандартні для системних змінних у **Visual FoxPro** функцій. Вона відіграє роль посилання на головне вікно **Visual FoxPro,** дозволяючи керувати ним як об’єктом. Наприклад, для очищення головного вікна **Visual FoxPro** від введених даних можна у вікні ***Command*** написати команду ***CLEAR.*** Той же самий результат буде досягнутий, якщо ми напишемо такий рядок:

_SCREEN . Cls

У зв’язку з тим, що об’єктна модель **Visual FoxPro** підтримує ***спадкування*** , під час роботи з класами не обійтись без відповідної інформаційної підтримки. Для одержання інформації про створені об’єкти можна скористатися властивостями, які мають статус „тільки для читання” і не можуть використовуватися для зміни властивостей об’єкта.

Object. BaseClass

повертає ім’я базового класу, на основі якого створений зазначений об’єкт.

Object. Class

повертає ім’я класу.

Object. ClassLibrary

повертає ім’я файла бібліотеці користувача, у якій містяться визначення класу, на основі якого створений об’єкт.

Object. ParentClass

повертає ім’я класу, що є батьківським для класу, на основі якого створений зазначений об’єкт.

Control. Parent

забезпечує посилання на елемент, що утримує керування об’єкт-контейнером.

Наприклад, ми можемо змінити колір фону форми, при будь-яких діях із включеним у неї елементом керування, без необхідності знати ім’я цієї форми:

THIS. Parent . BackColor =RGB (192,0,0)

Таким чином, головне призначення цієї властивості – використання її під час створення універсальних класів елементів керування, здатних впливати на об’єкт, у якому будуть розміщені елементи керування, створені на основі цього класу.

Для майже будь-якого об’єкта ми можемо записати коментар, що може виявитись дуже корисним на етапі налагодження чи супроводу програми. Необхідно скористатись властивістю:

Object. Comment [=c Expression]

Цю властивість зручно використовувати для додаткової ідентифікації об’єктів, якщо немає необхідності змінювати встановлені ідентифікатори, доступні для СУБД.

Записати будь-які дані для об’єкта можна за допомогою ще однієї властивості:

Object. Tag [= Expression]

Обидві перераховані вище властивості за замовчуванням, якщо їм не присвоєні будь-які значення, повертають порожній символьний рядок.

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Підготуватися до розробки програми, в основі якої лежить використання об’єктів.
3. Підготувати звіт по лабораторній роботі, зробити висновки.
4. Захистити лабораторну роботу.

Контрольні запитання

1. Що таке абстракція ?
2. Поняття класу. Ієрархія класу.
3. Поняття об’єкта.
4. Базовий клас. Класифікація.

5. Інкапсуляція. Поліморфізм.
6. Що таке звертання, його різновиди ?
7. Властивості щодо активного об’єкта.

Лабораторна робота №7

Тема: Команди і функції для роботи з класами і створеними на їхній основі об'єктами програми в **Visual FoxPro**.

Мета: Ознайомитися з можливостями роботи з класами.

Теоретичні відомості

Для роботи з класами і створеними на їхній основі об'єктами програми в **Visual FoxPro** існує декілька дуже важливих функцій і команд, які ми розглянемо.

Команда ***ADD CLASS***

Class Name [OF Class Library Name1] TO Class Library Name2 [OVER WRITE]

добавляє описання класу у візуальну бібліотеку класів. Параметр ***Class_Name*** визначає ім'я класу, що добавляється у візуальну бібліотеку класів ***Class Library Name2***. Якщо файл візуальної бібліотеки класів не існує, **Visual FoxPro** створює візуальну бібліотеку класів і добавляє в неї визначення класу.

Якщо ви опускаєте необов'язкову операцію ***OF Class Library Name1*****, Visual FoxPro** шукає описання класу в будь-яких візуальних бібліотеках класів, відкритих командою ***SET CLASSLIB*****. Visual FoxPro** згенерує помилку, якщо визначення класу не може бути розміщене або визначення класу з ім'ям, яке ви задаєте, вже існує в ***Class Library Name2***. Опція ***OF Class Library Name1*** визначає візуальну бібліотеку класів, з якої копіюється визначення класу.

Опція ***OVER WRITE*** видаляє всі класові визначення з візуальної бібліотеки класів перед тим як визначення нового класу буде додано.

Використання команди ***ADD CLASS*** добавляє визначення класу або копіює визначення класу з однієї візуальної бібліотеки класів в іншу. Визначення класу не може бути добавлене з програми процедурного файла або додатку ***Visual FoxPro*** (файли з розширенням *PRG* або *APP*).

Команда ***DEFINE CLASS***

```
DEFINE CLASS Class Name1 AS Parent Class
      [[PROTECTED Property1, Property2 …]
      Property = Expression …]
[ADD OBJECT [ PROTECTED] Object AS Class Name2 [NOINIT]
[WITH c Propertylist]]…
[[PROTECTED] FUNCTION | PROCEDURE Name
[NODEFAULT]
c Statements
[ENDFUNC | ENDPROC]]…
ENDDEFINE
```

Створює визначуваний користувачем клас або підклас і задає властивості, події і методи для класу або підкласу. Параметр ***Class Name1*** визначає ім'я класу.

Опція ***AS Parent Class*** визначає батьківський клас, на якому буде заснований створюваний клас або підклас. Батьківським класом може бути базовий клас **Visual FoxPro**, такий, наприклад, як ***Form*** або будь-який інший, визначуваний користувачем клас або підклас. Невізуальний невизначений користувачем клас може бути створений визначенням імені ***Custom Parent*** для ***Class***.

За допомогою опції ***[PROTECT Property1, Property2…]Property=Expression…*** можна призначити властивості створюваному класу і встановити для них значення, що будуть використовуватися за замовчуванням. Знак рівності говорить про те, що властивості ***Property Name*** присвоюється значення виразу ***Expression***. Щоб запобігти доступу і зміні значень властивостей поза визначенням класу або підкласу, включайте опцію ***PROTECTED*** і список захищених властивостей. Методи і події в середині визначення класу або підкласу можуть звертатися до захищених властивостей.

Опція ***ADD OBJECT*** дозволяє добавити об'єкт до визначення класу або підкласу з базового класу **Visual FoxPro**, визначуваного користувачем класу або підкласу, або з класу ***OLE***. Параметр ***Object*** визначає ім'я об'єкта і використовується для посилання на об'єкт зсередини визначення класу. Опція ***NOINIT*** указує на те, що метод Init не виконується під час добавлення об'єкта.

Опція ***WITH*** с ***Property List*** визначає список властивостей і значень властивостей об'єкта, який ви добавляєте до визначення класу або підкласу. Опції ***FUNCTION Name*** або ***PROCEDURE Name*** дозволяють створити описання дій, виконуваних під час виникнення події або використання методу для класу або підкласу. Події і методи створюються як набір функцій або процедур.

Включення опції ***NODEFAULT*** указує на те, що **Visual FoxPro** не буде реагувати на події, як це повинно було б трапитися, або виконувати процедури обробки методів. Це дозволяє використовувати свої власні оброблювачі визначених подій, що відрізняються від логіки, прийнятої в **Visual FoxPro**. Опція ***NODEFAULT*** може розташовуватися в будь-якому місці всередині процедури обробки подій або методу в конструкторі форми ***(Form Designer)***.

Параметр с ***Statement*** – це команди **Visual FoxPro**, що виконуються, коли викликається подія або метод.

Функції і процедури подій і методів можуть приймати значення шляхом включення оператора ***PARAMETERS*** як першого виконуваного класу рядка у функцію або процедуру.

Щоб створити об'єкт на основі визначення класу або підкласу, використовуйте функцію ***CREATE OBJECT( )*** з ім'ям відповідного класу або підкласу.

Визначення класу і підкласу, створені за допомогою команди ***DEFINE CLASS,*** не можуть розміщатися усередині команд структурного програмування типу

IF… ENDIF або

DO CASE ... ENDCASE і в циклах типу

DO WHILE ... ENDDO або

FOR… ENDFOR

Як приклад подивимося, як використовується ця команда для створення об'єктів. Створимо три різні форми

```
oForm1= CREATEOBJECT("frmTestForm")
oForm2= CREATEOBJECT("frmTestForm")
oForm3= CREATEOBJECT("frmTestForm")
*Змінимо їхні заголовки
oForm1. Caption = "Перша форма"
oForm2. Caption = "Друга форма"
oForm3. Caption = "Третя форма"
*Виведемо на екран за допомогою методу Show
*першу форму oForm1. Show
oForm1. Show
*Трошки розсунемо їх на екрані
oForm1. Move(oForm1.Left+50. oForm1.Top1+50)
oForm2. Show
oForm3. AutoCenter=.T.
oForm3.  Show
*Почекаємо реакції користувача
READ EVENTS
*Визначимо клас для створення наших форм
DEFINE CLASS frm Test Form AS Form
Back Color = RGB (192, 192,193)
Caption = “Test Form”
* Додамо форму кнопці керування
*для закриття форми
ADD OBJECT cmd Exit AS Command Button WITH;
Caption = “\<Вихід”,;
Left = 150,;
Top = 100,;
AutoSize = . T.
PROCEDURE cmd Exit. Click
RELEASE THIS FORM
*Коли з екрана буде прибрана остання форма,
```

*скасуємо стан чекання

```
IF_SCREEN. FormCount = 1
CLEAR EVENTS
ENDIF
ENDPROC
ENDDEFINE
```

Команда ***SET CLASSLIB TO***

```
SET CLASSLIB TO Class Library Name
[ADDITIVE]
[ALIAS Alias Name]
```

Відкриває візуальну бібліотеку класів із визначенням класів, що зберігаються в них. Параметр ***Class Library*** визначає ім'я файла бібліотеки. Опція ***ADDITIVE*** дозволяє задати псевдоніми для бібліотеки, на які можна посилатися створюючи об'єкт на базі класу, визначення якого зберігається в даній бібліотеці. Завдання цієї команди у вигляді ***SET CLASSLIB TO*** закриває усі відкриті бібліотеки класів.

```
RELEASE CLASSLIB Class Library Name
```

Дозволяє закрити вказану візуальну бібліотеку класів із відкритих раніше.

Крім команд для роботи з класами в програмі не обійтися без таких функцій.

```
CREAT OBJECT (ClassName [,Parameter 1, Parameter 2,...])
```

Створює об'єкт з опису класу або об'єкта ***OLE***. Аргумент ***ClassName*** визначає клас або ***OLE***-об'єкт, із якого буде створений новий об'єкт. **Visual FoxPro** шукає клас або ***OLE***-об'єкти у такій послідовності:

1. Базові класи **Visual FoxPro**.
2. Визначувані користувачами описи класів у тій послідовності, в якій вони були завантажені в пам'ять.
3. Класи в поточній програмі.
4. Класи в бібліотеках класів, відкриті за допомогою ***SET CLASSLIB*.**
5. Класи в процедурах, відкритих за допомогою ***SET PROCEDURE*.**
6. Класи в послідовності виконання програм **Visual FoxPro.**
7. Регістр ***Windows*** (для об'єктів ***OLE*).**

Для створення ***OLE***-об'єктів використовується такий синтаксис параметра ***ClassName:***

```
ApplicationName. Class
```

Наприклад, для роботи з таблицями ***Microsoft Excel*** за допомогою засобів ***OLE*** можна написати:

```
OExcelSheet = CREATE OBJECT ("Excel. Application").
```

Коли цей код буде виконаний, запускається Microsoft Excel у відкритому для користувача вигляді. Ви не зможете виявити його відображення на панелі задач Windows 95. Але в переліку завантажених

задач, що з′являється під час натискання клавіш Ctrl+Alt+Del, пакет Excel присутній.

Зауважте також, що навіть якщо пакет Excel завантажений на комп'ютері, то в прихованому вигляді завантажується ще одна його копія.

Необов'язкові параметри Parameter 1, Parameter 2, … використовуються, щоб передати значення процедурі події Init для класу. Подія Init виконується, коли використовується функція CREATE OBJECT() і вона дозволяє ініціалізацію об'єкта.

Використовуйте функцію CREATE OBJECT() для створення об'єкта з описання класу або об'єкта OLE і призначення посилання на об'єкт за допомогою змінної або елемента масиву.

Як приклад за допомогою цієї функції давайте програмно створимо декілька об'єктів на базі одного класу. У візуальній бібліотеці класів у нас зберігається описання класу панелі інструментів. Створимо на його основі три панелі і задамо їм різні властивості.

```
*Відкриваємо візуальну бібліотеку класів
    SET CLASS LIB TO Office
*Створюємо три об'єкти
    oTbr1=CREATE OBJECT("Office_Toolbar")
    oTbr2=CREATE OBJECT("Office_Toolbar")
    oTbr3=CREATE OBJECT("Office_Toolbar")
*Змінюємо для кожної створеної панелі
*заголовок їх вікна
    oTbr1. Caption= "Перша панель"
    oTbr2. Caption= "Друга панель"
    oTbr3. Caption= "Третя панель"
*Для першої панелі змінимо колір фону
    oTbr1. BackColor=RGB(0,0,255)
*виведемо їх на екран
    oTbr1. Show
    oTbr2. Show
    oTbr3. Show
*Дамо 20 секунд для того щоб їх можна
*було роздивитися і порухати
    READ TIMEOUT 20
```

Для визначення посилання на створювані об'єкти зручно використовувати масив, як це очевидно з наступного прикладу, у якому створюється п'ять форм.

```
*Визначаємо масив для створення посилання
    PABLIC ARRAY aTese(1)
    DIMENSION aTest(5)
```

```
*Очищуємо екран
      _SCREEN. CLS
*Створюємо п’ять форм і поміщуємо посилання
*на них у масив
      FOR nCount= 1 TO 5
      ATest (nCount)= CREATEOBJECT(“frm TestForm”)
*Розміщуємо їх по центру екрана
      aTest(nCount). AutoCenter= .T.
      END FOR
*Виводимо створені форми на екран
      FOR nCount= 1 TO 5
      IF TYPE(“aTest (nCount)”)=”0”
      aTest (nCount). Show
      ENDIF
      ENDFOR
*Визначаємо клас для створюваних форм
*із кнопкою для їхнього стирання
      DEFINE CLASS frm Test Form AS Form
      ADD OBJECT cmd EXIT As Command Button WITH;
      Caption = ”\< Вихід”,;
      Top = 111,;
      Left = 108,;
      Height = 29,;
      Wight = 94,;
      Visible = . T.
      PROCEDURE cmd Exit. Click
      RELEASE This Form
      ENDPROC
      ENDDEFINE
```

Функція

AINSTANCE (ArrayName, cClassName)

Розміщує всі зразки класу в масиві. Аргумент ArrayName визначає ім′я масиву, у якому розміщуються зразки. Якщо масив, який визначаєте, не існує, **Visual FoxPro** автоматично створює його. Якщо масив існує, але не достатньо великий, щоб умістити всі зразки, **Visual FoxPro** автоматично збільшить його розмір. Якщо масив виявляється більший ніж необхідно, **Visual FoxPro** зріже його до потрібних розмірів. У випадку, коли масив існує і функція ***AINSTANCE ()*** повертає 0 – ніякі зразки не знайдені, - масив залишається незмінним. Аргумент ***cClassName*** визначає ім′я базового класу Visual FoxPro, класу користувача або об’єкта **Visual FoxPro(*Cursor, Data Environment, Relation*).** Функція ***AINSTANCE ()*** повертає кількість зразків класу, розміщених у масиві.

Функція

AMEMBERS (ArrayName, Object [,1|2])

Розміщує в масиві для вказаного об'єкта імена властивостей, процедур і включених об'єктів. Аргумент ***ArrayName*** визначає масив, у який записуються імена елементів властивостей для ***ObjectName.*** Якщо масив не достатньо великий, щоб вмістити всі імена, **Visual FoxPro** автоматично збільшить його розмір. Якщо ви визначаєте наявний двовимірний масив, **Visual FoxPro** перетворить його в одновимірний. Аргумент ***Object*** визначає об'єкт, елементи властивостей якого розміщуються в масиві, вказаному в ***Array-Name*** . Аргумент ***Object*** може являти собою будь-який вираз, що має відношення до об'єкта, типу посилання на об'єкт, змінної об'єкта або елементу масиву об'єкта. Аргумент 1 указує на те, що в масив будуть включені властивості об'єкта, методи об'єкта. Масив є двовимірним із другим стовпцем, який уточнює, до якого типу відноситься елемент, внесений у список у першому стовпці. Можливі значення для другого стовпця:

Property, Event, Method або Object.

Аргумент 2 вказує на те, що масив буде містити імена об'єктів, що є елементами об'єкта, указаного в ***Object.*** Масив є одновимірним. Ця опція забезпечує метод для визначення імен усіх форм у наборі форм або елементів керування формами. Функція ***AMEMBERS ()*** повертає кількість об'єктів, властивостей і процедур для об'єкта, або 0, якщо масив не може бути створений.

Функція

COMPOBJ (oExpression 1, oExpression 2)

Порівнює властивість двох об'єктів і повертає значення "істина" (. T.), якщо їх властивості і значення властивостей ідентичні. Аргументи ***oExpression 1*** і ***oExpression 2*** визначають об'єкти для порівняння. Параметри ***oExpression 1*** і ***oExpression 2*** можуть бути будь-якими виразами, що визначають об'єкти такими, як посилання на об'єкт, змінні об'єкта або елементи масиву об'єкта.

Функція ***COMPOBJ ()*** повертає значення "хибність"(. F.), якщо об'єкт має такі властивості, яких немає в іншого об'єкта, або якщо об'єкти мають ідентичні властивості, але значення одного або декількох властивостей виявляються різними.

Об'єкти і їхні властивості

Основний зміст ООП полягає в створенні об'єктів і наявності зручних засобів маніпуляції ними.

У **Visual FoxPro** створення об'єктів підтримується стрункою і потужною системою класів. На основі базових класів програміст може

створити свої класи, і компонуючи їх у відповідні бібліотеки, забезпечитиїхню доступність для створення об’єктів і елементів керування в розроблюваному додатку, як це показано на схемі, приведеній на рис. 3. Звичайно, якщо бути зовсім точним, ця схема істотно спрощена в порівнянні з реальними можливостями. У розпорядженні програміста, що розробляє додаток на **Visual FoxPro,** ще є об’єкти для роботи з даними під час побудови форм і звітів, що не мають класів, і є можливість використовувати об’єкти інших додатків і об’єкти **ActiveX**.

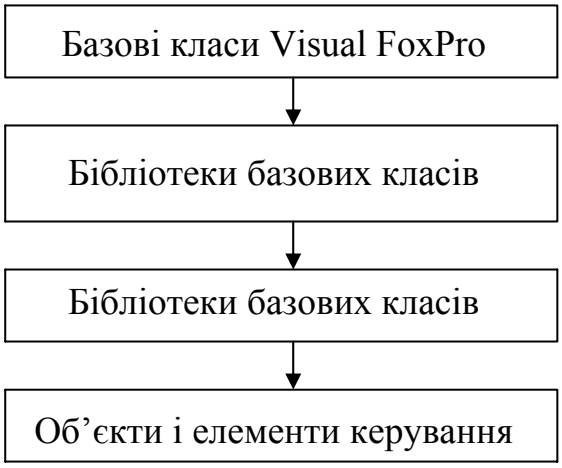

Рисунок 3

У таблиці 9(див. лабораторну роботу №6) був приведений список об’єктів, які програміст може створити в **Visual FoxPro** на основі базових класів. Візуально подані об’єкти легко створюються при проектуванні форм простим перетягуванням потрібного класу з панелі інструментів класу об’єкта.

Хід роботи

1. Ознайомитись з теоретичними відомостями.
2. Створити програму за виданим викладачем варіантом, використовуючи положення про класи.
3. Налагодити програму і отримати результат.
4. Оформити звіт по лабораторній роботі та захистити її.

Контрольні питання

1. Основні команди та їх застосування.
2. Основні функції та їх застосування.
3. Об’єкти та їхні властивості.

Лабораторна робота №8

Тема: Семантичні мережі.
Мета: Ознайомитись з поданням знань за допомогою семантичних мереж.

Теоретичні відомості

Для розробки експертних систем графічного подання знань використовуються семантичні мережі. Семантичні мережі, як і дерева розв'язків, складаються з вузлів, зображуваних колами і ліній зі стрілками, що з'єднують їх. Вузли мережі позначають деякі пропорції інформації, а лінії, що їх з'єднують – взаємозв'язки між ними.

На рис.4 показана семантична мережа, що графічно подає базу знань про птахів і літаки. Лінії вказують на відношення між вузлами, що містять інформацію. Наприклад, дивлячись на рисунок, можна сказати, що вузол "двигун" пов'язаний з вузлом "бензин" відношенням "використовує", тобто двигун використовує бензин.

Структура і об'єкти семантичних мереж

Один із способів застосування семантичних мереж – створення об'єктів. Подивимось, як це робиться на прикладі семантичної мережі (рис.4). Передусім потрібно розробити структуру, що описує всі об'єкти з загальною назвою ***ПТАХ***. Належність об'єктів структурі ***ПТАХ*** обумовлюється відношенням "це". Цим відношенням в структурі пов'язані об'єкти ***ОРЕЛ*** і ***СОКІЛ***, тобто так констатується факт, що орел і сокіл птахи. Всі інші лінії (відношення) на рисунку пов'язані з атрибутами структури ***ПТАХ***. Відношенням "має" в структурі пов'язані атрибути ***КРИЛА***, ***ОПЕРЕННЯ*** і ***ДЗЬОБ***. Іншими словами, об'єкти зі спільною назвою ***ПТАХ*** мають всі вказані атрибути, як це показано на рисунку. Відношенням "вміє" в структурі пов'язаний атрибут ***ЛІТАТИ***, тобто птахи вміють літати. Атрибут ***АЕРОДИНАМІЧНІ ПРИНЦИПИ*** пов'язаний в структурі відношенням "використовує", тобто птахи використовують аеродинамічні принципи. Скориставшись описаними атрибутами, створимо структуру ***ПТАХ***.

```
СТВОРИТИ-СТРУКТУРУ(ІМ'Я СТРУКТУРИ=ПТАХ,
                   ЧИСЛО АТРИБУТІВ=5,
                   АТРИБУТ=ДЗЬОБ, АТРИБУТ=ОПЕРЕННЯ,
                   АТРИБУТ=КРИЛА,
                   АТРИБУТ=ЛІТАТИ,
                   АТРИБУТ=АЕРОДИНАМІЧНІ ПРИНЦИПИ)
```

На основі цієї структури можна створити об’єкти ОРЕЛ і СОКІЛ. Створимо, наприклад, об’єкт ОРЕЛ:

```
СТВОРИТИ-ОБ′ЄКТ(ІМ′Я СТРУКТУРИ=ПТАХ,
                ІМ′Я ОБ′ЄКТА=ОРЕЛ,
                ДЗЬОБ=ДОВГИЙ,
                ОПЕРЕННЯ=БЛІДЕ, КРИЛА=ШИРОКІ,
                ЛІТАТИ=ВИСОКО,
                АЕРОДИНАМІЧНІ ПРИНЦИПИ=ПЛАНЕРУВАННЯ)
```

Тепер застосовуючи дані об’єкти, можна сформулювати базу знань, яка допоможе нам з відповідями на різноманітні питання, що відносяться до цих об’єктів.

Використання семантичної мережі в системі основане на правилах:

ЯКЩО..., ТО... .

З допомогою семантичної мережі можна створити базу знань, що зберігає відомості про птахів і літаки. В подальшому її можна застосовувати для ідентифікації літальних об’єктів.

Частиною бази знань, що містить відомості про птахів і літаки, можуть бути, наприклад, такі правила:

```
ЯКЩО АТРИБУТ-ОБ′ЄКТА=ДЗЬОБ,
ТО   ЛІТАЛЬНИЙ ОБ′ЄКТ=ПТАХ.

ЯКЩО АТРИБУТ-ОБ′ЄКТА=КРИЛА,
ТО   ЛІТАЛЬНИЙ ОБ′ЄКТ=ПТАХ АБО
     ЛІТАЛЬНИЙ ОБ′ЄКТ=ЛІТАК.

ЯКЩО АТРИБУТ-ОБ′ЄКТА=СОКІЛ АБО
     АТРИБУТ-ОБ′ЄКТА=ОРЕЛ,
ТО   ЛІТАЛЬНИЙ ОБ′ЄКТ=ПТАХ.

ЯКЩО АТРИБУТ-ОБ′ЄКТА=ОПЕРЕННЯ І
     АТРИБУТ-ОБ′ЄКТА=КРИЛА І
     АТРИБУТ-ОБ′ЄКТА=ДЗЬОБ,
ТО   ЛІТАЛЬНИЙ ОБ′ЄКТ=ПТАХ.

ЯКЩО ЛІТАЛЬНИЙ ОБ′ЄКТ=ПТАХ,
ТО   АТРИБУТ-ОБ′ЄКТА=ОПЕРЕННЯ
```

Підсумуємо основні властивості семантичних мереж.

1. Семантичні мережі описують відношення між об’єктами, які задаються вузлами мережі.
2. Вузли починаються колами і мають імена.
3. Відношення між вузлами вказуються лініями, що їх зв’язують.
4. Семантичну мережу можна використовувати для створення структур і об’єктів.

5. Семантичну мережу можна використовувати для створення правил бази знань.

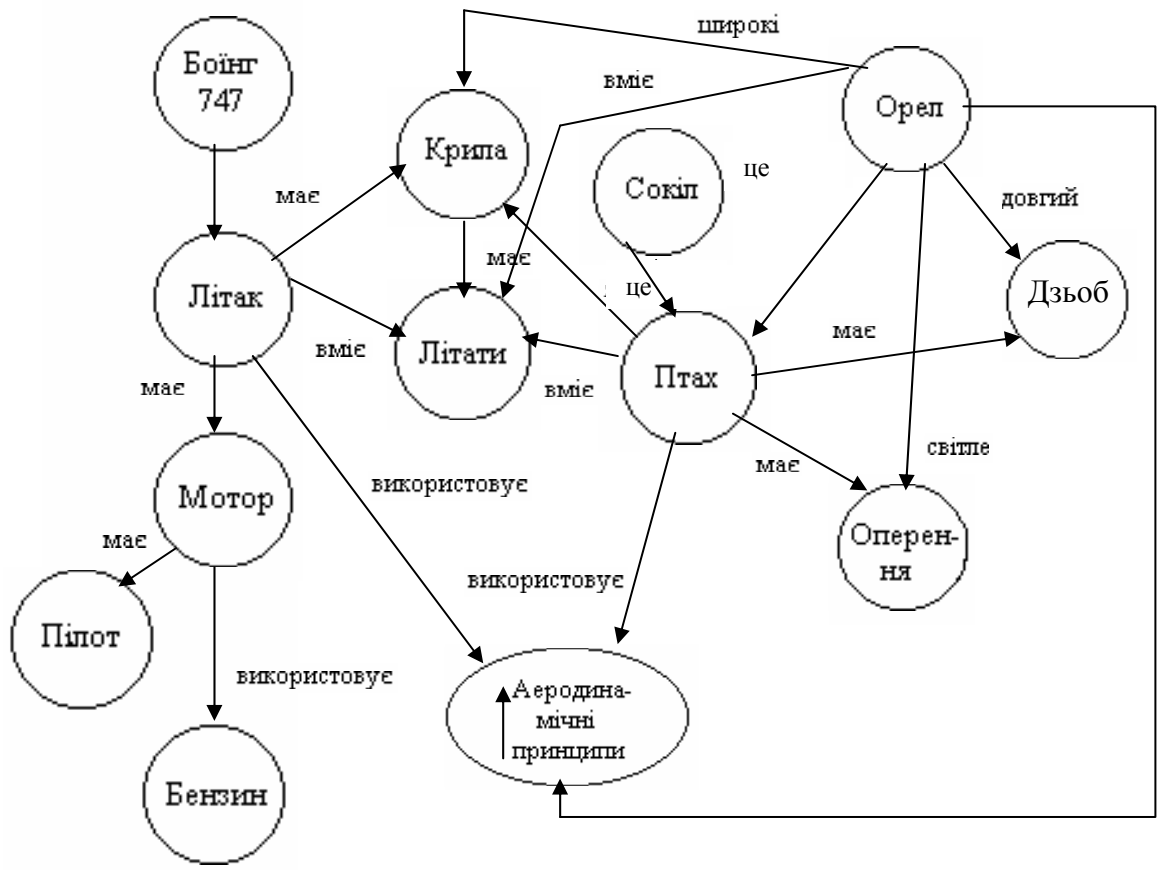

Рисунок 4 – Семантична мережа, що відображає взаємозв'язки між атрибутами птаха і літака

Приклад програми

```
Dimension R(5,5)
Dimension N(5)
For I=1 to 5
  For J=1 to 5
    R (I,J)=' '
    N(I)=' '
  EndFor
EndFor
N(1)='PTAH'
N(2)='SOKIL'
N(3)='OREL'
N(4)='KRYLA'
N(5)='OPERENNJA'
R(2,1)='E'
```

```
R(3,1)=’E’
R(1,4)=’MAE’
R(1,5)=’MAE’
NOM=0
DO WHILE NOM<4
  CLEAR
?   ‘ ВВЕДІТЬ ПОТРІБНИЙ НОМЕР; ‘
?   ‘ 1 – ОТРИМАТИ ВСІ ВУЗЛИ, ПОВ’ЯЗАНІ З ОКРЕМИМ ВУЗЛОМ
КОНКРЕТНИМ ВІДНОШЕННЯМ’
?   ‘ 2 – ОТРИМАТИ ІМЕНА ВСІХ ВІДНОШЕНЬ ДЛЯ ОКРЕМОГО ВУЗЛА’
?   ‘ 3 – ОТРИМАТИ ВСІ ПАРИ ВУЗЛІВ, ПОВ’ЯЗАНІ ОКРЕМИМ ВІДНОШЕННЯМ’
?   ‘ 4 – ВИХІД’
        INPUT   ‘ВВЕДІТЬ ПОТРІБНИЙ НОМЕР;’ TO NOM
        DO CASE
              CASE NOM=1
                    ACCEPT ‘ ВВЕДІТЬ ІМ’Я ВУЗЛА: ‘ TO NN
                    ACCEPT ‘ ВВЕДІТЬ ІМ’Я ВІДНОШЕННЯ: ‘ TO RM
                    FOR I=1 TO 5
                      IF NN=N(I)
                            TMP=I
                            ? N(1)
                      ENDIF
                    ENDFOR
                    FOR J=1 TO 5
                       IF R(TMP, J)=RM
                            ? N(J)
                       ENDIF
                    ENDFOR
                    WAIT
              CASE NOM=2
                    ACCEPT ‘ВВЕДІТЬ ІМ’Я ВУЗЛА: ‘ TO NN
                    FOR I=1 TO 5
                       IF NN=N(I)
                            TMP=I
                       ENDIF
                    ENDFOR
                    FOR J=1 TO 5
                       IF R(TMP, J)<>’’
                            ? N(J),’ ‘,R(TMP, J)
                       ENDIF
                    ENDFOR
                    WAIT
              CASE NOM=3
                    ACCEPT ‘ ВВЕДІТЬ ІМ’Я ВІДНОШЕННЯ: ‘ TO RM
                    FOR I=1 TO 5
                       FOR J=1 TO 5
                            IF R (I, J)=RM
                                ? N(I),’ ‘, N(J)
                            ENDIF
                        ENDFOR
                    ENDFOR
```

```
                    WAITE
                ENDCASE
ENDDO
        CLEAR
```

На лістингу програми показана написана на Visual FoxPro програма, що працює є семантичною мережею (рис. 5). Ця семантична мережа є лише частина показаної на рис. 4.

Приклади роботи програми:

```
ВВЕДІТЬ ПОТРІБНИЙ НОМЕР;
1 – ОТРИМАТИ ВСІ ВУЗЛИ, ПОВ'ЯЗАНІ З ОКРЕМИМ ВУЗЛОМ КОНКРЕТНИМИ
ВІДНОШЕННЯМИ
2 – ОТРИМАТИ ІМЕНА ВСІХ ВІДНОШЕНЬ ДЛЯ ОКРЕМОГО ВУЗЛА
3 – ОТРИМАТИ ВСІ ПАРИ ВУЗЛІВ, ПОВ'ЯЗАНІ ОКРЕМИМ ВІДНОШЕННЯМ
4 – ВИХІД
ВВЕДІТЬ ПОТРІБНИЙ НОМЕР: 1
ВВЕДІТЬ ІМ'Я ВУЗЛА: РТАН
ВВЕДІТЬ ІМ'Я ВІДНОШЕННЯ: МАЕ
KRULA
OPERENIA

ВВЕДІТЬ ПОТРІБНИЙ НОМЕР;
1 – ОТРИМАТИ ВСІ ВУЗЛИ, ПОВ'ЯЗАНІ З ОКРЕМИМ ВУЗЛОМ КОНКРЕТНИМИ
ВІДНОШЕННЯМИ
2 – ОТРИМАТИ ІМЕНА ВСІХ ВІДНОШЕНЬ ДЛЯ ОКРЕМОГО ВУЗЛА
3 – ОТРИМАТИ ВСІ ПАРИ ВУЗЛІВ, ПОВ'ЯЗАНІ ОКРЕМИМ ВІДНОШЕННЯМ
4 – ВИХІД
ВВЕДІТЬ ПОТРІБНИЙ НОМЕР: 2
ВВЕДІТЬ ІМ'Я ВУЗЛА: РТАН
ВВЕДІТЬ ІМ'Я ВІДНОШЕННЯ: МАЕ
KRULA MAE
OPERENIA MAE

ВВЕДІТЬ ПОТРІБНИЙ НОМЕР;
1 – ОТРИМАТИ ВСІ ВУЗЛИ, ПОВ'ЯЗАНІ З ОКРЕМИМ ВУЗЛОМ КОНКРЕТНИМИ
ВІДНОШЕННЯМИ
2 – ОТРИМАТИ ІМЕНА ВСІХ ВІДНОШЕНЬ ДЛЯ ОКРЕМОГО ВУЗЛА
3 – ОТРИМАТИ ВСІ ПАРИ ВУЗЛІВ, ПОВ'ЯЗАНІ ОКРЕМИМ ВІДНОШЕННЯМ
4 – ВИХІД
ВВЕДІТЬ ПОТРІБНИЙ НОМЕР: 3
ВВЕДІТЬ ІМ'Я ВУЗЛА: МАЕ
ВВЕДІТЬ ІМ'Я ВІДНОШЕННЯ: МАЕ
РТАН KRULA
РТАН OPEREINA
```

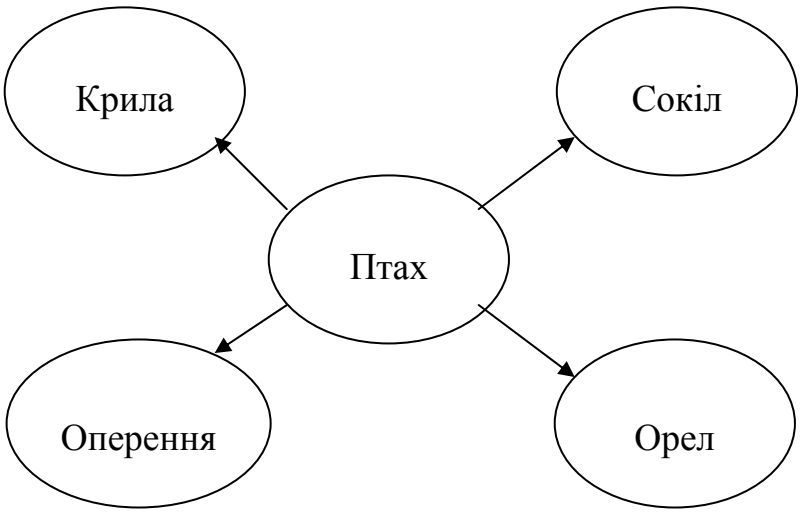

Рисунок 5 – Семантична мережа

Хід роботи

1. Ознайомитися з теоретичними відомостями.
2. Розробити програму для своєї семантичної мережі.
3. Підготувати звіт по лабораторній роботі. Зробити висновки.
4. Захистити лабораторну роботу.

Контрольні питання

1. Поняття семантичних мереж.
2. Правила використання семантичної мережі.
3. Які основні властивості семантичних мереж.

Література

1. Атре Ш. Структурный подход к организации баз данных. – М.: Финансы и статистика, 1983.-320 с.
2. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. – М.: Финансы и статистика, 1989.- 351 с.
3. Дейт К. Руководство по реляционной СУБД DB2. – М. : Финансы и статистика, 1988. – 320 с.
4. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. – М.: Мир, 1991.- 252 с.
5. Костин А.Е., Шальгин В.Ф. Организация и обработка структур данных в вычислительных системах. Учебное пособие для вузов. – М.: Высшая школа., 1987.-248 с.
6. Мартин Дж. Планирование развития автоматизированных систем. – М.: Финансы и статистика, 1984.- 196 с.
7. Месюра В.І., Романюк О.Н., Денисюк А.В. Методичні вказівки до виконання лабораторних робіт з дисципліни “Організація та управління базами даних” для студентів інженерної спеціальності 7.080403 “Програмне забезпечення обчислювальної техніки та автоматизованих систем”. – Вінниця: ВДТУ.
8. Мейер М. Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.
9. Системы управления базами данных и знаний: Справочное издание/ А.Н. Наумов, А.М. Вендров, В.К. Иванов и др./Под ред. А.Н. Наумова.-М.: Финансы и статистика, 1991. –352 с.
10. Тиори Т., Фрай Дж. Проектирование структур баз данных. В 2 кн. - М.: Мир, 1985. Кн. 1. – 287 с.: Кн. 2. –320 с.
11. Хаббард Дж. Автоматизированное проектирование баз данных. – М.: Мир, 1984. – 294 с.
12. Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных. – М.: Высшая школа, 1993.