Міністерство освіти і науки України

Чернівецький національний університет

імені Юрія Федьковича

Факультет комп’ютерних наук

Кафедра комп’ютерних систем та мереж

Курсова робота

**Опис мови Delphi**

2008

АНОТАЦІЯ

Даний розділ містить основні відомості про історію розвитку мови Object Pascal та середовища Delphi, їх основні технології та їх застосування для роботи файлами. Розділ містить також опис основних особливостей мови, основних елементів мови пов’язаних побудовою та відображенням діаграм.

Опис мови займає 25 сторінок друкованого тексту, 3 рисунки.

**Зміст**

Вступ

1. Загальні відомості

1.1 Історія розвитку середовища Delphi

1.2 Основні елементи середовища

1.3 Структура програми

2. Елементи та засоби Delphi

2.1 Принципи об'єктно-орієнтованого програмування

2.2 Використання класів

2.3 Опис компоненту TDBChart

2.4 Робота з файлами в Delphi

2.5 Основи вбудови ассемблерного коду

Висновки

Список літератури

# Вступ

На сучасному етапі розвитку комп’ютерних систем в світі розвинулася і поширилася велика кількість мов програмування, які призначені для різних областей людської діяльності. Визначальним моментом при виборі мови програмування додатків в більшості випадків являється досвід у використанні тієї чи іншої мови.

Згідно технічного завдання мені необхідно використати середовище Delphi для розробки архіватора на основі статистичного метода Хаффмана.

Delphi – є середовищем розробки програм, яке використовує багато передових ідей і концепцій, закладених в графічному інтерфейсі Windows. Як відомо, середовище розробки великою мірою визначає ефективність роботи програміста. У середовищі програмування Delphi є всі необхідні інструменти для того, щоб створювати повноцінні програми. Писати, компілювати і тестувати програму – все це можна робити, не виходячи з Delphi.

# 1. Загальні відомості

## 1.1 Історія розвитку середовища Delphi

Delphi — [мова програмування](http://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), що ґрунтується на діалекті мови [Pascal](http://uk.wikipedia.org/wiki/Pascal) від компанії [Borland](http://uk.wikipedia.org/w/index.php?title=Borland&action=edit&redlink=1). До версії 7.0 мала назву Object Pascal. Окрім того Delphi - середовище розробки ([IDE](http://uk.wikipedia.org/wiki/IDE)) для однойменної мови. Ця плутанина часто викликає непорозуміння. По суті, Delphi є об'єктно-орієнтованим розширенням мови [Pascal](http://uk.wikipedia.org/wiki/Pascal).

Delphi - це нащадок Турбо Паскаля, який був випущений для операційної системи Cp/m в 1983 році. У лютому 1994 року Турбо Паскаль був перенесений на операційну систему MS-DOS. На ранньому етапі розвитку комп'ютерів IBM РС, Турбо Паскаль був однією з найбільш популярних мов розробки програмного забезпечення - головним чином тому, що це був цілком серйозний компілятор, який, включаючи компілятор, редактор і відгадчик. Середовище мало змогу працювати на машині з 64 Kb оперативної пам'яті.

Під Windows – Турбо Паскаль був перенесений фірмою Borland в 1990 році. А найостанніша версія Borland Pascal 7.0 (що має тепер таку назву), не рахуючи Delphi, вийшла в світ в 1992 році. Розробка Delphi почалася в 1993 році. Після проведення beta-тестування Delphi показали на "Software Development '95".

Спочатку на Delphi можна було програмувати під MS [Windows](http://uk.wikipedia.org/wiki/Windows) 3.1. Починаючи з версії 2.0 на Delphi можна створювати програми під будь-яку з 32-бітних версій MS Windows.

В [2000](http://uk.wikipedia.org/wiki/2000) році була спроба створити варіант Delphi під операційну систему на базі ядра Linux, така модифікація Delphi мала назву [Kylix](http://uk.wikipedia.org/w/index.php?title=Kylix&action=edit&redlink=1). Було випущено 3 версії Kylix, проте експеримент виявився невдалим і [2003](http://uk.wikipedia.org/wiki/2003) року проект був заморожений.

[2003](http://uk.wikipedia.org/wiki/2003) року була створена модифікація мови під платформу Microsoft.NET, що отримала назву Delphi.NET. Цей варіант мови послідовно розвивається в версіях Delphi 8, 2005, 2006, 2007.

Частково Delphi підтримується також у відкритому проекті [FreePascal](http://uk.wikipedia.org/wiki/FreePascal), що потенційно дозволяє створювати програми під велику кількість платформ.

## 1.2 Основні елементи середовища

Інтегроване середовище Delphi складається з чотирьох основних елементів: головне вікно, вікно інспектора об’єктів, вікно форми та вікно модуля (вікно коду).

Головне вікно має заголовок Delphi 7.0 – Project1. Це вікно містить головне меню, панель кнопок швидкого доступу і палітру компонент.

* Головне меню – стандартне меню в стилі Windows. Це меню дозволяє керувати всіма аспектами роботи в Delphi. Рядок меню можна налаштувати за власним бажанням, наприклад, додати власні елементи до пункту меню інструментів Tools.
* Кнопки і гарячі клавіші. Кнопки використовуються для швидкого доступу до найнеобхідніших пунктів меню. Вони розташовані в лівій частині екрану на панелі швидкого доступу. Серед них є кнопки для компіляції і запуску програм, для перегляду вихідного коду рядок за рядком тощо. Для того, щоб з’ясувати призначення кнопки досить навести на неї вказівник миші і прочитати підказку. Панель швидкого доступу за замовчанням містить 14 кнопок, але її склад можна налогодити відповідно до вимог користувача. Більша частина найнеобхідніших функцій середовища Delphi також має гарячі клавіші, які можна натиснути замість відповідної кнопки чи то пункту меню.
* Палітра компонентів – це каталог візуальних і невізуальних об’єктів, які можна включати до власних форм и програм. У Delphi компоненти об’єднані в кілька основних груп: стандартна, додаткова, група Windows 95, група доступу до даних, група управління даними, група Windows 3.1, діалогова група, системна група, група звітів, OCX група і група взірців. Кожна з цих груп представлена на окремій сторінці палітри компонент. Щоб з’ясувати призначення компоненти, досить лише виділити її і натиснути F1.

Вікно, яке знаходиться у центрі, називається формою. Під час розробки форма являє собою вікно програми. У цьому вікні проходить основна частина роботи по проектуванню програми. Деякі елементи у вікні форми (лінії сітки, невізуальні компоненти) не будуть видимими під час виконання програми. Але, оскільки Delphi – це середовище програмування типу WYSIWYG (What – You – See – Is – What – You – Get, що бачите, те й отримаєте), то більша частина того, що ми бачимо під час проектування є тим, що ми побачимо і під час виконання програми. Є можливість змінити різні його властивості, наприклад, прибрати кнопки максимізації та мінімізації вікна тощо.

Вікно коду працює аналогічно до простого текстового редактора. Можна використовувати клавіші PgUp i PgDn, клавіші курсору, мишу, можна виділити, скопіювати, вставити текст за допомогою меню EDIT і відповідних гарячих клавіш.

Вгорі вікна коду є закладка. Вона належить до файлу, який зараз редагується. Якщо відкрити декілька файлів, кожен з них буде мати свою закладку.

Інспектор об’єктів або Object Inspector як правило знаходиться в лівій частині екрану і містить інформацію про виділений об’єкт. Інспектор об’єктів складається з таких елементів: комбінованої панелі (Combo box) вибору об’єкту, сторінки властивостей (Properties Page) та сторінки подій (Events Page) вибраного об’єкту.

У інспекторі об’єктів описані всі властивості об’єкту, і його використовують для зміни цих властивостей. Наприклад, можна змінити заголовок кнопки, клацнувши на ній мишкою, а потім записавши нову назву в полі Caption інспектора об’єктів.

Крім того за допомогою інспектору об’єктів можна переглянути та змінити всі події, що пов’язані з виділеним об’єктом.

## 

## 1.3 Структура програми

В основі будь-якої Delphi-програми лежить проект. Основою проекту в свою чергу є форма, на якій розміщаються необхідні для розв’язку конкретної задачі компоненти. Проект складається з різних частин, кожна з яких розміщена в окремому файлі й виконує чітко визначені функції. Набір файлів, необхідних для створення програми, називається проектом. Компілятор послідовно обробляє файли проекту і будує з них EXE-програму, що виконується. Проект складається з таких файлів:

* Головний файл проекту – текстовий файл з розширенням DPR, який містить головний програмний блок. Файл проекту підключає всі програмні модулі і містить методи для запуску програми. Цей файл створює і контролює автоматично середовище Delphi.
* Файли опису форм – двійкові файли з розширенням DFM, які описують форми з компонентами. В цих файлах запам’ятовуються початкові значення властивостей, встановлених в Інспекторі Об’єктів.
* Файли програмних модулів – текстові файли з розширенням PAS, які містять код на мові Object Pascal. У цих файлах містяться методи обробки подій, які генеруються компонентами і формами.
* Файл ресурсів з розширенням RES. У ньому, наприклад, зберігається піктограма програми, яку видно на Панелі Задач Windows.
* Об’єктні файли з розширенням OBJ, написані на інших мовах програмування.
* Файл опцій з розширенням DOF, де розміщені задані програмістом параметри компіляції і компонування проекту.
* Файл з розширенням DSK, де розміщені настройки візуального середовища для даного проекту.

У проект можуть входити також логічно автономні елементи: малюнки (BMP-файли), значки (ICO-файли), файли довідників (HLP-файли) і т. і., але ними управляє сам програміст.

Схема формування Delphi-програми від постановки задачі до отримання готового до виконання файла представлена на Рис.1.3.1.

Програма Задача Проект

Специфікація

Файли: проекту, форм, модулів, ресурсів, об’єктні, опцій, настройки і т. і.

EXE–програма

Рис. 1.3.1. Схема формування Delphi-програми.

# 2. Елементи та засоби Delphi

## 2.1 Принципи об’єктно-орієнтованого програмування

Об’єктно-орієнтоване програмування (ООП) — одна з [парадигм](http://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [програмування](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), яка розглядає програму як множину взаємодіючих об'єктів. У ній використано декілька технологій від попередніх парадигм, включаючи [успадкування](http://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%29), [модульність](http://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%29&action=edit&redlink=1), [поліморфізм](http://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%29) та [інкапсуляцію](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D1%8F%D1%86%D1%96%D1%8F). Не зважаючи на те, що ця парадигма з'явилась в [1960-тих](http://uk.wikipedia.org/wiki/1960-%D1%82%D1%96) роках, вона не мала широкого застосування до [1990-тих](http://uk.wikipedia.org/wiki/1990-%D1%82%D1%96). Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування [Симула](http://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D1%83%D0%BB%D0%B0) в 1960-тих роках, одночасно з посиленням дискусій про [кризу програмного забезпечення](http://uk.wikipedia.org/w/index.php?title=%D0%9A%D1%80%D0%B8%D0%B7%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F&action=edit&redlink=1).

Об'єктно-орієнтовані програми можна розглядати як множину взаємодіючих об'єктів, на відміну від традиційних поглядів, коли програма розглядається як набір [підпрограм](http://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%B0), або як перелік [інструкцій](http://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D1%8F) комп'ютеру. Відповідно до парадигми ООП, кожен об'єкт здатен отримувати [повідомлення](http://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BC%D1%96%D0%BD_%D0%BF%D0%BE%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC%D0%B8), обробляти дані, та надсилати повідомлення іншим об'єктам. Кожний об'єкт можна розглядати як незалежний автомат з окремим призначенням або відповідальністю.

Об’єктно-орієнтоване програмування засноване на трьох принципах:

* інкапсуляції;
* наслідуванні;
* поліморфізмі.

Спостережуване в об'єктах об'єднання даних і операцій в одне ціле було позначене терміном інкапсуляція. Застосування інкапсуляції зробило об'єкти схожими на маленькі програмні модулі і забезпечило приховування їх внутрішньої структури. Для об'єктів з'явилося поняття інтерфейсу, що значно підвищило їх надійність і цілісність.

Наслідування означає, що якщо ви хочете створити новий клас, лише небагато чим, що відрізняється від того, що вже існує, то немає необхідності в переписуванні наново всіх полів, методів і властивостей. Ви оголошуєте, що новий клас є нащадком (або дочірнім класом) наявного класу, так званим предком (або батьківським класом), і додаєте до нього нові поля, методи та властивості. Іншими словами додається те, що потрібне для переходу від загального до приватного. Процес створення нових класів на основі інших класів називається наслідуванням. Нові класи мають як успадковані ознаки, так і нові.

Поліморфізм означає, що в похідних класах ви можете змінювати роботу методів, що вже існують в базовому класі. При цьому весь програмний код, що керує об'єктами батьківського класу, придатний для управління об'єктами дочірнього класу без будь-якої модифікації.

## 2.2 Використання класів

Delphi підтримує дві моделі представлення об'єктів - стару і нову. Стара модель існує лише для сумісності з ранішими версіями компілятора, зокрема з Borland Pascal 7.0, тому ми її не розглядатимемо. Все, що сказане нижче, відноситься до нової моделі представлення об'єктів, могутнішої і багатшої по своїх можливостях. Для підтримка ООП в мову Object Pascal введена об'єктні типи даних, за допомогою яких одночасно описуються дані і операції над ними. Об'єктні типи називають класами, а їх екземпляри - об'єктами.

Класи об'єктів визначаються в секції type глобального блоку. Опис класу починається словом class і закінчується словом end. За формою оголошення класи схожі на звичайні записи, але крім полів даних можуть містити оголошення призначених для користувача процедур і функцій. Такі процедури і функції узагальнено називають методами, вони призначені для виконання над об'єктами різних операцій.

Заголовки методів, наступні за списком полів, грають роль попередніх (forward) оголошень. Програмний код методів поміщається нижче за визначення класу.

Клас, зазвичай, описує суть, що моделюється в програмі. Наприклад, клас Tdiskgauge описує вимірювач дискового простору. Клас містить два поля: Driveletter - буква накопичувача, що знаходиться під спостереженням, і Percentcritical - відсоток вільного простору на диску, з яким працює програма. Функція Getpercentfree визначена як метод роботи над будь-яким об'єктом класу Приведений вище опис є ні чим іншим, як декларацією інтерфейсу для управління об'єктами класу Tdiskgauge. Реалізація методів Getpercentfree і Checkstatus відсутня, але для створення і використання екземплярів класу вона поки і не потрібна. У цьому якраз і полягає сила інкапсуляції, яка робить об'єкти аналогічними програмним модулям. Для використання модуля необхідно вивчити лише його інтерфейсну частину, розділ реалізації для цього вивчати не потрібно.

## 2.3 Опис компоненту TDBChart

Для представлення даних з деякого набору даних у вигляді графіків різних видів призначений компонент Tdbchart. За його допомогою можна одночасно показувати графіки для декількох полів даних. Графіки будуються на основі всіх наявних в наборі даних значень полів.

Налаштування параметрів компоненту здійснюється спеціальним редактором, який можна відкрити подвійним клацанням на перенесеному на форму компоненті.

Основою будь-якого графіка в компоненті Tdbchart є так звана серія, властивості якої представлені класом Tchartseries. Для того, щоб побудувати графік значень деякого поля набору даних, необхідно виконати наступні дії, більшість з яких виконується в спеціалізованому редакторі компоненту:

1. Створити нову серію і визначити її тип.
2. Задати для серії набір даних.
3. Пов'язати з осями координат потрібні поля набору даних і, залежно від типу серії, задати додаткові параметри.
4. Відкрити набір даних.

Редактор має дві головні сторінки - Chart і Series. Сторінка Chart містить багатосторінковий блокнот і призначена для настройки параметрів самого графіка. Сторінка Series також містить багатосторінковий блокнот і використовується для настройки серій значень даних.

Для створення нової серії необхідно в редакторові перейти на головну сторінку Chart, а на ній відкрити сторінку Series. На цій сторінці потрібно клацнути на кнопці Add, а потім в діалозі, що з'явився, вибрати тип серії. Після цього в списку на сторінці Series з'являється рядок нової серії. Тут можна перевизначити тип, колір і видимість серії, клацнувши на відповідній зоні рядка. Решта всіх сторінок блокнота на головній сторінці Chart призначена для настройки параметрів графіка. Можна перейти на головну сторінку Series і на ній із списку назв серій вибрати необхідну. Після цього на сторінці Data Source із списку вибирається рядок Dataset. Далі в списку Dataset, що з'явився, вибирається потрібний набір даних.

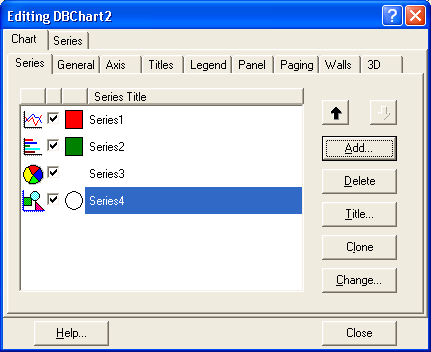


Рис. 2.3.1. Спеціалізований редактор компоненту Tdbchart.

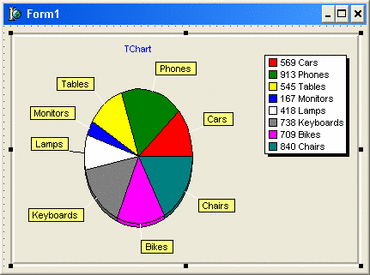


Рис. 2.3.2. Приклад використання компонента Tdbchart

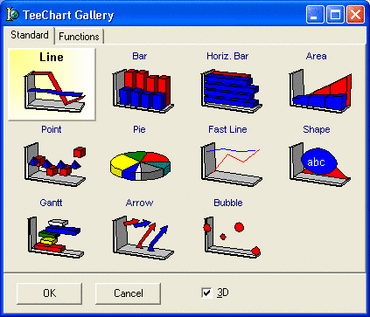


Рис. 2.3.3. Можливі види графіків

Список X дозволяє вибрати поле набору даних, значення якого послідовно відкладатимуться по осі абсцис. Список Y дозволяє вибрати поле набору даних, значення якого будуть відкладені по осі ординат. Відповідність між значеннями полів по двох осях визначається приналежністю до одного запису набору даних. Вибір поля в списку Labels прив'язує його значення у вигляді міток до осі абсцис.

Тепер залишилося тільки відкрити набір даних і компонент Tdbchart побудує графік. Аналогічним чином на цей же компонент можна помістити і інші графіки. Кожна серія відповідатиме одній кривій на графіку.

Властивості компонента TDBChart:

1. Allowpanning - визначає можливість користувача прокручувати спостережувану частину графіка під час виконання, натискаючи праву кнопку миші:

* pmnone - прокрутка заборонена;
* pmhorizontal - дозволена прокрутка тільки в горизонтальному напрямі;
* pmvertical - тільки у вертикальному pmboth - в обох напрвленіях.

1. Allowzoom - дозволяє користувачеві змінювати під час виконання масштаб зображення, вирізуючи фрагменти діаграми або графіка курсором миші.
2. Title - визначає заголовок діаграми.
3. Foot - визначає підпис під діаграмою. За замовчанням - відсутній. Текст підпису визначається підвластивістю Text.
4. Frame - визначає рамку навколо діаграми.
5. Legend - легенда діаграми (список позначень).
6. Marginleft, Marginright, Margintop, Marginbuttom - значення лівого, правого, верхнього і нижнього полів.
7. Buttomaxis, Leftaxis, Rightaxis - ці властивості визначають характеристики відповідно нижньої, лівої і правої осей. Завдання цих властивостей має сенс для графіків і деяких інших типів діаграм.
8. Leftwall, Buttomwall, Backwall - ці властивості визначають характеристики відповідно лівої, нижньої і задньої граней області тривимірного відображення графіка.
9. Serieslist - список серій даних, що відображаються в компоненті
10. View3d - дозволяє або забороняє тривимірне відображення діаграми.
11. View3doptions - характеристики тривимірного відображення.
12. Chart3dpersent - масштаб тривимірності (товщина діаграми, ширина стрічок графіка).

Редактор діаграм викликається:

* кнопкою з три крапкою поряд з назвою властивості в інспекторові об'єктів;
* подвійним клацанням на компоненті Chart при проектуванні форми;
* вибором команди Edit Chart в контекстному меню компоненту Chart при проектуванні форми.

Для задання значень, що відображаються, треба використовувати методи серій Series:

* clear - очищає серію від занесених раніше даних;
* add - дозволяє додати в діаграму нову точку:

Add(Const Avalue:double; Const Alabel:string; Acolor:tcolor)

Параметр Avalue відповідає значенню, що додається, параметр alabel - назва, яка буде відображатися на діаграмі і в легенді, параметр Acolor - колір. Параметр alabel необов'язковий, його можна задавати порожнім.

* addxy - дозволяє додати нову крапку в графік функції:
* ADDXY(Const Axvalue, Ayvalue: Double; Const Alabel: String; Acolor: Tcolor). Параметри Axvalue іayvalue відповідають аргументу і функції, параметри Alabel і Acolor - ті ж, що і в методі Add.

## 2.4 Робота з файлами в Delphi

Середовище Delphi дає вам можливість вибрати один з чотирьох варіантів роботи з файлами:

1. використання традиційного набору функцій роботи з файлами, успадкованого від Turbo Pascal;
2. використання функцій введення/виводу з Windows API;
3. використання потоків (rstream і його нащадки);
4. використання файлів, що відображаються.

Часто сучасний програмний код Delphi для читання даних з файлу дуже схожий на той, що використовується в Turbo Pascal. Програмісти Borland зберегли незмінним старий набір файлових функцій, що працюють через файлові змінні.

При організації операцій файлового введення/виведення у додатку велике значення має, якого роду інформація міститься у файлі. Частіше за все це - рядки, але зустрічаються двійкові дані або структурована інформація, наприклад масиви або записи.

Природно, що відомості про тип даних, що зберігаються у файлі важливо задати спочатку. Для цього використовуються спеціальні файлові змінні, що визначають тип файлу. Вони поділяються на ті, що не типізуються і типізуються. Перед початком роботи з будь-яким файлом необхідно описати файлову змінну, що відповідає типу даних цього файлу. Надалі ця змінна використовується при зверненні до файлу.

У Delphi є можливість створювати файли, що не типізуються. Для їх позначення використовується ключове слово file: var Untypedfile: file;

Такі файлові змінні використовуються для організації швидкого і ефективного введення/виводу незалежно від типу даних. При цьому мається на увазі, що дані читаються або записуються у вигляді двійкового масиву. Для цього застосовуються спеціальні процедури блокового читання і запису.

Типізовані файли, забезпечують введення/виведення з урахуванням конкретного типу даних. Для їх оголошення використовується ключове слово file of, до якого додається конкретний тип даних. Наприклад, для роботи з файлом, що містить набір байтів, файлова змінна оголошується так: var Bytefile: file of byte;

При цьому можна використовувати будь-які типи фіксованого розміру, за винятком вказівників. Дозволяється застосовувати структурні типи, якщо їх складові частини задовольняють названому вище обмеженню. Наприклад, можна створити файлову змінну для запису:

type Country = record

Name: String;

Capital: String;

Population: Longlnt;

Square: Longlnt;

end;

var Countryfile: file of Country;

Для роботи з текстовими файлами використовується спеціальна файлова змінна Textfile або Text: var F: Textfile;

Тепер розглянемо дві найпоширеніші операції, що виконуються при роботі з файлами. Це читання і запис. Для їх здійснення застосовуються спеціальні функції файлового введення/виводу.

Отже, для виконання операції читання або запису необхідно провести наступні дії:

1. Оголосити файлову змінну необхідного типу.

2. За допомогою функції Assignfile пов'язати цю змінну з необхідним файлом.

3. Відкрити файл за допомогою функцій Append, Reset, Rewrite.

4. Виконати операції читання або запису. При цьому, залежно від складності завдання і структури даних, може використовуватися цілий ряд допоміжних функцій.

5. Закрити файл за допомогою функції Closefile.

Як приклад розглянемо невеликий фрагмент коду.

...

var F: Textfile;

S: string;

begin

if Opendlg.Execute

then Assignfiie(F, Opendlg.FileName)

else Exit; Reset(F);

while Not EOF(F) do

begin

Readln(F, S);

Memo.Lines.Add(S);

end;

Closefile(F);

end;

...

Якщо в діалозі відкриття файлу Opendlg був вибраний файл, то його ім'я зв'язується з файловою змінною F за допомогою процедури Assignfile. Як ім'я файлу рекомендується завжди передавати повне ім'я файлу (включаючи його маршрут). Якраз у такому вигляді повертають результат вибору файлу діалоги роботи з файлами. Потім за допомогою процедури Reset цей файл відкривається для читання і запису.

У циклі виконується читання з файлу текстових рядків і запис їх в компонент Tmemo. Процедура Readin здійснює читання поточного рядка файлу і переходить на наступний рядок. Цикл виконується, поки функція EOF не повідомить про досягнення кінця файлу. Після завершення читання файл закривається. Такий же код можна використовувати і для запису даних у файл. Необхідно тільки замінити процедуру читання на процедуру запису.

Відкриття файлу може здійснюватися трьома процедурами - залежно від типу його подальшого використання.

1. Процедура procedure Reset(var F: File [; Recsize: Word ]); відкриває існуючий файл для читання і запису, поточна позиція встановлюється на першому рядку файлу.
2. Процедура procedure Append(var F: Text); відкриває файл для запису інформації після його останнього рядка, поточна позиція встановлюється на кінець файлу.
3. Процедура procedure Rewrite(var F: File [; Recsize: Word ]); створює новий файл і відкриває його, поточна позиція встановлюється в початок файлу. Якщо файл з таким ім'ям вже існує, то він перезаписується.

Змінна Recsize використовується тільки при роботі з файлами, що не типізуються, і визначає розмір одного запису для операції передачі даних. Якщо цей параметр опущений, то за замовчуванням Recsize рівне 128 байт.

Читання даних з файлів, що типізуються і текстових, виконують процедури Read і Readin. Процедура Read має різне оголошення для текстових і інших файлів, що типізуються:

* procedure Read([var F: Text;] VI [, V2...,Vn]) - для текстових файлів;
* procedure Read(F, VI [, V2...,Vn]) - для інших типізованих файлів.

При одному виклику процедури можна читати дані в довільне число змінних. Звичайно, що тип змінних повинен співпадати з типом файлу. При читанні в чергову змінну читається рівно стільки байтів з файлу, скільки займає тип даних. У наступну змінну читається стільки ж байтів, розташованих услід. Після виконання процедури поточна позиція встановлюється на першому непрочитаному байті. Аналогічно працюють декілька процедур Read для однієї змінної, виконаних підряд.

Процедура procedure Readln([ var F: Text; ] VI [, V2...,Vn ]); прочитує один рядок текстового файлу і встановлює поточну позицію на наступному рядку.

Процедури для запису у файл write і writein описані аналогічно: procedure Write([var F: Text; ] PI [, P2..., Pn]); procedure Writein([ var F: Text; ] PI [, P2...,Pn ]). Параметри P1, P2..., Pn можуть бути одним з цілих або речових типів, одним із строкових типів або логічним типом. Але у них є можливість додаткового форматування при виводі. Кожен параметр запису може мати форму: Рn [: Minwidth [: Decplaces ] ], де Рn - змінна, що виводиться, або вираз; Minwidth - мінімальна ширина поля в символах, яка повинна бути більше 0; Decplaces - містить кількість десяткових символів після коми при відображенні дійсних чисел з фіксованою крапкою.

Для текстових файлів у функціях Read і write файлова змінна F може бути відсутня. У цьому випадку читання і запис здійснюються в стандартні файли введення/виводу. Коли програма компілюється як консольне застосування, Delphi автоматично пов'язує вхідний і вихідний файли з вікном консолі.

Для контролю за поточною позицією у файлі застосовуються дві основні функції.

1. Функція EOF(F) повертає значення True, якщо досягнутий кінець файлу.
2. Функція EOLN(F) аналогічно сигналізує про досягнення кінця рядка. Природно, як параметр у функції необхідно передавати файлову змінну.

Процедура procedure Seek(var F; N: Longint); забезпечує зсув поточної позиції на N елементів. Розмір одного елементу в байтах залежить від типу даних файлу.

Розглянемо тепер режим блокового введення/виводу даних між файлом і областю адресного простору (буфером). Цей режим відрізняється значною швидкістю передачі даних, причому швидкість пропорційна розміру одного передаваного блоку - чим більше блок, тим більше швидкість.

Для реалізації цього режиму необхідно використовувати файлові змінні, що тільки не типізуються. Розмір блоку визначається в процедурі відкриття файлу (Reset, Rewrite). Безпосередньо для виконання операцій використовуються процедури Blockread і Blockwrite. Процедура procedure Blockread(var F: File; var Buf; Count: Integer [; var Amttransferred: Integer]); виконує запис блоку з файлу в буфер. Параметр F посилається на файлову змінну, що не типізується, пов'язану з потрібним файлом.

Параметр Buf визначає будь-яку змінну (число, рядок, масив, структуру), в яку читаються байти з файлу. Параметр Count містить число прочитуваних блоків. Нарешті, необов'язковий параметр Amttransferred повертає число реально лічених блоків.

При використанні блокового читання або запису розмір блоку необхідно вибирати так, щоб він був кратний розміру одного значення того типу, який зберігається у файлі. Наприклад, якщо у файлі зберігаються значення типу Double (8 байт), то розмір блоку може бути рівний 8, 16, 24, 32 і так далі.

## 2.5 Основи вбудови ассемблерного коду

## Якщо потрібно використовувати команди Асемблера не тільки процесора 8086, але і математичного співпроцесора 8087, то необхідно включати директиву компілятора {$N+}. А для доступу до команд процесора 80286 необхідна директива {$G+}. Таким чином комбінація {$G+,$N+} дасть доступ до всіх команд, включаючи співпроцесор 80287. Наступна особливість вбудованого Ассемблера полягає в тому, що в ньому можна звертатися до імен, визначених в програмі на Паскалі. Наприклад, можна читати значення змінних або викликати процедури прямо з Ассемблерної вставки. Ось, приклад: mov AX, Summa, де Summa - змінна, оголошена в розділі var. Причому розмірність змінної в даному випадку не важлива і контролювати збіг типів змінних доводиться у ручну. Є специфічні змінні, доступні тільки з Ассемблерних вставок:

## @Code - поточний сегмент коду;

## @Data - поточний сегмент даних;

## @Result - посилання усередині функції на її результат.

## Наприклад:

## Code (Pascal):

## mov AX, SEG @Data mov DS, AX

## Для доступу до змінних з програми на Паскалі, імена яких співпадають із зарезервованими словами Асемблера використовується операція перекриття ідентифікатора. У Ассемблерних вставках окрім звичайних міток Паскаля, що оголошуються в розділі label можна використовувати і не оголошені, в цьому випадку необхідно перед такою міткою ставити знак @.

Для того, що б вставити Ассемблерний код в програму на Паскалі, можна скористатися складеним оператором asm... end. Разом із звичайним складеним оператором begin... end Паскаль підтримує і складений оператора asm... end в який записується код не на Паскалі, а на Ассемблері.

Приклад:

var A, B, Max: Integer;

begin Write ('Введіть A: ');

Readln (A); Write ('Введіть B: ');

Readln (B);

asm

mov AX, A

cmp AX, B

jg @e

mov AX, B

@e: mov Max, AX

end; {asm} Writeln ('Max = ', Max);

Readln; end.

Як видно з коду, програма видасть максимальне значення між змінними A і B.

**Висновки**

Мова програмування Delphi дуже широко використовується і має дуже велику кількість розроблених елементів та засобів, що значно полегшують розробку додатків. У Delphi, використовується багато передових ідей і концепцій, закладених в графічному інтерфейсі Windows. У середовищі програмування Delphi є всі необхідні інструменти для того, щоб створювати повноцінні програми. Писати, компілювати і тестувати програму – все це можна робити, не виходячи з Delphi.

Завдяки можливості вбудови в програму ассемблерних вставок, програміст отримує змогу створювати програми з інтерфейсом Windows та дуже високою швидкодією, характерною для низькорівневих мов програмування.

# Список літератури

1. Delphi 7 / под общ. ред. А.Д. Хомоненко. – СПб.: БХВ-Петербург, 2007. – 1216 с.
2. Assembler. Учебник для вузов. 2-е изд. – СПб.: Питер, 2007. – 637с.
3. Архангельский А.Я. Приемы программирования в Delphi на основе VCL. — М.: «Бином-Пресс», 2006 г. — 944 с.
4. Фленов М.Е. Библия Delphi. – СПб.: БХВ-Петербрг, 2004. – 880 с.
5. Стивенс Р. Delphi. Готовые алгоритмы / Род Стивенс; Пер. с англ. Мерещука П.А. – 2-е изд., стер. – М.: ДМК Пресс; СПб.: Питер, 2004. – 384 с.: ил.