

**TRƯỜNG ĐẠI HỌC CẦN THƠ
KHOA KHOA HỌC TỰ NHIÊN**



Giáo trình lý thuyết

LẬP TRÌNH CĂN BẢN B

MSMH: TN035



(Áp dụng cho chương trình tín chỉ)

**Biên soạn: ThS. VŨ DUY LINH
ThS. NGUYỄN NHỊ GIA VINH
ThS. LÊ THỊ DIỄM**

Năm 2010

MỤC LỤC

<i>CHƯƠNG 1: DỮ LIỆU VÀ THUẬT TOÁN</i>	1
I. Dữ liệu và thông tin:	1
I.1. Khái niệm	1
I.2. Sơ đồ tổng quát của một quá trình xử lý thông tin.....	1
II. Từ bài toán tới chương trình.....	1
II.1. Các giai đoạn giải một bài toán trên máy tính điện tử.....	1
II.2. Thuật toán (Algorithm).....	2
 <i>CHƯƠNG 2: BORLAND DELPHI</i>	5
I. Tổng quan về Delphi	5
I.1. Delphi là gì.....	5
I.2. Các phiên bản của Delphi	5
II. Môi trường phát triển tích hợp (IDE) của Delphi	5
II.1. Cửa sổ chính của Delphi.....	6
II.2. Thanh thực đơn chính và thanh công cụ.....	6
II.3. Bảng chứa các thành phần của Delphi (Component Palette)	7
II.4. Cửa sổ thiết kế biểu mẫu (Form Designer) và Cửa sổ soạn thảo mã lệnh (Code Editor)	8
II.5. Cửa sổ thuộc tính và sự kiện của đối tượng (Object Inspector)	9
II.6. Cửa sổ liệt kê các đối tượng dạng cây (Object TreeView).....	10
III. Cấu trúc một dự án Delphi	10
III.1. Tập tin dự án: (Delphi project file)	10
III.2. Các tập tin chứa mã lệnh (Unit file)	11
III.3. Các tập tin đặc tả biểu mẫu (Form file).....	11
III.4. Các tập tin tài nguyên (Windows Resource File).....	13
IV. Các thao tác cơ bản trên Delphi	14
IV.1. Mở một dự án mới.....	14
IV.2. Lưu dự án	14
IV.3. Lưu Form (tập tin unit) với tên khác	18
IV.4. Lưu dự án với tên khác.....	18
IV.5. Đóng dự án	18
IV.6. Thoát khỏi Delphi	18
 <i>CHƯƠNG 3: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ DELPHI (OBJECT PASCAL)</i>	19
I. Bộ chữ viết	19
II. Tùy khóa	19
III. Tên danh hiệu tự đặt	20
IV. Hằng	20
IV.1. Khái niệm	20
IV.2. Hằng trị.....	20

IV.3. Hằng định kiểu:	21
V. Kiểu (Type).....	21
V.1. Định nghĩa	21
V.2. Cách khai báo	21
VI. Biến	22
VII. Biểu thức	22
VII.1. Định nghĩa.....	22
VII.2. Thứ tự ưu tiên.....	22
VIII. Chuyển kiểu (Typecast).....	23
IX. Lời chú thích và các chỉ dẫn biên dịch	24
X. Cấu trúc một dự án ở chế độ Form (Form Application)	25

CHƯƠNG 4: CÁC KIỂU DỮ LIỆU SƠ CẤP CHUẨN, LỆNH ĐƠN.....26

I. Các kiểu dữ liệu sơ cấp (Simple type)	26
I.1. Kiểu số nguyên (Integer).....	26
I.2. Kiểu ký tự (Char)	26
I.3. Kiểu số thực (Real type).....	30
I.4. Kiểu logic (Boolean)	34
II. Câu lệnh (statement)	34
III. Lệnh đơn (Simple statement)	35
IV. Lệnh gán (Assignment statement).....	35
V. Lệnh gọi thủ tục và hàm	35
VI. Lệnh Goto.....	37

CHƯƠNG 5: LẬP TRÌNH XỬ LÝ SỰ KIỆN – CÁC THÀNH PHẦN TRONG GIAO DIỆN DELPHI.....38

I. Lập trình xử lý sự kiện	38
I.1. Lớp (Class) và đối tượng (Object)	38
I.2. Thuộc tính	43
I.3. Phương thức (Method)	44
I.4. Sự kiện (Event).....	44
I.5. Xử lý sự kiện (Event Handlers).....	45
I.6. Trình hỗ trợ mã lệnh (Code Completion /IntelliSense).....	46
II. Sinh mã tự động và một số cách sửa lỗi	46
II.1. Cách tự động sinh mã (generate code) trong Delphi	46
II.2. Cách sửa lỗi sinh mã trong Delphi.....	49
II.3. Thêm tập tin unit vào dự án	54
II.4. Viết lệnh sau khi thêm unit form vào dự án.....	58
II.5. Gỡ bỏ tập tin unit form ra khỏi dự án	62
III. Biểu mẫu (TForm)	62
IV. Các thành phần (Component) giao diện phổ biến	68
IV.1. Nhãn (TLabel)	68
IV.2. Hộp văn bản (TEdit).....	71

IV.3. Nút lệnh (TButton)	78
IV.4. Nhãn và Hộp nhập (TLabeledEdit)	80
IV.5. Hộp đánh dấu hay hộp kiểm (TCheckbox)	81
IV.6. Nút tùy chọn (TRadioButton)	87
IV.7. Nhóm tùy chọn (TRadioGroup)	88
IV.8. Vùng văn bản (TMemo)	91
IV.9. Hộp danh sách (TListBox)	96
IV.10. TSpinEdit	100
IV.11. Hộp danh sách đánh dấu (TCheckListBox)	101
IV.12. Hộp danh sách các khoá (TValueListEditor)	107
IV.13. Hộp liệt kê thả (TComboBox)	113
IV.14. Lưới chuỗi (TStringGrid)	118
IV.15. Bảng chứa các thành phần (TPanel)	124
IV.16. Thanh thực đơn chính (TMainMenu)	125
IV.17. Thực đơn (TMenuItem)	126
IV.18. Menu đôi tượng (TPopupMenu)	134

CHƯƠNG 6: CÁC LỆNH CÓ CẤU TRÚC.....138

I. Lệnh ghép (Compound statement)	138
II. Lệnh cấu trúc rẽ nhánh	138
II.1. Lệnh if ... then ... và lệnh if ... then ... else	138
II.2. Lệnh Case ... of	146
III. Cấu trúc lệnh lặp.....149	
III.1. Lệnh lặp có số lần xác định trước	149
III.2. Lệnh lặp có số lần không xác định trước	155

CHƯƠNG 7: CHƯƠNG TRÌNH CON.....164

I. Khái niệm.....164	
II. Hàm.....165	
III. Thủ tục.....172	
IV. Truyền tham số.....175	
IV.1. Định kiểu và không định kiểu cho tham số hình thức.....175	
IV.2. Truyền bằng tham trị (Value parameter).....177	
IV.3. Truyền bằng tham biến (Variable parameter)	179
IV.4. Truyền bằng tham số hằng (Constant parameter)	181
IV.5. Truyền bằng tham số xuất (Out parameter)	184
V. Chương trình con đệ quy	190

CHƯƠNG 8: KIẾU LIỆT KÊ, MIỀN CON, TẬP HỢP194

I. Kiểu vô hướng liệt kê (Enumerated scalar type)	194
I.1. Khái niệm.....194	
I.2. Cách khai báo: Có hai cách khai báo là gián tiếp và trực tiếp	194
I.3. Một số hàm chuẩn áp dụng cho kiểu vô hướng	195

II. Kiểu miền con (Subrange types).....	196
II.1. Khái niệm.....	196
II.2. Cách khai báo.....	196
III. Kiểu tập hợp (Set).....	197
III.1. Khái niệm	197
III.2. Cách khai báo	197
III.3. Mô tả tập hợp.....	198
III.4. Một số phép toán trên kiểu tập hợp	198
 <i>CHƯƠNG 9: KIỂU MẢNG</i>	208
I. Khái niệm về mảng (Array-type data)	208
II. Mảng tĩnh (Static array)	208
II.1. Mảng một chiều (One-Dimensional array)	208
II.2. Mảng nhiều chiều (Multi-Dimensional array)	213
II.3. Mảng hai chiều.....	215
III. Mảng động (Dynamic array)	222
III.1. Mảng động một chiều.....	222
III.2. Mảng động nhiều chiều	227
III.3. Hằng mảng.....	228
 <i>CHƯƠNG 10: KIỂU CHUỖI KÝ TỰ</i>	229
I. Các loại chuỗi ký tự trong Object Pascal	229
I.1. Chuỗi ngắn (ShortString)	229
I.2. Chuỗi dài 1 byte (AnsiString)	230
I.3. Chuỗi dài 2 byte (WideString)	232
II. Các thao tác trên chuỗi.....	232
II.1. Phép toán cộng chuỗi	232
II.2. Phép toán so sánh.....	232
II.3. Các thủ tục và hàm chuẩn xử lý chuỗi ký tự.....	234
 <i>PHỤ LỤC</i>	246
<i>TÀI LIỆU THAM KHẢO</i>	252
<i>BẢN QUYỀN TÁC GIẢ</i>	252

CHƯƠNG 1: DỮ LIỆU VÀ THUẬT TOÁN

I. Dữ liệu và thông tin:

I.1. Khái niệm

Dữ liệu (Data) là các sự kiện không có cấu trúc, không có ý nghĩa rõ ràng tại thời điểm xét.

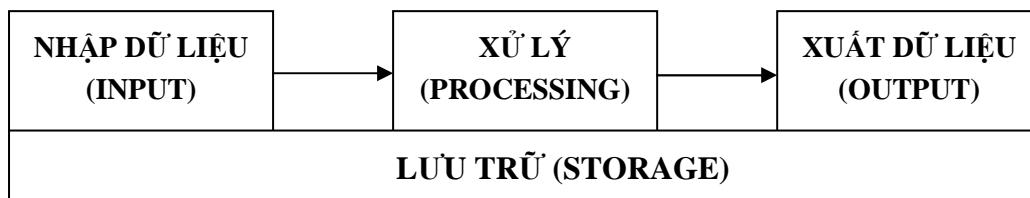
Thông tin (Information) là một khái niệm trừu tượng được thể hiện qua các thông báo, các biểu hiện,... đem lại một nhận thức chủ quan cho một đối tượng nhận tin. Thông tin là dữ liệu đã được xử lý xong, mang ý nghĩa rõ ràng tại thời điểm xét.

Một hệ thống thông tin (Information system) là một tiến trình ghi nhận dữ liệu, xử lý nó và cung cấp tạo nên dữ liệu mới có ý nghĩa thông tin, liên quan mật màng đến một tổ chức, để trợ giúp các hoạt động liên quan đến tổ chức.

I.2. Sơ đồ tổng quát của một quá trình xử lý thông tin

Mọi quá trình xử lý thông tin bằng máy tính hay bằng con người đều được thực hiện theo một quy trình sau:

Dữ liệu được nhập ở đầu vào (Input). Máy tính hay con người sẽ thực hiện quá trình xử lý nào đó để nhận được thông tin ở đầu ra (Output).



Hình 1: Mô hình tổng quát quá trình xử lý thông tin

II. Từ bài toán tới chương trình

Trong cuộc sống, chúng ta có rất nhiều bài toán cần phải giải quyết. Sử dụng máy tính điện tử để giải các bài toán là điều rất hiệu quả do khả năng tính toán của máy tính. Để máy tính có thể hiểu được, chúng ta cần phải có những bước giải cụ thể của bài toán một cách đúng logic, từ đó sử dụng một ngôn ngữ chương trình hay phần mềm lập trình để hướng dẫn máy tính thực hiện các thao tác cần thiết để tìm ra kết quả của bài toán.

II.1. Các giai đoạn giải một bài toán trên máy tính điện tử

Để giải quyết một bài toán trên máy tính điện tử, cần qua các giai đoạn sau:

- Tìm hiểu mục tiêu chính của bài toán: dữ liệu nhập vào và kết quả xuất.
- Xây dựng một chuỗi thao tác tính toán theo một thứ tự logic, gọi là thuật giải.
- Lập chương trình diễn tả chi tiết các bước tính theo thuật giải.
- Nhập chương trình vào máy tính, thông dịch và chạy thử để sửa chữa lỗi.
- Thực hiện giải bài toán với số liệu thu thập được và ghi nhận kết quả.

- Thử nghiệm với nhiều trường hợp dữ liệu nhập khác nhau của bài toán để chương trình kiểm tra đúng trong mọi trường hợp tổng quát.
- Phân tích kết quả và hoàn chỉnh chương trình.

Trong các bước trên, việc thiết kế thuật toán là giai đoạn quan trọng nhất.

II.2. Thuật toán (Algorithm)

II.2.1. Định nghĩa

Thuật toán là một phương pháp trình bày các bước giải quyết một hay nhiều bài toán theo một tiến trình xác định.

II.2.2. Đặc tính của thuật toán

- Tính xác định: Các thao tác của thuật toán là rõ ràng và chắc chắn thực hiện được để dẫn đến kết quả nào đó.
- Tính hữu hạn và dừng: thuật toán phải có một số bước giải nhất định và cuối cùng phải có kết thúc ở điểm dừng.
- Tính kết quả: Với dữ liệu hợp lý, thuật toán phải cho kết quả thỏa yêu cầu.
- Tính phổ dụng: Thuật toán phải giải được nhiều bài toán có cùng cấu trúc với các dữ liệu khác nhau và đều dẫn đến một kết quả mong muốn.
- Tính hiệu quả: Thuật giải phải đơn giản, dễ hiểu trong các bước giải, tối thiểu hóa bộ nhớ và thời gian thực hiện.
- Tính hình thức: Các bước trong thuật toán là máy móc, nghĩa là nó phải thực hiện đúng như quy định mà không cần biết đến mục tiêu cuối cùng.

II.2.3. Các phương pháp biểu diễn thuật toán

Thuật toán có thể được diễn giải thông qua 3 phương pháp phổ biến, đó là sử dụng ngôn ngữ tự nhiên, ngôn ngữ giả và ngôn ngữ sơ đồ (lưu đồ).

II.2.3.1 Ngôn ngữ tự nhiên

Là cách diễn đạt tự nhiên của con người bằng ngôn ngữ của mình để mô tả các bước giải bài toán.

Ví dụ 1: Mô tả thuật toán theo ngôn ngữ tự nhiên của phương trình $ax + b = 0$

- Bước 1: Nhập vào 2 hệ số a và b.
- Bước 2: Xét điều kiện $a = 0$?

Nếu đúng là $a = 0$, thì đi đến bước 3, nếu không (nghĩa là $a \neq 0$) thì đi đến bước 4.

- Bước 3: Xét điều kiện $b = 0$?

Nếu $b = 0$, thì báo phương trình có vô số nghiệm. Đi đến bước 5.

Nếu $b \neq 0$, thông báo phương trình vô nghiệm. Đi đến bước 5.

- Bước 4: Thông báo phương trình có một nghiệm duy nhất là $x = -b/a$.
- Bước 5: Kết thúc thuật toán.

II.2.3.2 Ngôn ngữ giả

Là cách diễn đạt các bước giải của bài toán thông qua một số câu lệnh có cấu trúc (như câu lệnh if..then..else, for..to..do, while..do, repeat..until)

Ví dụ 2: Mô tả thuật toán theo ngôn ngữ giả của phương trình bậc nhất $ax + b = 0$

- Bước 1: Nhập vào 2 hệ số a và b.
- Bước 2: **if** ($a=0$) **then**

if ($b=0$) **then** phương trình vô số nghiệm

else phương trình vô nghiệm

else phương trình có một nghiệm duy nhất $x = -b/a$

- Bước 3: Kết thúc thuật toán.

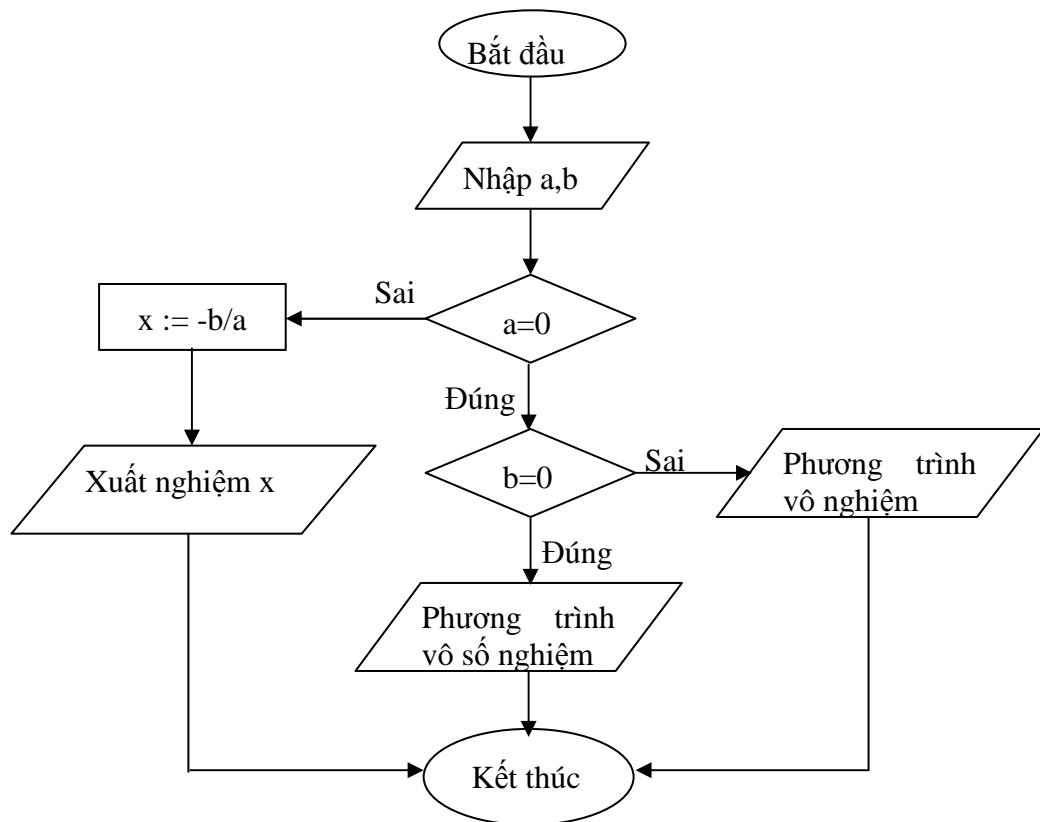
II.2.3.3 Ngôn ngữ lưu đồ (Flowchart)

Là cách diễn đạt các bước giải của bài toán thông qua các ký hiệu của lưu đồ. Một số qui ước ký hiệu lưu đồ:

Ký hiệu	Mô tả
	Điểm bắt đầu hoặc kết thúc thuật toán
	Thao tác nhập hay xuất
	Khối xử lý công việc
	Khối quyết định chọn lựa
	Điểm nối
	Chuẩn bị
	Tập hợp các tập tin dữ liệu
	Khối chương trình con
	Các ghi chú, giải thích
	Dòng tính toán, thao tác của chương trình

Hình 2: Các ký hiệu lưu đồ

Ví dụ 3: Mô tả thuật toán theo ngôn ngữ lưu đồ của phương trình $ax + b = 0$



Hình 3: Lưu đồ thuật toán PTB1

CHƯƠNG 2: BORLAND DELPHI

I. Tổng quan về Delphi

I.1. Delphi là gì

Delphi là một ngôn ngữ lập trình cấp cao, có trình biên dịch hoàn hảo, hỗ trợ mạnh về các kiểu dữ liệu có cấu trúc và thiết kế hướng đối tượng dựa trên nền tảng ngôn ngữ lập trình hướng đối tượng (OOP: Object-Oriented Programming) của Borland Pascal. Ngày nay, Delphi đã được phát triển thành môi trường xây dựng ứng dụng tức thời RAD (Rapid Application Development). Từ những công cụ của RAD, bạn có thể giải quyết những vấn đề phức tạp trong qua trình phát triển phần mềm như: lập trình ứng dụng về cơ sở dữ liệu (Database), lập trình mạng và Internet (Internet/Networking), lập trình Multimedia (Animation, sound), lập trình trò chơi (Game) cũng như đồ họa (Graphic) hoặc lập trình hệ thống, v.v... không những trên nền Windows mà còn cho cả Linux và .NET. Trong giáo trình này, chúng tôi chỉ giới thiệu các khái niệm của Delphi trong phạm vi hệ điều hành Windows. Với khả năng mạnh như vậy của Delphi, bạn an tâm khi dùng nó để triển khai các ứng dụng của bạn ở quy mô nhỏ hay lớn. Điều cần quan tâm ở Delphi là một ngôn ngữ rất thân thiện với người dùng, phù hợp cho những người bắt đầu làm quen với nó cũng như những nhà lập trình chuyên nghiệp.

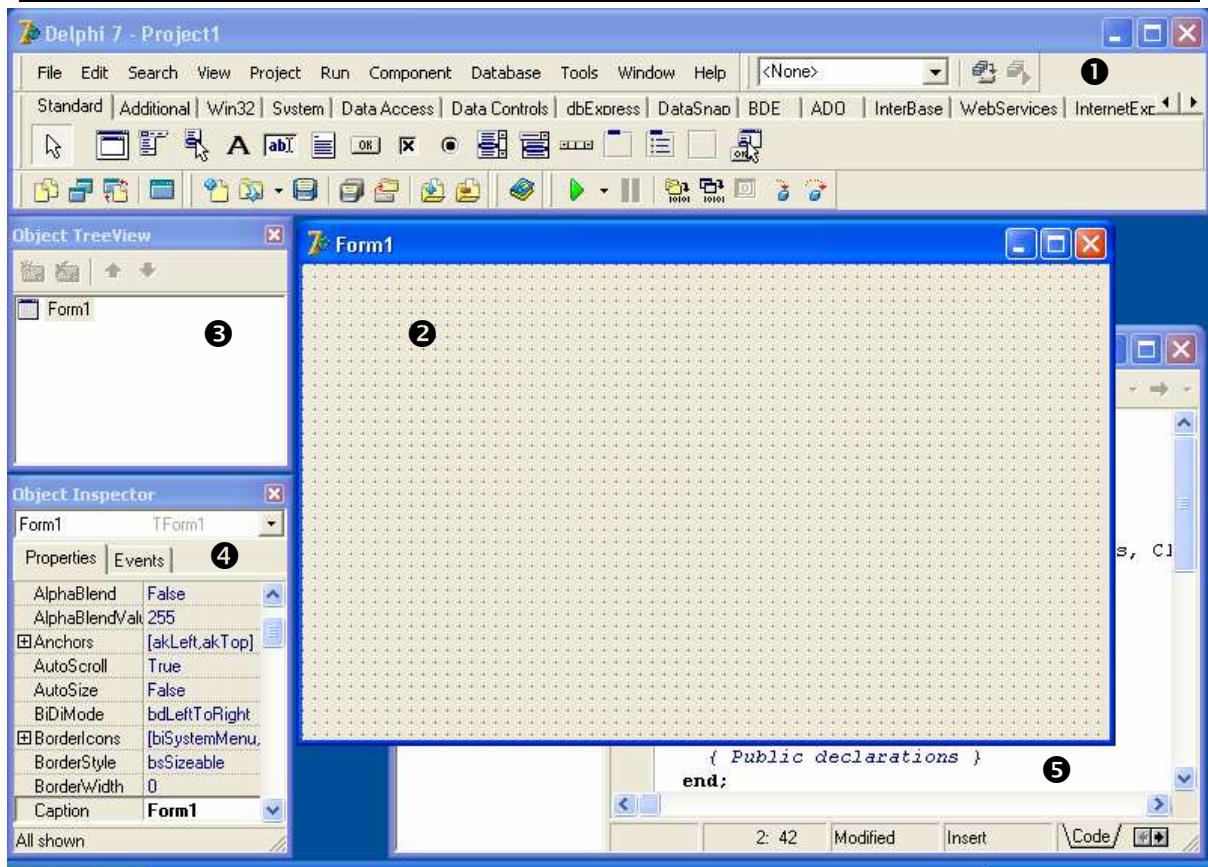
Vì đây là giáo trình được soạn dành cho đối tượng là các sinh viên bắt đầu học Tin học Đại cương thông qua ngôn ngữ Delphi. Do vậy, giáo trình này chỉ giới thiệu những khái niệm căn bản nhằm giúp cho người học có một số kiến thức nhất định đủ để vận dụng vào giải các bài toán hoặc viết những chương trình ứng dụng nhỏ. Các chương trong giáo trình này sẽ trình bày các phần tử cơ bản của Delphi để tạo một chương trình chạy trên nền Windows, đó chính là môi trường phát triển tích hợp IDE (Integrated Development Environment) và ngôn ngữ hướng đối tượng Borland Pascal. Các bạn sẽ được học cách thiết kế (Design), phát triển (Develop) hay viết mã lệnh (Code), và kiểm tra lỗi (Test) chương trình ứng dụng bởi việc sử dụng Delphi.

I.2. Các phiên bản của Delphi

Tiền thân của Delphi chính là ngôn ngữ đối tượng Borland Pascal, và đến ngày nay Delphi đã có một quá trình phát triển vững mạnh từ phiên bản 1 (Delphi1) vào năm 1995, đến phiên bản 8 (Delphi for .NET) năm 2005. Trong giáo trình này, chúng tôi thống nhất trình bày với các bạn Delphi 7. Đây là phiên bản phù hợp nhất và dễ dùng nhất hiện nay cho các bạn, đối tượng là các sinh viên học Tin học Đại cương.

II. Môi trường phát triển tích hợp (IDE) của Delphi

Môi trường soạn thảo và thiết kế ứng dụng của Delphi 7 chia ra làm 5 phần: Cửa sổ chính của chương trình Delphi, cửa sổ thiết kế biểu mẫu (Form designer), cửa sổ liệt kê các thành phần, đối tượng dạng cây (Object TreeView), cửa sổ thiết lập thuộc tính đối tượng (Object Inspector), và cửa sổ soạn thảo mã lệnh (Code Editor). Với môi trường IDE này, bạn sẽ có một giao diện (Interface) để thiết kế (Design), biên dịch (Compile) và gỡ lỗi (Debug) dự án mà bạn đang phát triển.



Hình 1: Giao diện của dự án mới tạo trong Delphi 7

- ① Cửa sổ chính của Delphi 7
- ② Cửa sổ thiết kế biểu mẫu
- ③ Cửa sổ liệt kê các đối tượng dạng cây
- ④ Cửa sổ thiết lập các thuộc tính và sự kiện của đối tượng
- ⑤ Cửa sổ soạn thảo mã lệnh

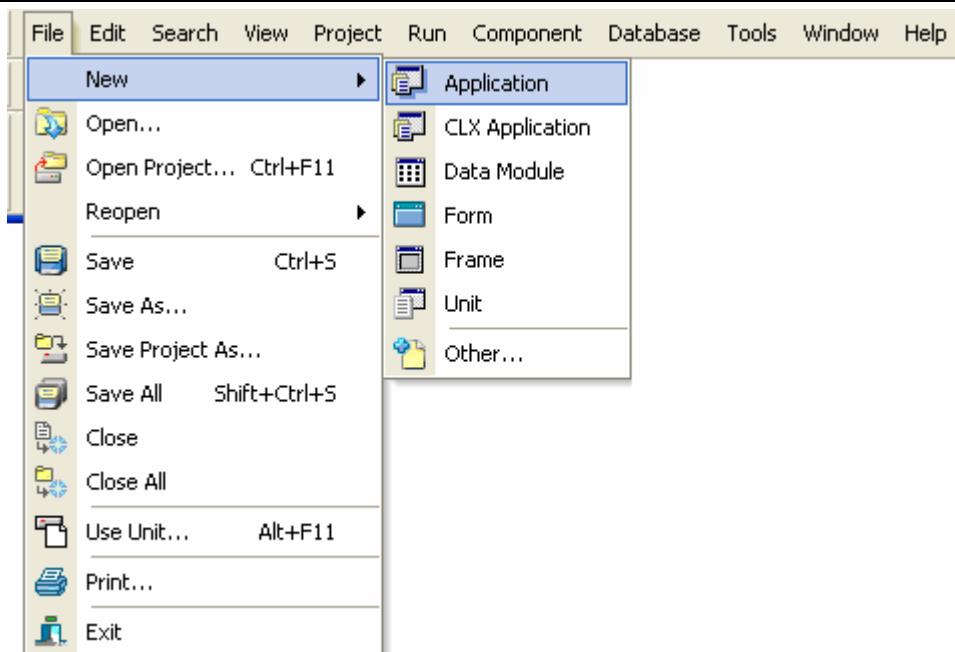
II.1. Cửa sổ chính của Delphi

Cửa sổ chính của Delphi chính là cửa sổ có thành tiêu đề chứa tên dự án (Project name) mà bạn đang phát triển, nó bao gồm thực đơn chính (Main menu), thanh công cụ (Toolbar), và bảng thành phần (Component palette).

II.2. Thanh thực đơn chính và thanh công cụ

II.2.1. Thanh thực đơn chính

Bao gồm các thực đơn thả xuống (Drop-down menu) như: File, Edit, Search, View,... và Help. Trong mỗi menu thả xuống có nhiều chức năng khác nhau như: mở một dự án mới, lưu dự án, biên dịch, gỡ lỗi, chạy chương trình,... mà giáo trình sẽ trình bày chi tiết ở phần sau.

**Hình 2:** Thanh thực đơn chính

II.2.2. Thanh công cụ (Toolbars)

Trong Delphi có nhiều thanh công cụ, như: thanh công cụ chuẩn (Standard), thanh gỡ lỗi (Debug), thanh hiển thị (View),... Mỗi nút (Button) trên thanh công cụ thường là một thao tác hay một lệnh mà khi ta Click vào nó sẽ thi hành, ví dụ như: Biên dịch (Compile), chạy (Run) hoặc kết thúc chạy chương trình (Program Reset), ...

**Hình 3:** Các thanh công cụ

II.3. Bảng chứa các thành phần của Delphi (Component Palette)

Thành phần (Component), hay còn được gọi là điều khiển, chính là đối tượng có sẵn trong Delphi mà bạn có thể thao tác trên nó trong thời điểm thiết kế form. Có 2 loại thành phần, đó là thành phần trực quan (visual component): nhìn thấy khi chạy chương trình, và thành phần không trực quan (nonvisual component): không nhìn thấy khi chạy chương trình. Mỗi thành phần có một số tính chất riêng, và được quản lý thông qua các thuộc tính (Properties), sự kiện (Events) và các phương thức (Methods). Các thuộc tính này giúp bạn có thể quản lý và điều khiển chương trình của bạn. Khi bạn đặt một thành phần lên form, thì nó sẽ xuất hiện trong cửa sổ Object TreeView và Object Inspector – sẽ được trình bày ở phần sau.

Trong bảng chứa các thành phần có nhiều thẻ (tab) khác nhau, như thẻ Standard, Addition, Win32, System, Data Access, ADO, Internet, Rave, Server,... Trên mỗi thẻ chứa các biểu tượng (icon) đại diện cho các thành phần.

**Hình 4:** Bảng chứa các thành phần trong thẻ Standard

II.4. Cửa sổ thiết kế biểu mẫu (Form Designer) và Cửa sổ soạn thảo mã lệnh (Code Editor)

II.4.1. Cửa sổ thiết kế biểu mẫu

Khi bạn tạo một dự án mới, môi trường phát triển tích hợp IDE của Delphi sẽ tự tạo ra một biểu mẫu mới (Form) để bạn tùy nghi thiết lập các giá trị của thuộc tính dựa trên Properties, và các thủ tục sự kiện dựa vào Events, được xác định trong Object Inspector, cho việc thiết kế chương trình. Trong hầu hết trường hợp, một dự án thường có ít nhất một form. Cùng với các thành phần trên bảng các thành phần, bạn sẽ thiết kế được một giao diện cho chương trình mà bạn đang xây dựng là thân thiện nhất cho người sử dụng.



Hình 5: Form “Giai phuong trinh bac nhat” và các Components

Các giá trị của form và các giá trị của các component đặt trên form được lưu trong tập tin form (form file) và có phần mở rộng **.dfm**, với phần tên được xác định giống như phần tên của đơn vị chương trình form. Tập tin form này chứa đựng các giá trị của form cũng như các giá trị của các component mà ta đặt trên form trong quá trình thiết kế.

II.4.2. Cửa sổ soạn thảo mã lệnh

Mỗi một form trong dự án được quản lý trong một tập tin đơn vị chương trình (Unit file/ Form unit), tên tập tin của form unit này được đặt tên trong qua trình lưu (Save) và có phần mở rộng là **.pas**, nội dung của tập tin form unit này chứa đựng các khai báo thông thường của unit (sẽ đề cập chi tiết phần sau) cũng như các hàm sự kiện tương ứng cho form và các component trên nó.

```

unit unitPtb1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Dialogs, StdCtrls, ExtCtrls;

type
  TfrmPTB1 = class(TForm)
    lbl_a: TLabel;
    lbl_b: TLabel;
    edt_a: TEdit;
    edt_b: TEdit;
    lbl_KQua: TLabel;
    memKQua: TMemo;
    pn1Line: TPanel;
    btnXuLy: TButton;
    btnKetThuc: TButton;
    procedure btnXuLyClick(Sender: TObject);
    procedure btnKetThucClick(Sender: TObject);
  private
  end;

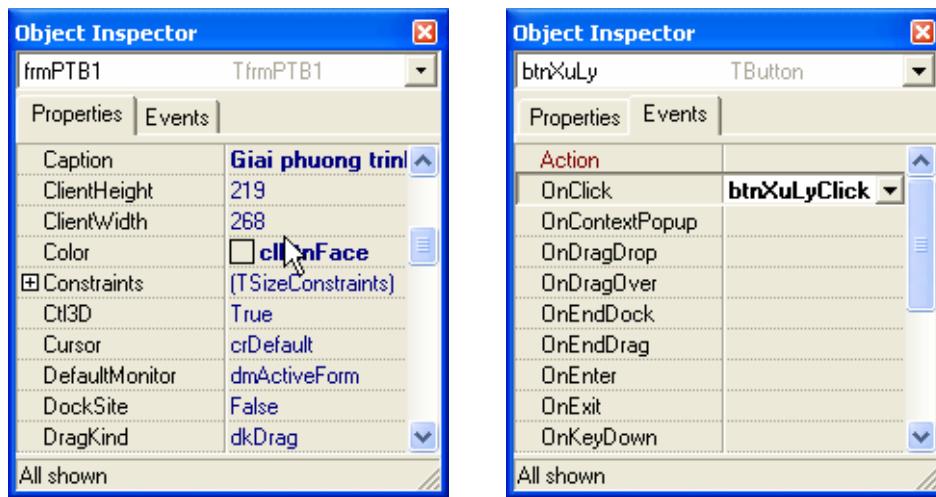
```

Hình 6: Cửa sổ soạn thảo mã lệnh cho form unit untPtb1.Pas

Để chuyển đổi giữa cửa sổ soạn thảo mã lệnh với cửa sổ thiết kế biểu mẫu, ta sử dụng chức năng **View/Toggle Form/Unit** từ main menu hoặc gõ phím chức năng **F12**.

Ví dụ 1: Trong project “Giai phuong trinh bac nhat” ở trên, ta lưu unit file với tên untPtb1.Pas và form file sẽ được tự động đặt theo là untPtb1.dfm

II.5. Cửa sổ thuộc tính và sự kiện của đối tượng (Object Inspector)

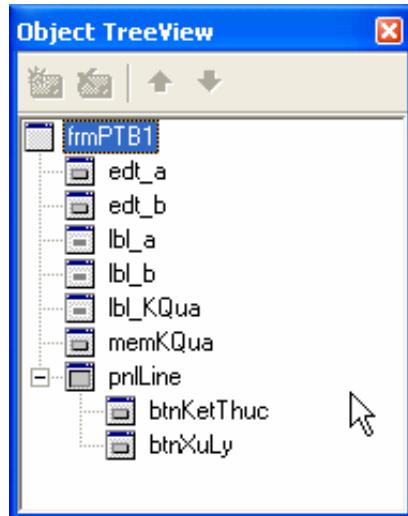
**Hình 7:** thẻ Properties của Form và thẻ Events của Button

Cửa sổ Object Inspector của form hay component đều có 2 thẻ: Properties và Events. Thẻ Properties dùng để xác định các tính chất của đối tượng nói chung (Form, component) hiện hành một cách trực quan, chẳng hạn như Caption, Name, Position, Visible,... Thẻ Events dùng để lập trình sự kiện: xác định những đáp ứng của đối tượng khi nhận tác động

từ chuột (Mouse) hoặc bàn phím (Keyboard) như: nhấp chuột (OnClick), đóng cửa sổ (OnClose), gõ phím Enter (OnEnter),... thông qua các thủ tục sự kiện.

Khi thay đổi các giá trị trong thẻ Properties trong quá trình thiết kế, thì mã lệnh tương ứng của đối tượng sẽ được thay đổi theo trong cửa sổ soạn thảo mã lệnh.

II.6. Cửa sổ liệt kê các đối tượng dạng cây (Object TreeView)



Hình 8: Cửa sổ liệt kê các đối tượng dạng cây trên frmPTB1

Object TreeView liệt kê các thành phần trực quan và không trực quan mà bạn đặt chúng trên form hiện hành. Mỗi thời điểm chỉ có một form duy nhất được liệt kê.

III. Cấu trúc một dự án Delphi

III.1. Tập tin dự án: (Delphi project file)

The screenshot shows the Delphi IDE interface with the project file 'prjPtb1.dpr' open. The left pane shows the project structure under 'Uses': 'Forms' and 'untPtb1'. The right pane displays the Delphi code for the project:

```

program prjPtb1;

uses
  Forms,
  untPtb1 in 'untPtb1.pas' (frmPTB1);

  {$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmPTB1, frmPTB1);
  Application.Run;
end.

```

Hình 9: Tập tin dự án prjPtb1.dpr

Khi bạn bắt đầu mở mới một chương trình ứng dụng bằng thao tác: **File/New/Application**, Delphi 7 sẽ tự động tạo ra một tập tin dự án có tên mặc nhiên ban đầu là Project1. Khi bạn lưu dự án, Delphi sẽ lần lượt nhắc bạn đặt tên cho các form unit và tên dự án. Ở ví dụ giải phuong trình bậc nhất ở trên, form unit có tên là **untPTB1.pas** và tên dự án là **prjPTB1.dpr**. Phần tên của tập tin dự án cũng chính là tên của chương trình, được khai báo sau từ khóa **program**.

Bạn chú ý: tiền tố của unit file là **unt**, còn project file là **prj**. Ví dụ như **untPTB1.pas** và **prjPTB1.dpr**

Để hiển thị tập tin dự án, ta vào chức năng **Project/View Source**

Mã lệnh nguồn (Source code) được Delphi tự động sinh ra trong tập tin dự án bao gồm:

- Tên chương trình, được khai báo bởi từ khóa **program**, đây cũng chính là tên chương trình ứng dụng (Application file).
- Các câu lệnh chỉ dẫn tới các form và các unit được sử dụng trong dự án được khai báo sau từ khóa **uses**.
- Chỉ thị biên dịch {\$R *.res}: sử dụng các tập tin tài nguyên (Resources).
- Khối lệnh thân chương trình chính, bắt đầu bởi từ khóa **begin** và kết thúc là **end**.
 - + **Initialize** : Khởi động chương trình ứng dụng.
 - + **CreateForm** : Tạo form chính (Main form).
 - + **Run** : Chạy chương trình ứng dụng.

Tập tin dự án là tập tin trung tâm của một dự án được triển khai bởi Delphi. Tập tin này chứa đựng các tham khảo (Reference) tới tất cả các file khác trong dự án và liên kết với các form cũng như tập tin đơn vị (Unit file) kèm theo của form. Khi chương trình được thi hành (Run, execute), nó sẽ bắt đầu từ tập tin dự án này.

Tập tin này được tự động sinh ra, và như vậy, bạn không nên thay đổi nội dung của nó ngoại trừ trong những trường hợp cần thiết.

Tập tin dự án được lưu với phần mở rộng **.dpr**

III.2. Các tập tin chứa mã lệnh (Unit file)

Trong mỗi ứng dụng được viết bởi Delphi thường gồm có 1 tập tin dự án (Project file) và nhiều tập tin đơn vị chương trình (Unit file) – xem lại phần "Cửa sổ soạn thảo mã lệnh" ở phần trên. Mỗi khi ta thêm vào mới một form (hoặc Data Module, Frame,...) thì một tập tin unit sẽ được tạo ra. Thông thường một dự án được triển khai trong thực tế sẽ có nhiều tập tin unit này. Nội dung tập tin unit này chứa đựng mã lệnh của chương trình và của các sự kiện điều khiển (Event handles) trên form cũng trên các thành phần (Component) mà chúng được đặt trên nó.

Unit file được lưu với phần mở rộng là **.pas**

III.3. Các tập tin đặc tả biểu mẫu (Form file)

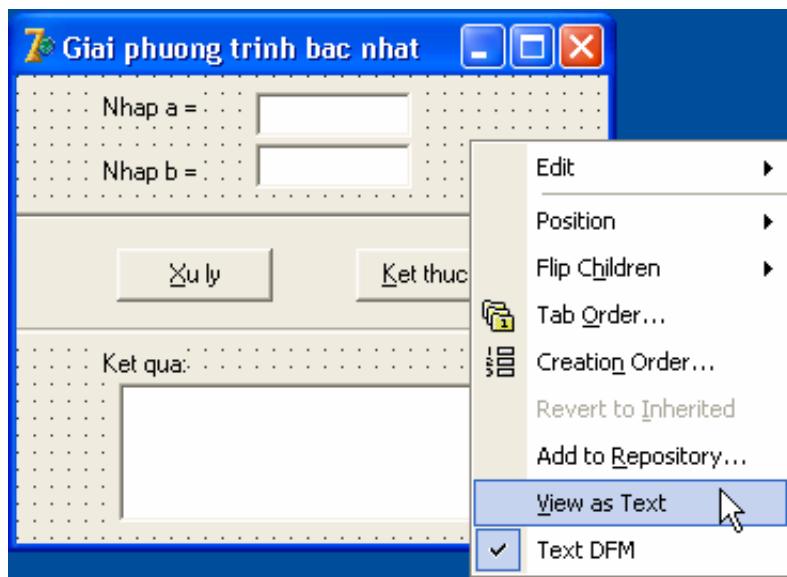
Nội dung của tập tin này xác định một cách chi tiết các tính chất hay thuộc tính (Properties) của form và tất cả các đối tượng (nói chung) lập trình viên đã thiết kế. Tập tin này luôn đi kèm với một tập tin unit file.

Form file được lưu với phần mở rộng là **.dfm**

Để hiện thị nội dung tập tin này, bạn **RClick** trên form rồi chọn **View as Text**.

Ví dụ 2: Xem nội dung tập tin form file có tên untPTB1.dfm như sau:

Từ chế độ thiết kế của form có tên là frmPTB1, bạn RClick trên form rồi chọn View as Text như hình sau:



Hình 10: Chuyển cách hiển thị từ dạng Form sang dạng Text

Nội dung của tập tin untPTB1.dfm ở dạng văn bản sẽ có như hình sau:

```

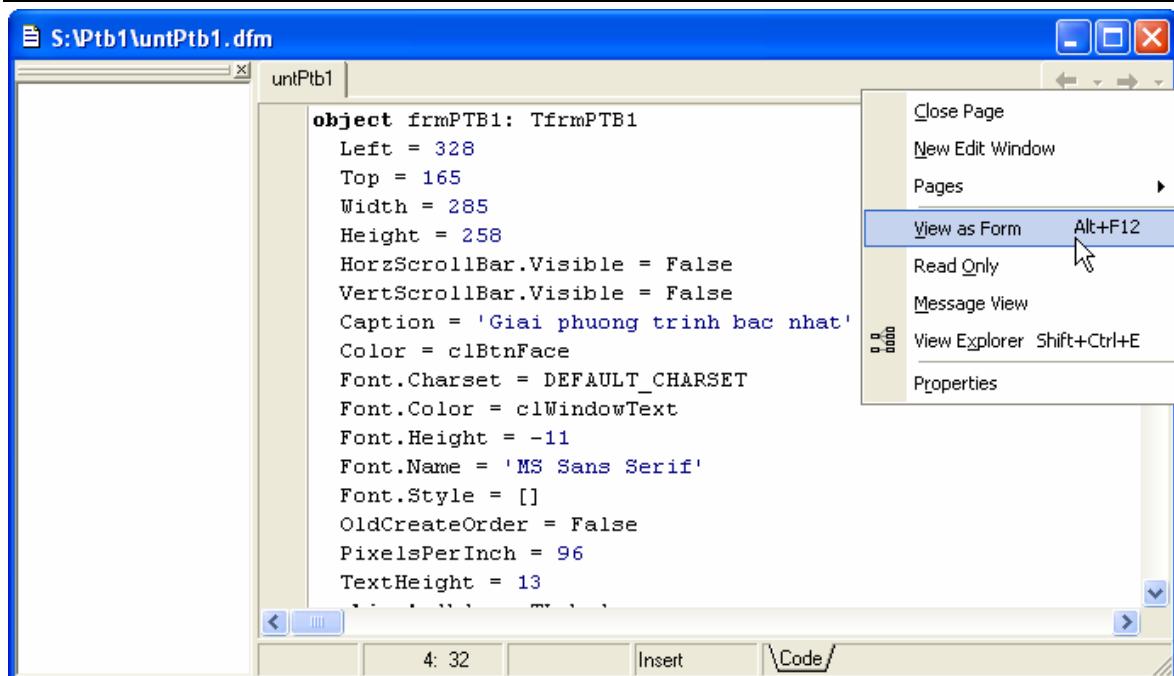
S:\PtB1\untPtB1.dfm
untPtB1

object frmPTB1: TfrmPTB1
  Left = 328
  Top = 165
  Width = 285
  Height = 258
  HorzScrollBar.Visible = False
  VertScrollBar.Visible = False
  Caption = 'Giai phuong trinh bac nhat'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13

```

Hình 11: Hiển thị dạng text của untPTB1.dfm

Để chuyển đổi từ cách hiện thị dạng Text sang dạng Form, ta sử dụng chức năng **RClick** chọn chức năng **View as Form** hay tổ hợp phím **Alt+F12**:



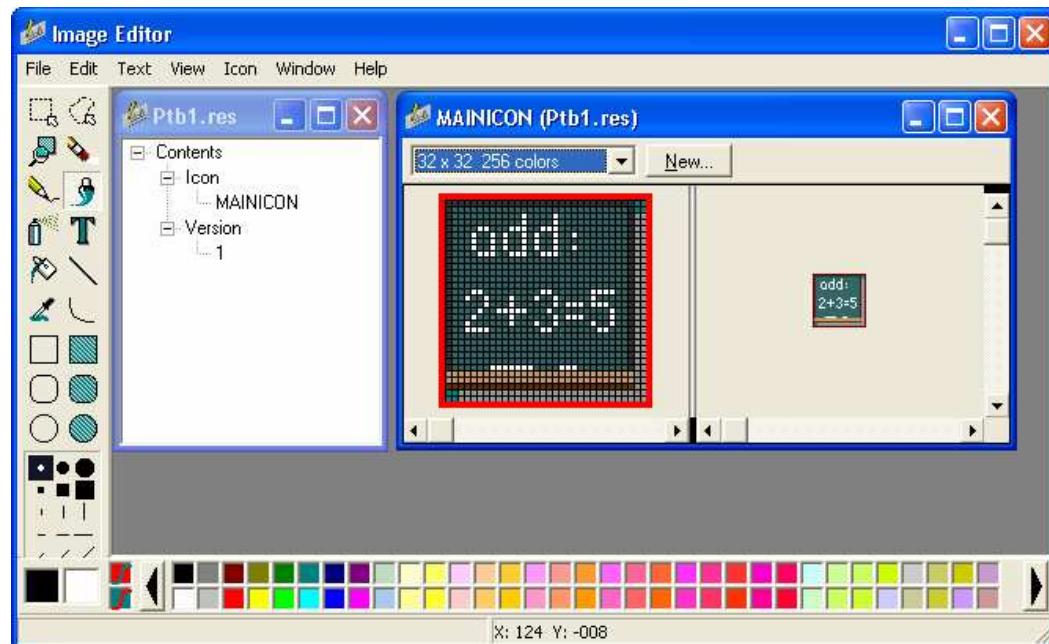
Hình 12: Chuyển cách hiển thị dạng Text sang dạng Form.

III.4. Các tập tin tài nguyên (Windows Resource File)

Tập tin này cũng được tự động sinh ra. Nội dung của nó chứa đựng các thông tin về phiên bản (Version) cũng như biểu tượng của ứng dụng (Application icon) ở dạng mã nhị phân.

Tập tin này có phần mở rộng .res

Ta có thể mở tập tin tài nguyên này ở chế độ thiết kế (Design) bằng chương trình **Image Editor** được kèm theo bộ Delphi.

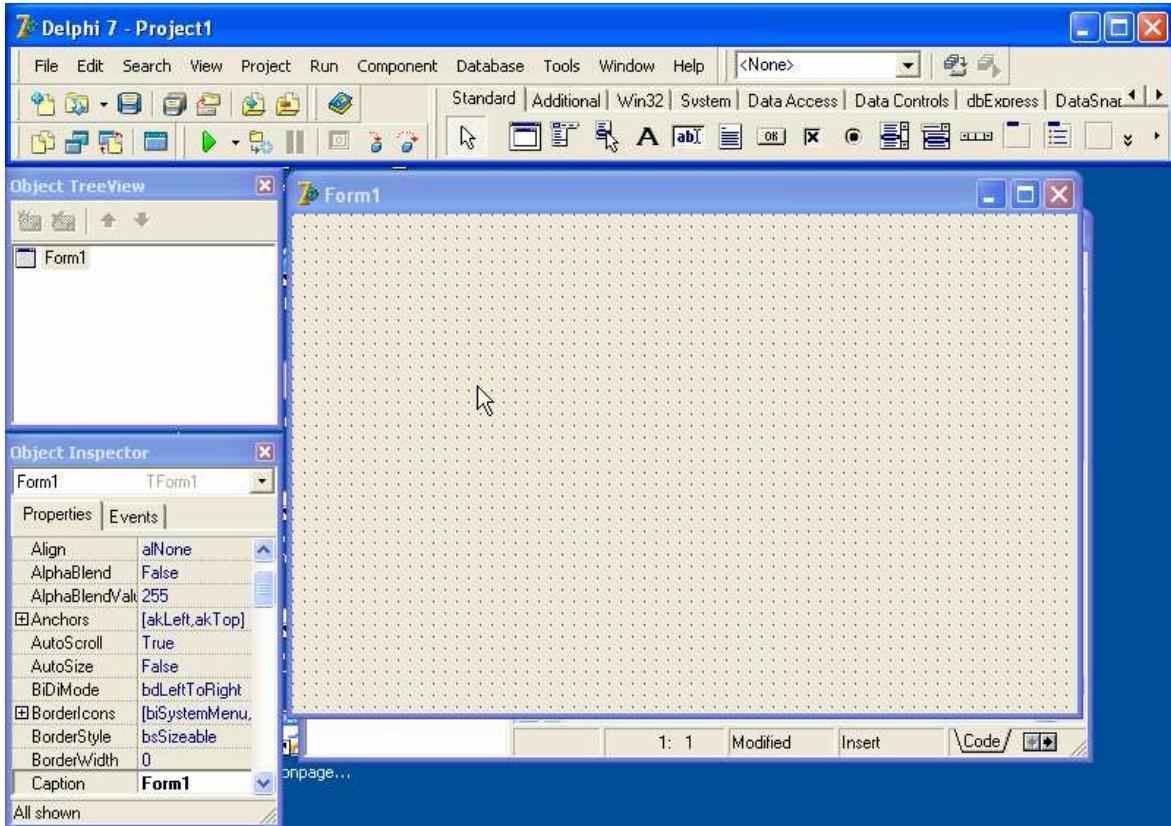


Hình 13: Giao diện chương trình xử lý ảnh – Image Editor

IV. Các thao tác cơ bản trên Delphi

IV.1. Mở một dự án mới

Để tạo một dự án mới, bạn sử dụng lệnh **File/New/Application**. Hình ảnh một dự án mới tạo như hình sau:



Hình 14: Dự án mới tạo

IV.2. Lưu dự án

Một dự án trong Delphi sẽ sinh ra khá nhiều tập tin. Do vậy, khi lưu dự án, bạn cần phải tạo ra 1 thư mục để chứa các tập tin trong dự án đó. Quá trình lưu dự án như sau:

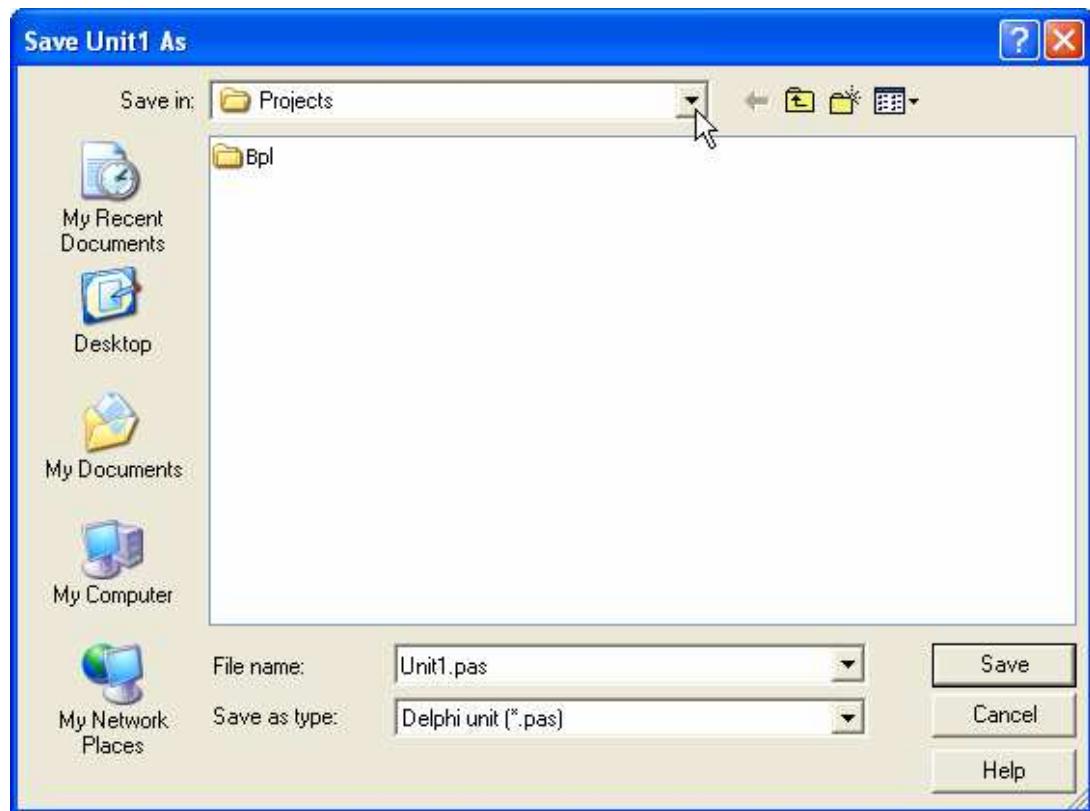
Để lưu một dự án, bạn sử dụng lệnh: **File/Save All** hay tổ hợp phím **Shift+Ctrl+S**. Thông thường bạn phải lưu các form trước (mỗi form là một tập tin unit có phần mở rộng là **.pas**), rồi sau đó mới lưu tập tin dự án (phần mở rộng là **.dpr**).

Ví dụ 3: Lưu dự án hiển thị câu "Hello world!..." chỉ có một form, vào thư mục **S:\HelloWorld** như sau:



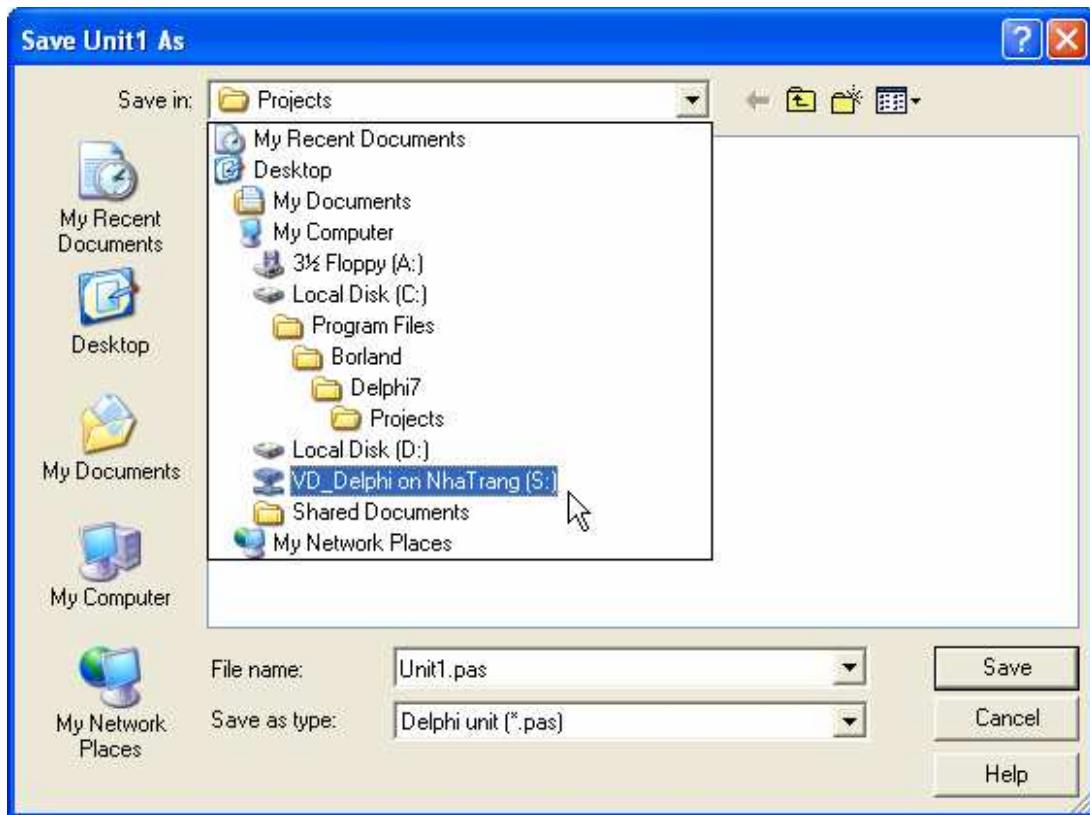
Hình 15: Cửa sổ thiết kế Form

- Từ cửa sổ thiết kế, chọn lệnh **File/Save All**, Delphi sẽ xuất hiện ra cửa sổ "Save Unit_i File" (với i =1..n) như sau:



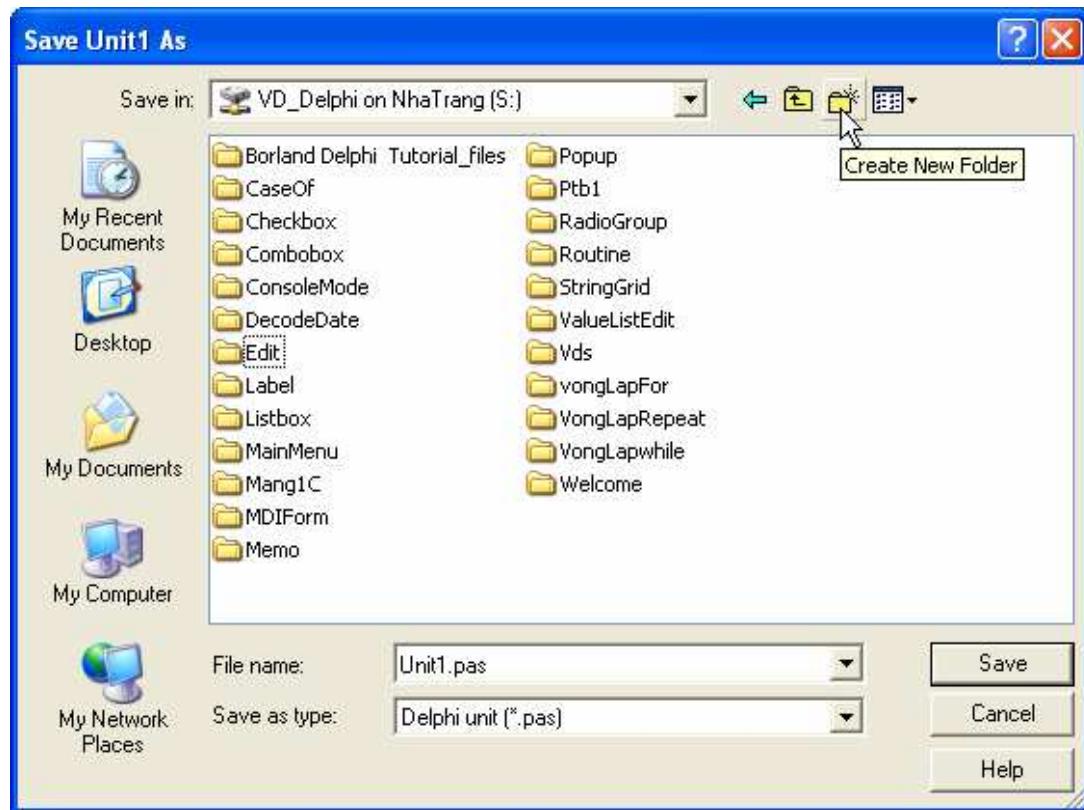
Hình 16: Hộp thoại Save Unit_i As

Click chọn hộp liệt kê thả "Save in" để chọn ổ đĩa S: như hình sau:



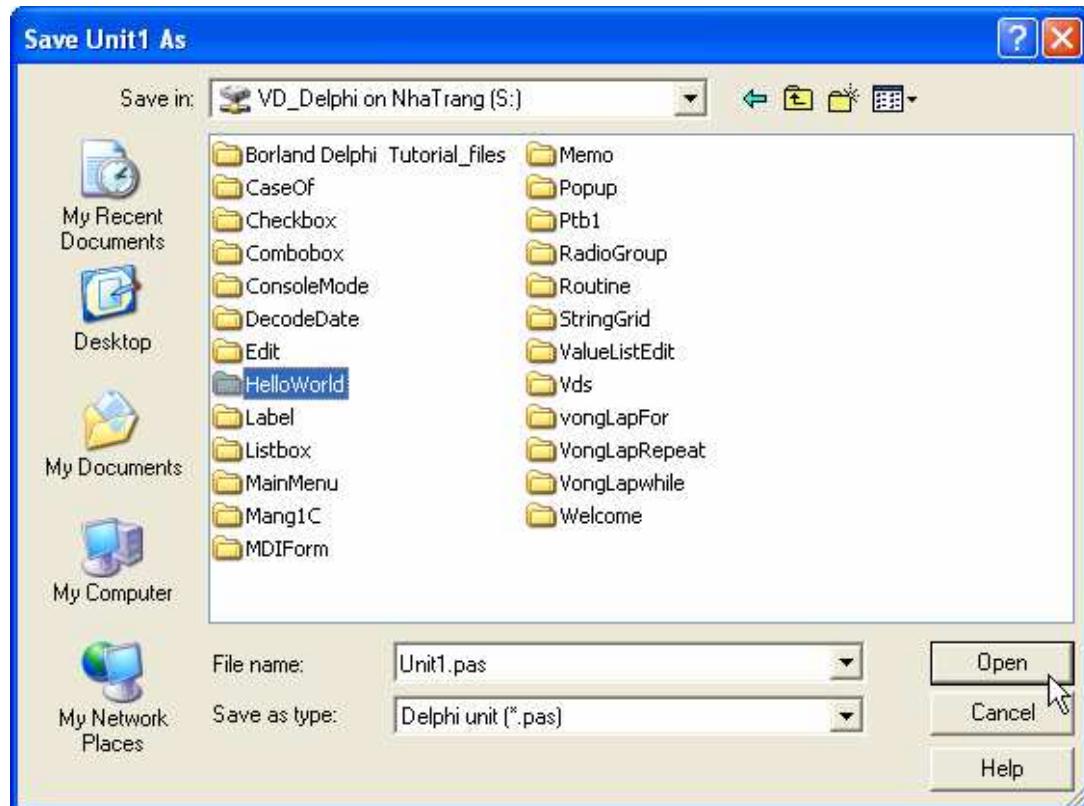
Hình 17: Hộp thoại Save Unit_i As: Chọn ổ đĩa

Tiếp đến, Click chọn biểu tượng "Create New Folder" để tạo thư mục mới chứa các tập tin của dự án mà bạn muốn lưu (thư mục **HelloWorld**).



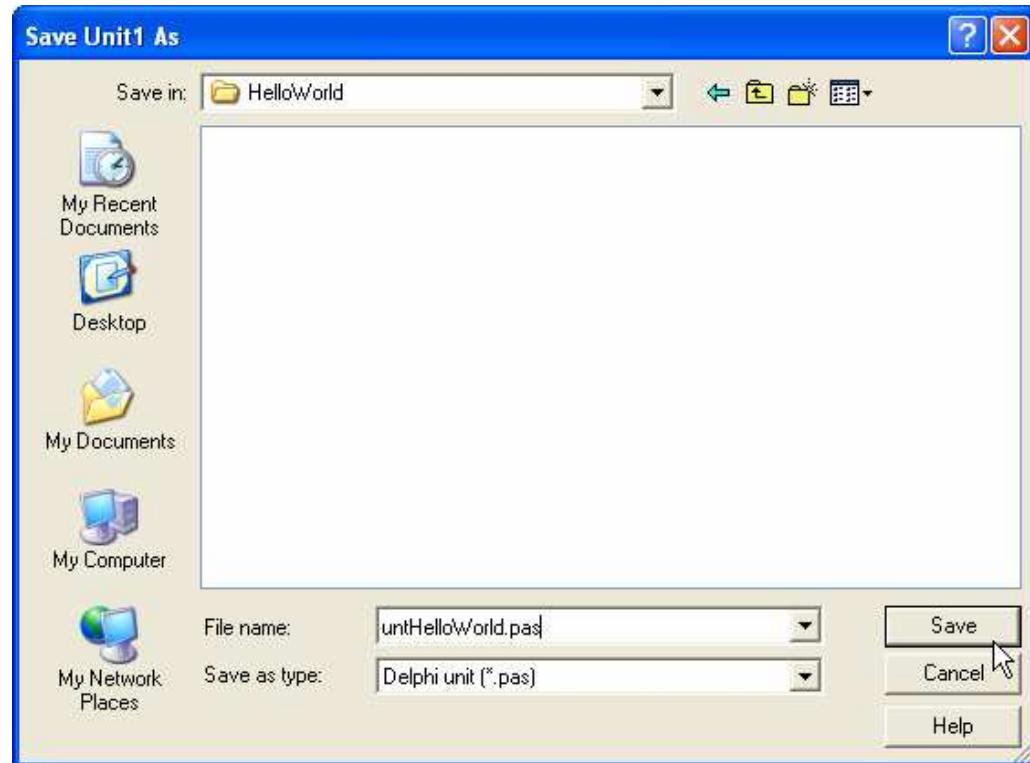
Hình 18: Hộp thoại Save Unit_i As: **Tạo thư mục**

Sau khi tạo thư mục xong, bạn Click nút lệnh **Open** để mở thư mục **HelloWorld** ra.



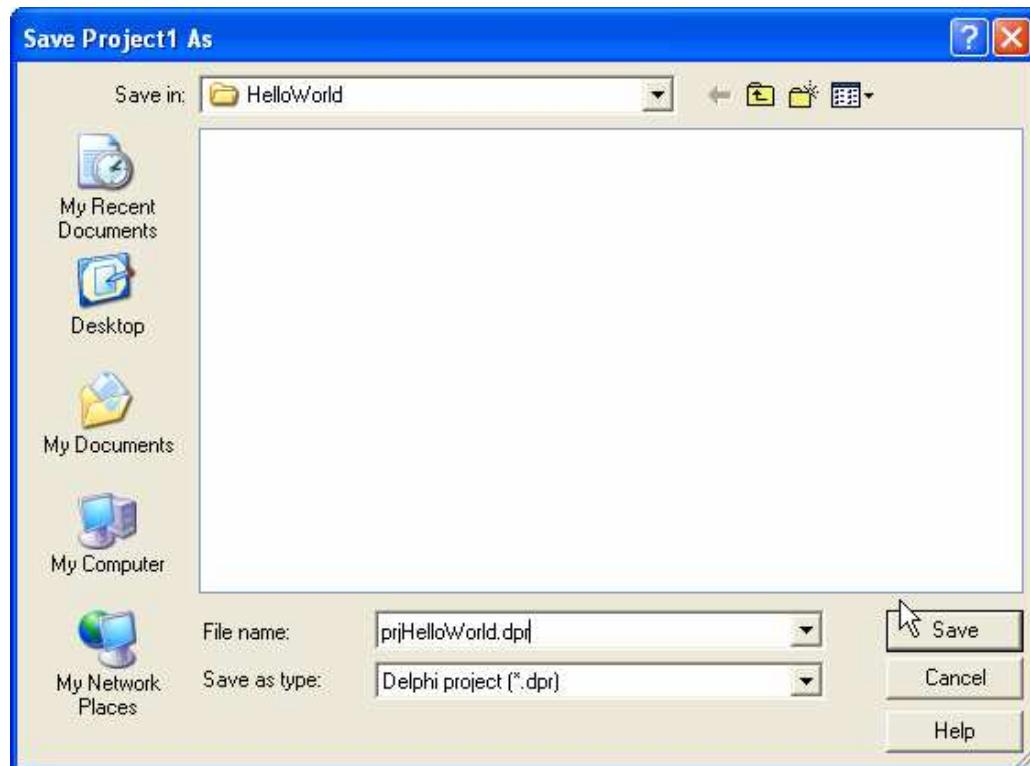
Hình 19: Hộp thoại Save Unit_i As: **Mở thư mục**

Tiếp đến, bạn gõ tên tập tin unit mà bạn muốn lưu vào trong hộp liệt kê thả "File name", ở đây là tên **untHelloWorld.pas**, rồi Click Save.



Hình 20: Hộp thoại Save Unit_i As: Nhập vào tên tập tin unit

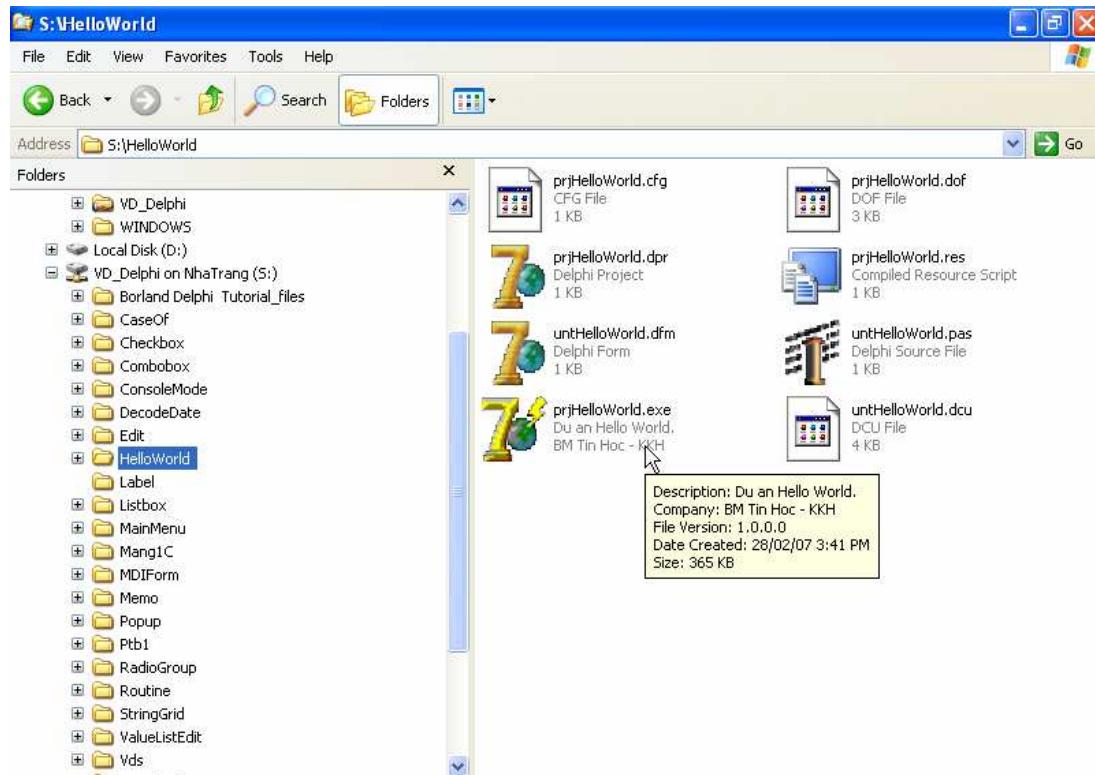
Đến đây, Delphi sẽ tiếp tục xuất hiện hộp thoại "Save Project; As" để bạn lưu tập tin dự án. Bạn gõ vào tên tập tin dự án vào trong hộp liệt kê thả "File name", ở ví dụ này là tên **prjHelloWorld.dpr**, và Click Save.



Hình 21: Hộp thoại Save Project_i As: Nhập tên tập tin dự án

Dự án đã được lưu và đặt tên xong. Sau việc này, nếu bạn có chỉnh sửa chương trình, để lưu lại những sự thay đổi này, bạn chỉ cần ra lệnh **File/Save All** để Delphi tự động lưu toàn bộ lại dự án.

Hình dưới đây cho bạn thấy số lượng các tập tin trong một dự án Hello World đã được lưu, thông qua cửa sổ Windows Explorer như sau:



Hình 22: Các tập tin của dự án Hello Wolrd trong thư mục S:\HelloWorld

IV.3. Lưu Form (tập tin unit) với tên khác

Bạn chọn lệnh **File/Save As** để lưu.

IV.4. Lưu dự án với tên khác

Bạn chọn lệnh **File/Save Project As** để lưu dự án.

IV.5. Đóng dự án

Bạn chọn lệnh **File/Close All** để đóng lại toàn bộ dự án.

IV.6. Thoát khỏi Delphi

Sau khi lưu lại toàn bộ dự án. Để thoát khỏi Delphi, bạn sử dụng lệnh: **File/Exit**

CHƯƠNG 3: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ DELPHI (OBJECT PASCAL)

Trong chương này và chương 4 tiếp theo sau, để giúp cho việc làm quen với các khái niệm cơ bản được dễ dàng. Giáo trình sẽ xây dựng các ví dụ theo dạng chương trình ở chế độ văn bản (Text mode) hay không hỗ trợ đồ họa (Console Application). Để tạo một project mới theo dạng Console Application, bạn sử dụng chức năng **File/New/Other/Console Application**.

I. Bộ chữ viết

- Bộ 26 chữ Latin:
 - o Chữ in : A, B, C, ..., X, Y, Z
 - o Chữ thường : a, b, c, ..., x, y, z
- Bộ chữ số thập phân : 0, 1, 2, 3, ..., 8, 9
- Ký tự gạch nối : _
- Các ký hiệu toán học : +, -, *, /, =, <, >, (,), [,]

II. Từ khóa

Là các từ riêng của Object Pascal, có ngữ nghĩa đã được xác định, không được dùng nó vào các việc khác hoặc đặt tên mới trùng với các từ khóa.

Từ khóa chung: program, begin, end, procedure, function

- Từ khóa để khai báo: const, var, type, array, string, record, set, file, label, class, ...
- Từ khóa của lệnh lựa chọn: if ... then ... else, case ... of
- Từ khóa của lệnh lặp: for... to... do, for... downto... do, while... do, repeat... until
- Từ khóa điều khiển: with, goto, exit, halt
- Từ khóa toán tử: and, or, not, in, div, mod

Dưới đây là danh sách một số từ khóa thông dụng:

and	for	record
array	function	repeat
begin	goto	set
case	if	to
class	implementation	try
const	interface	type
div	label	unit
do	mod	until
downto	object	uses

else	of	var
end	procedure	while
finalization	program	with
finally	property	

Bảng 1: Các từ khoá

III. Tên danh hiệu tự đặt

Trong Delphi để đặt tên cho các biến, hằng, kiểu, chương trình con ta dùng các danh hiệu (Identifier). Quy tắc đặt tên của danh hiệu trong Delphi là:

- Chiều dài tối đa: **256** ký tự.
- Bắt đầu bằng **chữ cái** hoặc dấu _ (gạch dưới); tiếp theo có thể kết hợp số và các ký tự khác.
- Không được chứa khoảng trống.
- Không trùng với các từ khóa.
- Không phân biệt chữ hoa và chữ thường.
- Các định danh lồng nhau được dùng dấu . (dấu chấm) để truy xuất.

Ví dụ 1:

- Danh hiệu tự đặt hợp lệ: x; s1; delta; pt_bac_2;
- Danh hiệu tự đặt không hợp lệ: **1S** (bắt đầu bằng số), **Del ta** (tên có khoảng trắng),...

IV. Hằng

IV.1. Khái niệm

Hằng là một đại lượng có giá trị không đổi trong quá trình chạy chương trình. Ta dùng tên hằng để chương trình được rõ ràng và dễ sửa đổi.

Cú pháp khai báo:

Const

Tên_hằng = biểu_thức_xác định_giá_trị_của_hằng;

IV.2. Hằng trị

Trong trường hợp này thì **biểu_thức_xác định_giá_trị_của_hằng** là một trị xác định. Giá trị của biểu thức này sẽ được định trị một lần và không thay đổi.

Ví dụ 2:

Const MaxSize = 100;

Const x = Abs(-100)+Length('Chao ban'); // hằng x =108

Const St = 'Borland'+ ' Delphi'; // hằng St = 'Borland Delphi'

Chú ý: Biểu thức hằng ở đây có thể là số, chuỗi ký tự, các hằng true, false, nil,... kết hợp với các toán tử thích hợp. Biểu thức hằng không thể chứa các lời gọi hàm, biến hay con trỏ, ngoại trừ một số hàm đơn giản như: Abs, Chr, Length, Succ, Odd,...

IV.3. Hằng định kiểu:

Có thể chứa các giá trị kiểu mảng, bản ghi, kiểu thủ tục và kiểu con trỏ.

Cú pháp khai báo hằng định kiểu:

Const

Tên_hằng: kiểu_dữ_liệu = giá_trị_hằng;

Ví dụ 3:

Const MaxSize : Integer =100;

Trong ví dụ trên chúng ta khai báo một hằng có tên là MaxSize, có kiểu Integer và mang giá trị bằng 100. Chúng ta thấy, cách khai báo hằng kiểu này rất giống khai báo và khởi tạo biến (chúng ta sẽ xem trong phần biến). Nếu người lập trình bắt chỉ thị biên dịch {\$J+} thì hằng được xem như biến; tức là **chúng ta có thể thay đổi giá trị của hằng** lúc chương trình thực thi.

Hằng mảng, hằng kiểu bản ghi, hằng kiểu thủ tục chúng ta sẽ tìm hiểu chi tiết trong các phần sau.

V. Kiểu (Type)

V.1. Định nghĩa

Ngoài các kiểu đã định sẵn, Delphi còn cho phép ta định nghĩa các kiểu dữ liệu khác từ các kiểu căn bản theo qui tắc xây dựng của Delphi.

V.2. Cách khai báo

Type

Tên_kiểu = Mô_tả_xây_dựng_kiểu;

Ví dụ 4: Định nghĩa các kiểu dữ liệu người dùng (kiểu dữ liệu mới) và khai báo biến theo kiểu vừa định nghĩa

type //Khai báo kiểu người dùng

SoNguyen = integer;

Diem = single;

Tuoi = 1 .. 100;

Color = (Red, Blue, Green);

Thu = (ChuNhat, Hai, Ba, Tu, Nam, Sau, Bay);

Và khi đã khai báo kiểu thì ta có quyền sử dụng kiểu đã định nghĩa để khai báo biến như sau:

var //Khai báo biến theo kiểu người dùng định nghĩa

i, j: SoNguyen;

dtb: Diem;

t: Tuoi;

mau: Color;
ngay_hoc: Thu;

VỊ. Biến

Biến là một cấu trúc ghi nhớ có tên (đó là tên biến hay danh hiệu của biến). Biến ghi nhớ một dữ liệu nào đó gọi là giá trị (value) của biến. Giá trị của biến có thể được biến đổi trong thời gian sử dụng biến.

Sự truy xuất của biến nghĩa là đọc giá trị hay thay đổi giá trị của biến được thực hiện thông qua tên biến.

Cú pháp khai báo biến:

Var

tên_biến [hoặc danh_sách_tên_biến] : Kiểu_biến;

Nếu chúng ta khai báo một danh sách các biến có cùng kiểu thì mỗi tên biến cách nhau một dấu phẩy. Kiểu biến là một kiểu dữ liệu chuẩn trong Delphi hoặc kiểu được định nghĩa bởi người dùng.

Ví dụ 5:

```
var a      : extended; // khai báo a là một biến kiểu số thực  
b, c    : integer;   // khai báo 2 biến b, c có kiểu số nguyên  
hten   : shortstring;  
m_Ok: boolean;    // khai báo m_Ok là một biến kiểu logic  
Chon : char;       // khai báo Chon là một biến kiểu ký tự
```

Cần khai báo các biến trước khi sử dụng chúng trong chương trình. Khai báo một biến là khai báo sự tồn tại của biến đó và cho biết nó thuộc kiểu gì.

VII. Biểu thức

VII.1. Định nghĩa

Một biểu thức là một công thức tính toán bao gồm các phép toán (toán tử), hằng, biến, hàm và các dấu ngoặc.

Ví dụ 6:

```
5 + u * sqrt(t) / sin(pi/2);  
(a=5) and (b<>7) or (c >= 10);  
max(x,y); // gọi hàm
```

Trong đó: a, b, c, x, y, u, t là các biến đã được khai báo.

VII.2. Thứ tự ưu tiên

❖ Toán tử:

Trong Delphi gồm có các toán tử được phân chia như sau:

- Toán tử số học : +, -, *, /, div, mod

- Toán tử logic : not, and, or, xor
- Toán tử quan hệ : =, >, <, <>, <=, >=
- Toán tử bitwise : not, and, or, xor, shl, shr
- Toán tử làm việc trên tập hợp : +, -, *, /, <=, >=, <>*, in
- Toán tử thao tác trên con trỏ : +, -, @, ^, =, <>
- Toán tử xử lý llop đối tượng : as, is

Khi tính giá trị của một biểu thức, ngôn ngữ Delphi qui ước thứ tự ưu tiên của các phép toán từ cao đến thấp như sau:

Phép toán	Thứ tự ưu tiên
@, not	ưu tiên 1
*, /, div, mod, and, shl, shr, as	ưu tiên 2
+, -, or, xor	ưu tiên 3
=, <>, <, >, <=, >=, in, is	ưu tiên 4

Bảng 2: Thứ tự ưu tiên toán tử

❖ Qui ước tính thứ tự ưu tiên:

Khi tính một biểu thức có 3 qui tắc về thứ tự ưu tiên như sau:

- Qui tắc 1. Các phép toán nào có ưu tiên cao hơn sẽ được tính trước.
- Qui tắc 2. Trong các phép toán có cùng thứ tự ưu tiên thì sự tính toán sẽ được thực hiện từ trái sang phải.
- Qui tắc 3. Phần trong ngoặc từ trong ra ngoài được tính toán để trở thành một giá trị đơn.

VIII. Chuyển kiểu (Typecast)

Chuyển kiểu hay còn gọi là ép kiểu là một kỹ thuật được dùng để chuyển giá trị của một biểu thức sang một kiểu dữ liệu khác.

Cú pháp chuyển đổi kiểu dữ liệu: **Tên_Kiểu(biểu_thức);**

Với biểu_thức là một hằng, thì ta có cách chuyển kiểu theo giá trị (value typecasts), còn với biểu_thức là một biến, thì ta có cách chuyển kiểu theo biến (variable typecasts).

Ví dụ 7: Chuyển kiểu theo trị:

```
t := integer('A');           // chuyển ký tự A sang số, t = 65
v := char(65);               // chuyển số sang ký tự, v = 'A'
b := boolean(0);             // chuyển số sang kiểu Boolean, b = False
```

Ví dụ 8: Chuyển kiểu theo biến:

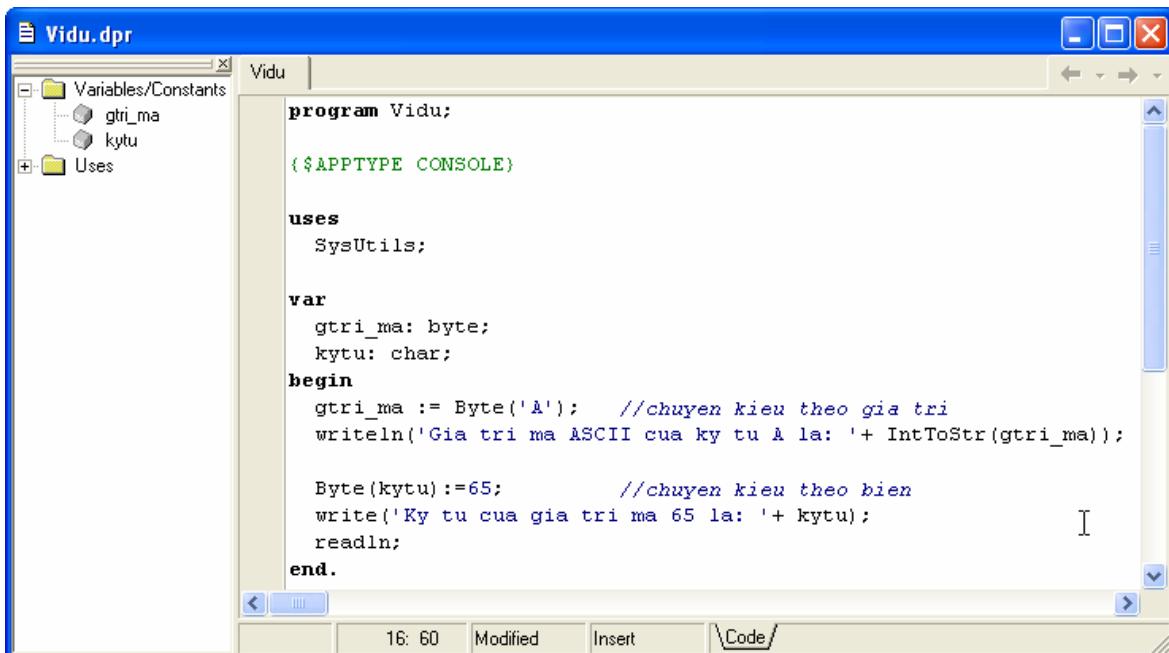
var MyChar: char;

```
Shortint(MyChar) := 122;      // Gán ký tự 'z' vào biến MyChar
```

Chương 3: Các thành phần cơ bản của ngôn ngữ Delphi (Object Pascal).

Ví dụ 9: Viết chương trình minh họa 2 cách chuyển kiểu theo trị và theo biến:

Từ **File/New/Other**, DClick vào Console Application để mở ra 1 dự án mới và nhập các dòng lệnh để hoàn chỉnh như hình sau:



```
program Vidu;
{$APPTYPE CONSOLE}

uses
  SysUtils;

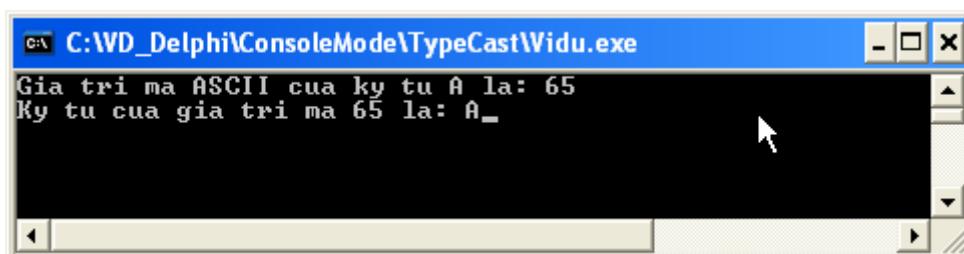
var
  gtri_ma: byte;
  kytu: char;
begin
  gtri_ma := Byte('A'); //chuyen kieu theo gia tri
  writeln('Gia tri ma ASCII cua ky tu A la: ' + IntToStr(gtri_ma));

  Byte(kytu) := 65; //chuyen kieu theo bien
  write('Ky tu cua gia tri ma 65 la: ' + kytu);
  readln;
end.
```

Hình 1: Chương trình chuyển kiểu ở Console mode

Nhấn tổ hợp phím Ctrl+F9 để biên dịch lỗi.

Nhấn phím chức năng F9 để chạy chương trình, và ta có kết quả ở chế độ Console như sau:



```
Gia tri ma ASCII cua ky tu A la: 65
Ky tu cua gia tri ma 65 la: A
```

Hình 2: Kết quả chạy chương trình.

IX. Lời chú thích và các chỉ dẫn biên dịch

Trong Delphi có 3 cú pháp để ghi chú thích như sau:

{ chú thích ghi trong cặp dấu ngoặc mòc }

(* chú thích được bao bởi cặp dấu ngoặc và dấu sao*)

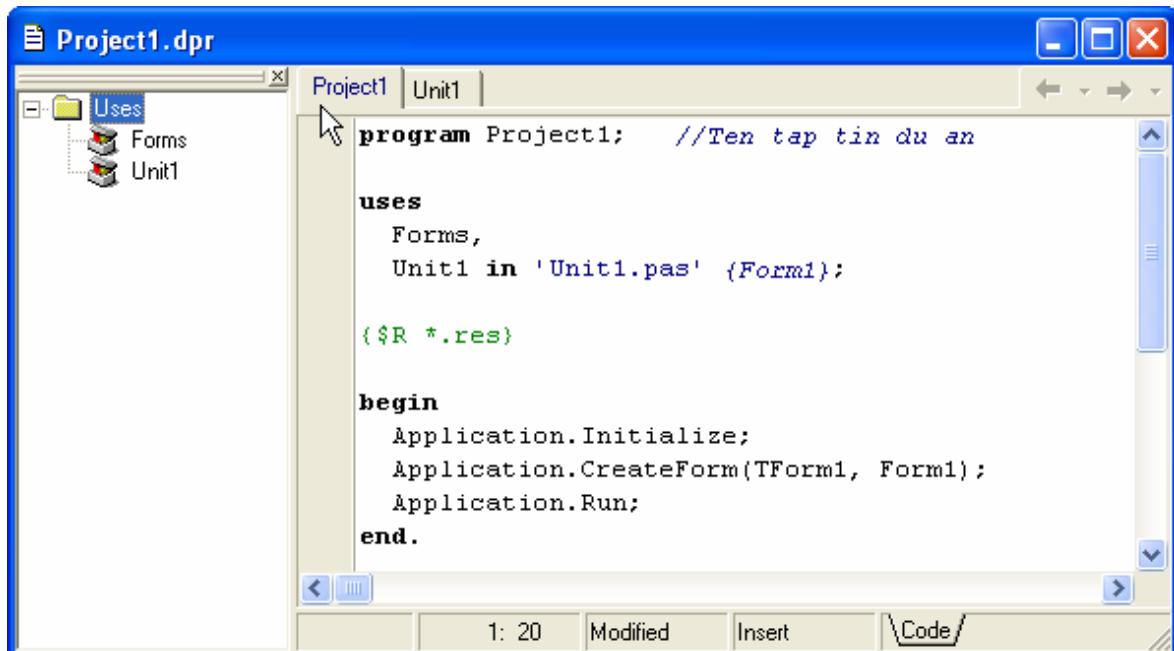
// chú thích trên một dòng

Hai kiểu chú thích khác nhau có thể thực hiện chú thích lồng nhau nhưng cùng kiểu thì không được phép lồng nhau.

X. Cấu trúc một dự án ở chế độ Form (Form Application)

Cấu trúc của một dự án (project) trong Delphi gồm 2 phần chính:

Phần 1: Là tập tin dự án (Project file)



```

Project1.dpr
Project1 | Unit1
program Project1; //Ten tap tin du an

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

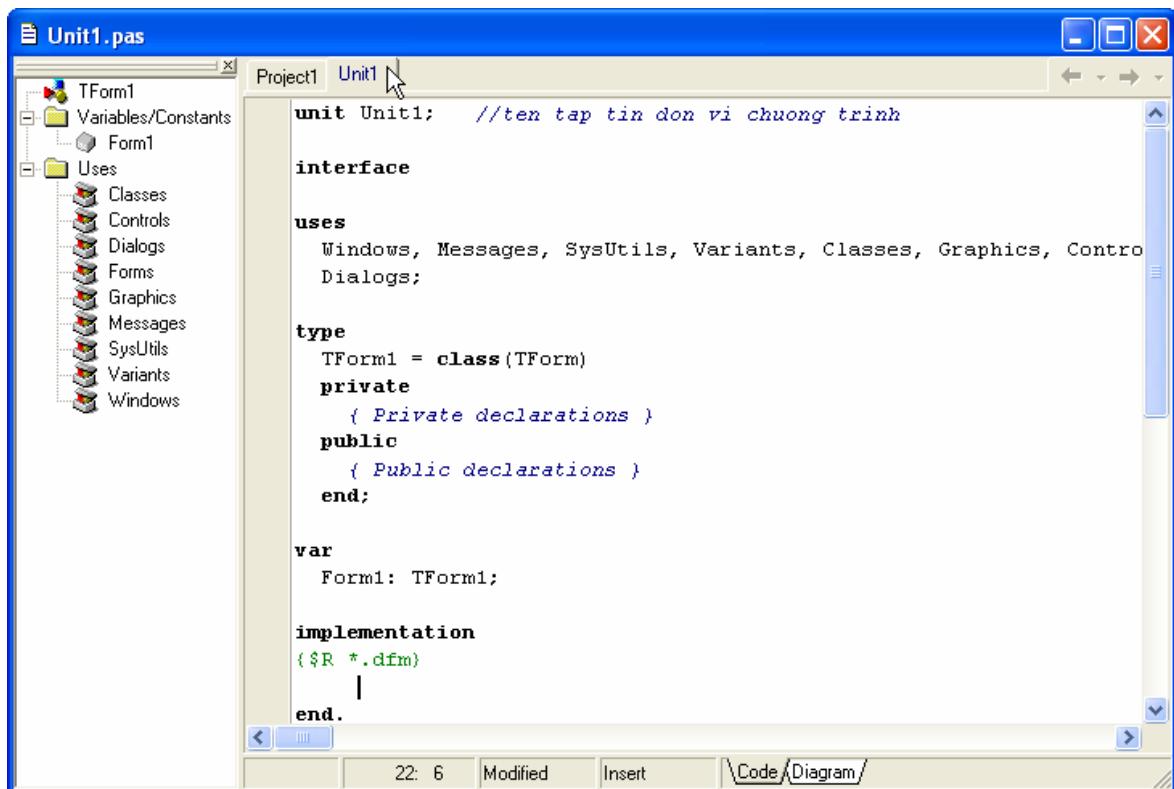
{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.

```

Hình 3: Project file.

Phần 2: Là (các) tập tin đơn vị chương trình (Unit file)



```

Unit1.pas
Project1 | Unit1
unit Unit1; //ten tap tin don vi chuong trinh

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.dfm}
end.

```

Hình 4: Kết quả chạy chương trình.

CHƯƠNG 4: CÁC KIỂU DỮ LIỆU SƠ CẤP CHUẨN, LỆNH ĐƠN

I. Các kiểu dữ liệu sơ cấp (Simple type)

I.1. Kiểu số nguyên (Integer)

Kiểu số nguyên biểu diễn một tập hợp con các số. Delphi phân làm hai loại số nguyên được định nghĩa bởi các từ khóa như sau:

Tùy khóa	Phạm vi	Kích thước
Integer	-2147483648..2147483647	số nguyên có dấu 32 bit
Cardinal	0..4294967295	số nguyên dương 32 bit

Bảng 1: Kiểu nguyên

Trong đó, kiểu Integer còn được chia ra thành một số tập con được định nghĩa bởi các từ khóa sau:

Tùy khóa	Phạm vi	Kích thước
Shortint	-128..127	Số nguyên có dấu 8 bit
Smallint	-32768..32767	Số nguyên có dấu 16 bit
Longint	-2147483648..2147483647	Số nguyên có dấu 32 bit
Int64	-2^63..2^63-1	Số nguyên có dấu 64 bit
Byte	0..255	Số nguyên dương 8 bit
Word	0..65535	Số nguyên dương 16 bit
Longword	0..4294967295	Số nguyên dương 32 bit

Bảng 2: Các kiểu nguyên

I.2. Kiểu ký tự (Char)

Kiểu ký tự cơ bản bao gồm AnsiChar và WideChar. AnsiChar là kiểu ký tự thông thường, chiếm kích thước một byte (tương tự bộ ký tự ASCII mà giáo trình sẽ đề cập ở phần tiếp theo). WideChar là kiểu ký tự hai byte, là kiểu ký tự dùng để biểu diễn bộ mã Unicode cho tập hợp ký tự.

Tuy nhiên, các bạn vẫn có thể khai báo sử dụng kiểu ký tự với từ khóa Char. Và trên thực tế thì kiểu Char tương đương với kiểu AnsiChar. Ngoài Char và AnsiChar, Delphi còn hỗ trợ tập ký tự một byte thông qua các kiểu: PChar, PAnsiChar và AnsiString. Tương tự thì Delphi cũng hỗ trợ tập ký tự Unicode (ngoài kiểu WideChar) thông qua các kiểu: PWideChar, WideString.

Một ký tự được viết trong cặp dấu nháy đơn (""). Để tiện trao đổi thông tin cần phải sắp xếp, đánh số các ký tự, mỗi cách sắp xếp như vậy gọi là bảng mã. Bảng mã thông dụng hiện nay là bảng mã ASCII (American Standard Code for Information Interchange). Hệ mã ASCII dùng nhóm 7 bit hoặc 8 bit để biểu diễn tối đa 128 hoặc 256 ký tự khác nhau và mã hóa theo ký tự liên tục theo cơ số 16.

Hệ mã **ASCII 7 bit**, mã hoá 128 ký tự liên tục như sau:

0	:	NUL (ký tự rỗng)
1 – 31	:	31 ký tự điều khiển
32 – 47	:	các ký tự khoảng trắng SP (space) ! “ # \$ % & ‘ () * + , - . /
48 - 57	:	ký số từ 0 đến 9
58 - 64	:	các dấu : ; < = > ? @
65 - 90	:	các chữ in hoa từ A đến Z
91 – 96	:	các dấu [\] _ `
97 – 122	:	các chữ thường từ a đến z
123 – 127	:	các dấu { } ~ DEL (xóa)

Hệ mã **ASCII 8 bit** (ASCII mở rộng) có thêm 128 ký tự khác ngoài các ký tự nêu trên, gồm các chữ cái có dấu, các hình vẽ, các đường kẻ khung đơn và khung đôi và một số ký hiệu đặc biệt.

Để thực hiện các phép toán số học và so sánh, ta dựa vào giá trị số thứ tự mã ASCII của từng ký tự, chẳng hạn: 'A' < 'a' vì số thứ tự mã ASCII tương ứng là 65 và 97.

Hàm Chr(n): Trả về ký tự tương ứng với số thứ tự mã ASCII n.

Hàm Ord(ch): Trả về số thứ tự của ký tự ch trong bảng mã ASCII.

Ví dụ 1: Với khai báo biến **var** ch: char; stt:byte; thì các câu lệnh sau sẽ có giá trị như sau:

```
ch := chr(65);           // ch = 'A'  
stt := ord('A');        // stt = 65
```

BẢNG MÃ ASCII với 128 ký tự chuẩn

Hex	0	1	2	3	4	5	6	7
0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
2	STX 2	DC2 18	“ 34	2 50	B 66	R 82	b 98	r 114
3	♥ 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
4	♦ 4	DC4 20	\$ 36	4 52	D 68	T 84	d 100	t 116
5	♣ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
6	♠ 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
7	BEL 7	ETB 23	‘ 39	7 55	G 71	W 87	g 103	w 119
8	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
9	HT 9	EM 25) 41	9 57	I 73	Y 89	i 105	y 121
A	LF 10	SUB 26	*	:	J 58	Z 74	j 90	z 106
B	VT 11	ESC 27	+	;	K 59	[75	k 91	{ 107
C	FF 12	FS 28	,	<	L 60	\ 76	l 92	 108
D	CR 13	GS 29	-	=	M 61] 77	m 93	}
E	SO 14	RS 30	.	>	N 62	^ 78	n 94	~ 110
F	SI 15	US 31	/	?	O 63	_ 79	o 95	DEL 111
								127

Hình 1: Bảng mã ASCII chuẩn

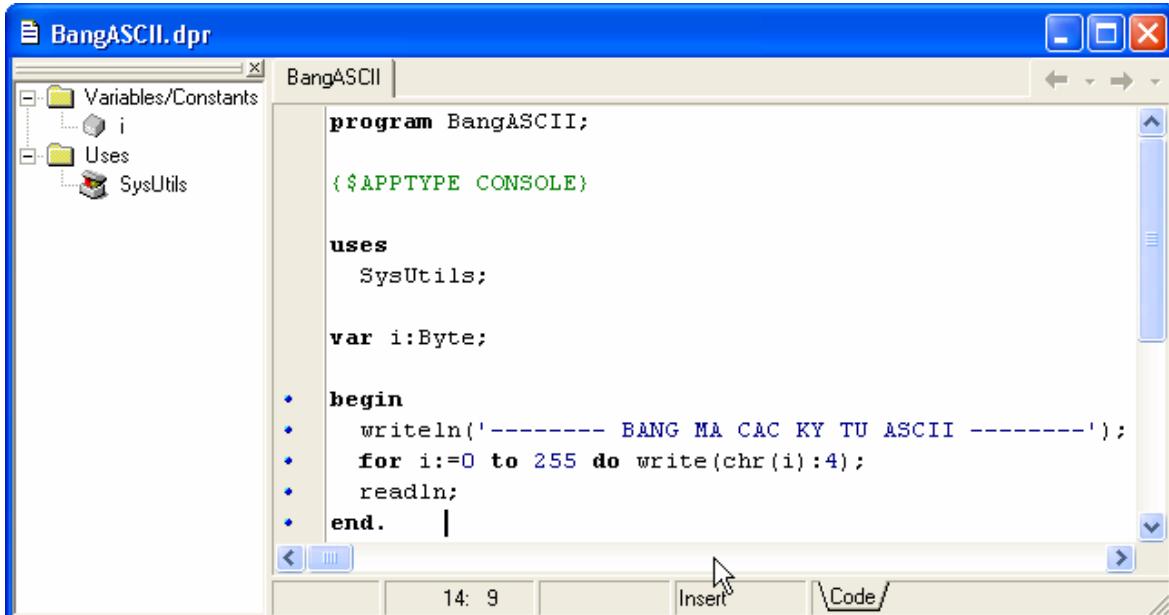
BẢNG MÃ ASCII với 128 mở rộng

Hex	8	9	A	B	C	D	E	F
0	ç 128	É 144	á 160	í 176	ł 192	ł 208	α 224	≡ 240
1	ü 129	æ 145	í 161	í 177	ł 193	ł 209	ß 225	± 241
2	é 130	Æ 146	ó 162	ó 178	ł 194	ł 210	ł 226	≥ 242
3	â 131	ô 147	ú 163	 179	ł 195	ł 211	ł 227	≤ 243
4	ä 132	ö 148	ñ 164	- 180	- 196	ł 212	ł 228	∫ 244
5	à 133	ò 149	Ñ 165	≠ 181	+	ƒ 213	σ 229	ј 245
6	å 134	û 150	ª 166	 182	ƒ 198	ł 214	μ 230	÷ 246
7	ç 135	ù 151	º 167	 183	 199	 215	τ 231	≈ 247
8	ê 136	ÿ 152	ξ 168	¶ 184	 200	 216	Φ 232	º 248
9	ë 137	ö 153	¬ 169	 185	 201	 217	Θ 233	· 249
A	è 138	Ü 154	¬ 170	 186	 202	Γ 218	Ω 234	· 250
B	ï 139	¢ 155	½ 171	 187	 203	■ 219	δ 235	√ 251
C	î 140	£ 156	¼ 172	 188	 204	■ 220	∞ 236	ⁿ 252
D	ì 141	¥ 157	¡ 173	 189	= 205	 221	φ 237	² 253
E	Ä 142	₱ 158	« 174	 190	 206	 222	ε 238	■ 254
F	Å 143	f 159	» 175	 191	 207	■ 223	∩ 239	255

Hình 2: Bảng ASCII mở rộng

Ví dụ 2: Viết chương trình in ra 256 ký tự của bảng mã ASCII:

Từ **File/New/Other**, DClick vào **Console Application** để mở ra 1 dự án mới và nhập các dòng lệnh để hoàn chỉnh như hình sau:



```

program BangASCII;

{$APPTYPE CONSOLE}

uses
  SysUtils;

var i:Byte;

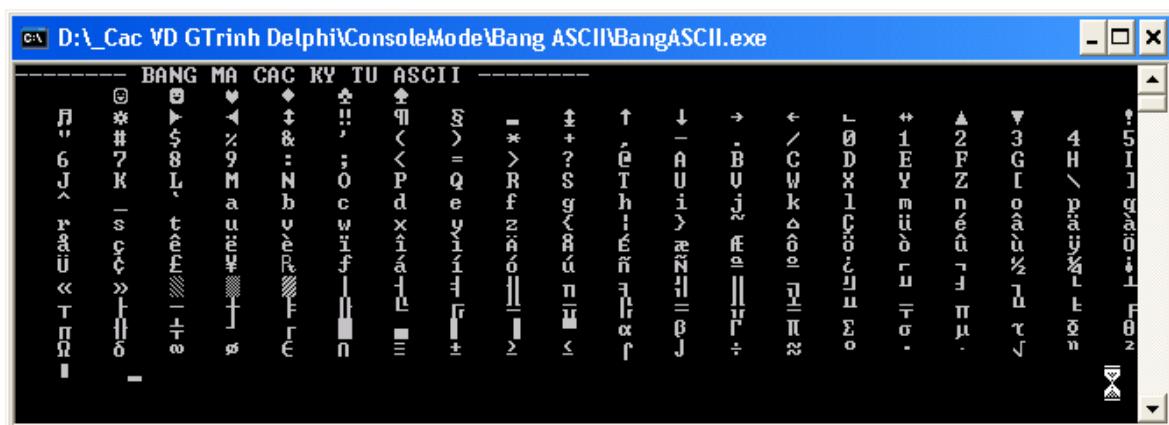
begin
  writeln('----- BANG MA CAC KY TU ASCII -----');
  for i:=0 to 255 do write(chr(i):4);
  readln;
end.

```

Hình 3: Chương trình in bảng mã ASCII

Nhấn tổ hợp phím Ctrl+F9 để biên dịch lỗi.

Nhấn phím chức năng F9 để chạy chương trình, và ta có kết quả ở chế độ Console như sau:



BANG	MA	CAC	KY	TU	ASCII
À	à	á	é	é	À
Ã	ã	ã	ê	ê	Ã
ß	ß	ß	ô	ô	ß
Ö	ö	ö	ü	ü	Ö
Ü	ü	ü	ë	ë	Ü
Ñ	ñ	ñ	ë	ë	Ñ
Ñ	ñ	ñ	ë	ë	Ñ
À	à	á	é	é	À
Ã	ã	ã	ê	ê	Ã
ß	ß	ß	ô	ô	ß
Ö	ö	ö	ü	ü	Ö
Ü	ü	ü	ë	ë	Ü
Ñ	ñ	ñ	ë	ë	Ñ
Ñ	ñ	ñ	ë	ë	Ñ

Hình 4: Kết quả chương trình ở chế độ Console

I.3. Kiểu số thực (Real type)

I.3.1. Khái niệm

Ở Delphi, kiểu số thực biểu diễn một tập hợp các số mang dấu chấm động (float, real). Kích thước của nó tùy thuộc vào từ khóa mà chúng ta sử dụng trong khai báo.

Tùy khóa	Phạm vi	Kích thước (byte)
Real48	$2.9 \times 10^{-39} .. 1.7 \times 10^{38}$	6
Single	$1.5 \times 10^{-45} .. 3.4 \times 10^{38}$	4
Double	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$	8
Extended	$3.6 \times 10^{-4951} .. 1.1 \times 10^{4932}$	10
Comp	$-2^{63}+1 .. 2^{63} -1$	8
Currency	-922337203685477.5808.. 922337203685477.5807	8

Bảng 3: Các tùy khóa của kiểu thực

Tuy nhiên, kiểu số thực chúng ta thường dùng được khai báo bằng tùy khóa **Real**, trong Delphi kiểu Real tương đương với kiểu **Double**.

Tùy khóa	Phạm vi	Kích thước (byte)
Real	$5.0 \times 10^{-324} .. 1.7 \times 10^{308}$	8

Bảng 4: Kiểu Real

I.3.2. Một số phép toán và hàm trên kiểu số (số nguyên và số thực)

Các phép toán số học hai ngôi:

Phép toán	Ý nghĩa	Kiểu toán hạng	Kiểu kết quả
+	Cộng	integer, real	integer, real
-	Trừ	integer, real	integer, real
*	Nhân	integer, real	integer, real
/	Chia thực	integer, real	real
div	Chia nguyên	integer	integer
mod	Lấy phần dư	integer	integer

Bảng 5: Các phép toán 2 ngôi

Ví dụ 3: Với các biến x, y, z kiểu integer

```
x := 3*2;           // x = 6
y := 15 div 6;       // t = 2
z := 15.5 mod 5;     // lõi
```

Tuy nhiên, phép toán + và toán - còn có thể là phép toán một ngôi:

Phép toán	Ý nghĩa	Kiểu toán hạng	Kiểu kết quả	Ví dụ
+ (unary)	dấu dương	integer, real	integer, real	+7
- (unary)	dấu âm (lấy số đối)	integer, real	integer, real	-7

Bảng 6: Các phép toán 1 ngôi

Chương 4: Các kiểu dữ liệu sơ cấp chuẩn, lệnh đơn

Khi thực hiện các phép toán chúng ta cần chú ý một số điểm đặc biệt sau:

- Kết quả của phép chia thực (/) luôn luôn có kiểu Extended, không quan tâm đến kiểu của các toán hạng.
- Phép toán mod là phép toán lấy phần dư.
- Khi thực hiện phép chia x/y (hay là x mod y hoặc x div y) thì y phải khác 0, nếu y có giá trị 0 thì chương trình sẽ phát sinh lỗi.

Một số hàm chuyển đổi kiểu dữ liệu:

```
function IntToStr ( Number : Integer ) : string;      //Unit: SysUtils  
function IntToStr ( BigNumber : Int64 ) : string;     //Unit: SysUtils
```

Hàm IntToStr là hàm thực hiện việc chuyển đổi một số nguyên (hoặc số int64) thành một chuỗi số.

```
function FloatToStr(Value: Extended): string;      //Unit: SysUtils  
function FloatToStr(Value: Extended; const FormatSettings: TFormatSettings): string;
```

Hàm FloatToStr chuyển đổi một số thực hay số dấu chấm động (**Value**) thành một chuỗi biểu diễn tương ứng cho số đó.

Bảng dưới đây là các hàm liên quan đến kiểu số:

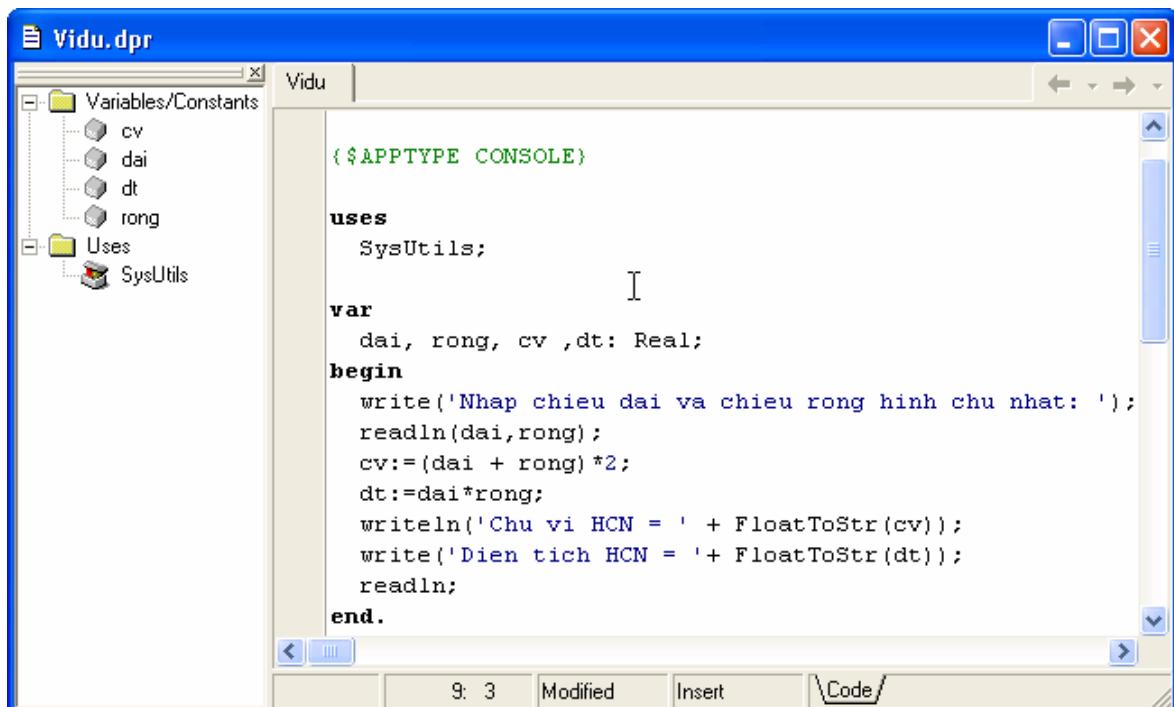
Hàm	Ý nghĩa
abs(x);	$ x $: lấy giá trị tuyệt đối của số x
arctan(x:extended): extended;	arctang(x)
cos(x: extended): extended;	cos(x) : lấy cos của x
exp(x: real): real;	e^x
frac(x:extended): extended;	Trả về phần thập phân của biến x
inc(var x [; n: longint]);	tăng biến x lên 1 hoặc n giá trị
int(x: extended): extended;	Trả về phần nguyên của biến x
ln(x);	ln x : lấy logarit nepe của trị x ($e \approx 2.71828$)
max(a,b: numbertype): numbertype;	Trả về số lớn hơn trong hai số a và b.
min(a,b: numbertype): numbertype;	Trả về số nhỏ thua trong hai số a và b.
odd(n: integer int64):boolean;	Kiểm tra n có phải là số lẻ hay không: True: Nếu n lẻ False: Nếu n chẵn.
pi: extended;	Trả về giá trị pi=3.1415926535897932385.
random[(range: integer)];	Trả về một số ngẫu nhiên x và $0 \leq x < \text{range}$, nếu range không chỉ ra thì $0 \leq x < 1$.
sqr(x);	x^2 : lấy bình phương trị số x
sqrt(x);	\sqrt{x} : lấy căn bậc 2 của x
sin(x);	sin (x) : lấy sin của x
succ(x);	Trả về giá trị sau của x

trunc(x: extended): int64;	Lấy phần nguyên của x
round(x: extended): int64;	Làm tròn giá trị của x, lấy số nguyên gần x nhất

Bảng 7: Các hàm xử lý số

Ví dụ 4: Viết chương trình tính chu vi và diện tích của hình chữ nhật:

Từ **File/New/Other**, DClick vào Console Application để mở ra 1 dự án mới và nhập các dòng lệnh để hoàn chỉnh như hình sau:



```

{$APPTYPE CONSOLE}

uses
  SysUtils;

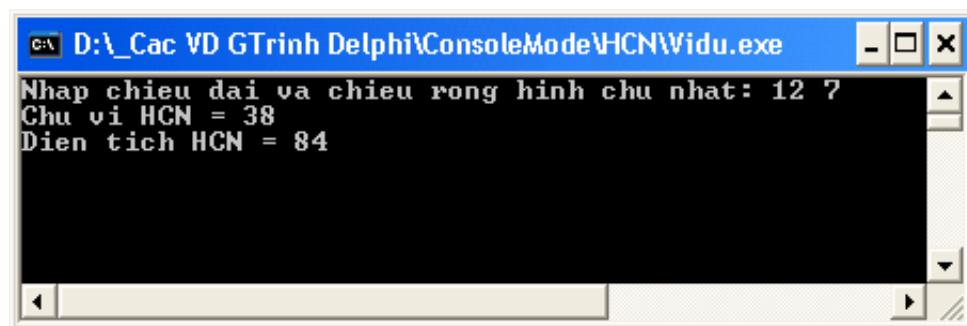
var
  dai, rong, cv, dt: Real;
begin
  writeln('Nhập chiều dài và chiều rộng hình chữ nhật: ');
  readln(dai, rong);
  cv:=(dai + rong)*2;
  dt:=dai*rong;
  writeln('Chu vi HCN = ' + FloatToStr(cv));
  writeln('Diện tích HCN = ' + FloatToStr(dt));
  readln;
end.

```

Hình 5: Chương trình tính chu vi diện tích hình chữ nhật

Nhấn tổ hợp phím Ctrl + F9 để biên dịch lỗi.

Nhấn phím chức năng F9 để chạy chương trình, và ta có kết quả ở chế độ Console như sau:



Hình 6: Kết quả chương trình ở chế độ Console

I.4. Kiểu logic (Boolean)

I.4.1. Khái niệm

Một dữ liệu thuộc kiểu boolean là một đại lượng có thể nhận được một trong hai giá trị logic là **true** (đúng) và **false** (sai). Delphi cung cấp 4 kiểu boolean sau:

Tùy khóa	Kích thước (byte)
Boolean	1
ByteBool	1
WordBool	2
LongBool	4

Bảng 8: Kiểu logic

Trong đó, kiểu Boolean là kiểu tương thích với kiểu Boolean của Pascal và được ưa thích hơn cả. Một giá trị kiểu ByteBool, LongBool hoặc WordBool được xem là true nếu nó có giá trị khác 0, ngược lại là false.

I.4.2. Phép toán Boolean

Tùy khóa	Ý nghĩa
Not	Phép toán phủ định
And	Phép toán và
Or	Phép toán hoặc

Bảng 9: Các phép toán trên kiểu logic

Cú pháp:

- Phép toán **and**: **(x1) and (x2) and ... and (xn)** trả về kết quả là true khi tất cả các toán hạng là true, ngược lại trả về false.
- Phép **or**: **(x1) or (x2) or ... or (xn)** cho kết quả là false khi tất cả các toán hạng là false, ngược lại trả về true.

Ví dụ 5: Ví dụ: với x, y, z kiểu boolean, ta có:

```
x := (2<3) and (5<5) and (0<>1); // x = False
y := (2>3) or (5>5) or (0<>1); // y = True
z := not (y); // z = False
```

Để hiển thị giá trị boolean lên màn hình, bạn sử dụng hàm có sẵn trong Delphi:

```
function BoolToStr(B: Boolean; UseBoolStrs: Boolean = False): string;
```

Nếu bạn muốn hiển thị giá trị của B là chuỗi ký tự True hoặc False thì cho UseBoolStrs = True; còn nếu muốn hiển thị giá trị của B là '-1' hoặc '0' thì cho UseBoolStrs = False.

II. Câu lệnh (statement)

Trong một chương trình Delphi, sau phần mô tả dữ liệu là phần mô tả các câu lệnh. Các câu lệnh có nhiệm vụ xác định các công việc mà máy tính phải thực hiện để xử lý các dữ liệu đã được mô tả và khai báo. Câu lệnh được chia thành câu lệnh đơn giản và câu lệnh có cấu trúc.

Các câu lệnh phải được ngăn cách với nhau bởi dấu **chấm phẩy** (;) và các câu lệnh có thể viết trên một dòng hay nhiều dòng.

III. Lệnh đơn (Simple statement)

Lệnh đơn là lệnh không chứa bất kỳ câu lệnh nào khác. Các lệnh đơn bao gồm: lệnh gán, lệnh gọi hàm và thủ tục, lệnh nhảy (**goto**).

IV. Lệnh gán (Assignment statement)

Một trong các lệnh đơn giản và cơ bản nhất của Delphi là lệnh gán. Mục đích của lệnh này là gán cho một biến đã khai báo một giá trị nào đó cùng kiểu (hay tương thích) với biến.

Cú pháp của lệnh gán là:

Biến := biểu_thức;

Lệnh gán sẽ đặt lại giá trị hiện hành của biến bằng giá trị mới nằm bên phải toán tử gán (dấu hiệu :=).

Ví dụ 6: Khi đã khai báo

var

c: char;	i: integer;
x: real;	p, q: boolean;

thì ta có thể có các phép gán sau:

c := 'A';	c := chr(90);
i := 35;	i := i div 7;
x := 0.5;	x := i + 1;
p := (i >= 1) and (i < 100);	q := not p;

Ý nghĩa:

Biến và các phát biểu gán là các khái niệm quan trọng của một họ các ngôn ngữ lập trình, chúng phản ánh cách thức hoạt động của máy tính, đó là:

- Lưu trữ các giá trị khác nhau vào một ô nhớ tại những thời điểm khác nhau.
- Một quá trình tính toán có thể coi như là một quá trình làm thay đổi giá trị của một (hay một số) ô nhớ nào đó, cho đến khi đạt được giá trị cần tìm.

V. Lệnh gọi thủ tục và hàm

Trong lập trình OOP, thuật ngữ hàm (Function), thủ tục (Procedure) và phương thức (Method) là khái niệm quen thuộc với người lập trình. Để các bạn có thể nắm bắt được kiến thức lập trình trong Delphi, Giáo trình đưa ra trước khái niệm này một cách ngắn gọn như sau:

Hàm và thủ tục được gọi chung là chương trình con. Hàm được xây dựng thông qua tên hàm và danh sách các tham số hình thức nếu có, **mỗi hàm thì luôn trả về duy nhất 1 giá trị**. Còn thủ tục thì có tên thủ tục và danh sách các tham số hình thức nếu có. Khác với

hàm, **thủ tục KHÔNG trả về một giá trị nào cả**, nó thường bao gồm một dãy các câu lệnh cần được thực hiện một cách tuần tự để giải quyết một công việc nào đó.

Bạn cần phân ra 2 giai đoạn rõ ràng: Xây dựng chương trình con, và sử dụng chương trình con.

Ví dụ 7: Về hàm - Function

- Xây dựng hàm: Bạn viết hàm tính tổng 2 số a và b và kết quả trả về của hàm chính là kết quả a + b. Nó có hình dạng như sau:

```
function Cong(a, b: single) : single;
begin
    Cong := a + b;
end;
```

- Sử dụng hàm (Lời gọi hàm): bạn cần tính phép cộng của 2 con số cụ thể nào, chẳng hạn bạn muốn 5 cộng với 7, thì như vậy bạn sẽ truyền số 5 vào cho a và 7 vào cho b. Cụ thể bạn sẽ gọi như sau:

```
mTong := Cong(5, 7); //với biến mTong kiểu Single và mTong = 12
```

Như vậy ở bước xây dựng: a và b được gọi là tham số hình thức. Còn ở bước gọi hàm thì 5 và 7 được gọi là các tham số thực.

Ví dụ 8: Về thủ tục – Procedure

- Xây dựng thủ tục: Bạn viết thủ tục hiển thị ra màn hình n câu chào “Welcome to Delphi!...”, với n có thể là 1 lần, 3, 10, ... 100 lần chẳng hạn:

```
procedure Welcome(n: integer);
var i: integer;
begin
    for i:=1 to n do writeln ('Welcome to Delphi!...');
end;
```

- Sử dụng thủ tục (Lời gọi thủ tục): ở đây muốn hiển thị 5 câu chào, thì bạn ra câu lệnh như sau:

```
Welcome(5); //Hiển thị 5 câu chào
```

Khái niệm phương thức trong một lớp cũng được xem là thủ tục hoặc hàm (Member method). Kiến thức này các bạn sẽ được học ở chương 5. Ở đây, các bạn cần chú ý điều này: Phương thức là chương trình con mà Delphi đã xây dựng sẵn trong mỗi lớp/thành phần (Class/Component) cho bạn (Thật khỏe làm sao!...), bạn chỉ cần lấy trong “túi khôn” của Delphi để sử dụng viết chương trình mà thôi.

Tóm lại: Khi sử dụng phương thức hay các chương trình con đã được xây dựng sẵn bởi Delphi, bạn cần nắm rõ mỗi phương thức, thủ tục hoặc hàm có bao nhiêu tham số hình thức và kiểu của chúng là gì để truyền vào cho chúng các tham số thực tương ứng cho đúng yêu cầu của bạn.

Ví dụ với cú pháp thủ tục có sẵn trong Delphi là:

```
procedure ShowMessage(const Msg: string);
```

thì bạn chỉ cần lấy ra sử dụng với câu lệnh đơn giản như sau:

```
ShowMessage('Toi dang hoc Delphi day nhe!...');
```

Các bạn sẽ được học chi tiết các khái niệm này trong chương “Lập Trình Xử Lý Sự Kiện” và “Chương Trình Con” ở các phần sau.

VI. Lệnh Goto

Tùy thân chương trình con, bạn có thể sử dụng lệnh goto để chuyển hướng thực hiện chương trình tới một vị trí bất kỳ thông qua một nhãn đã được khai báo trước.

Cú pháp lệnh goto: **Goto** nhãn;

Với nhãn được khai báo với cú pháp sau: **Label** Tên_nhãn;

Lệnh goto là lệnh nhảy không điều kiện, nó chuyển hướng thực thi của chương trình đến vị trí mà nhãn đang đứng.

Trong phong cách lập trình OOP, lệnh **goto** tỏ ra bị lỗi thời và rất ít khi được dân lập trình sử dụng.

Ví dụ 9: Minh họa lệnh **goto** để tính biểu thức a/b.

Từ **File/New/Other**, DClick vào Console Application để mở ra 1 dự án mới và nhập các dòng lệnh để hoàn chỉnh như hình sau:

```

program prjLabel;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a, b: extended;
label mLoi, mKetQua;
begin
  write('Nhập a và b = ');
  readln(a,b);
  if b<>0 then goto mKetqua; //nhảy đến phần tính toán kết quả
  goto mLoi; //b=0 thì nhảy đến phần báo lỗi.
  mKetqua: write('Giá trị của biểu thức a/b = ' + FloatToStr(a/b));
  readln;
  exit; //ket thuc chuong trinh/thu tuc.
  mLoi: write('Loi!... Phep toan chia 0 (zero).');
  readln;
end.

```

Hình 7: Ví dụ minh họa lệnh **goto** và khai báo nhãn - **label**

Nhấn phím chức năng F9 để chạy chương trình, và ta có kết quả ở chế độ Console như sau:

```

S:\GT Del-03.2007\LabelGoto\prjLabel.exe
Nhập a và b = 10 5
Giá trị của biểu thức a/b = 2

Nhập a và b = 5 0
Loi!... Phep toan chia 0 (zero).

```

Hình 8: Kết quả chương trình.

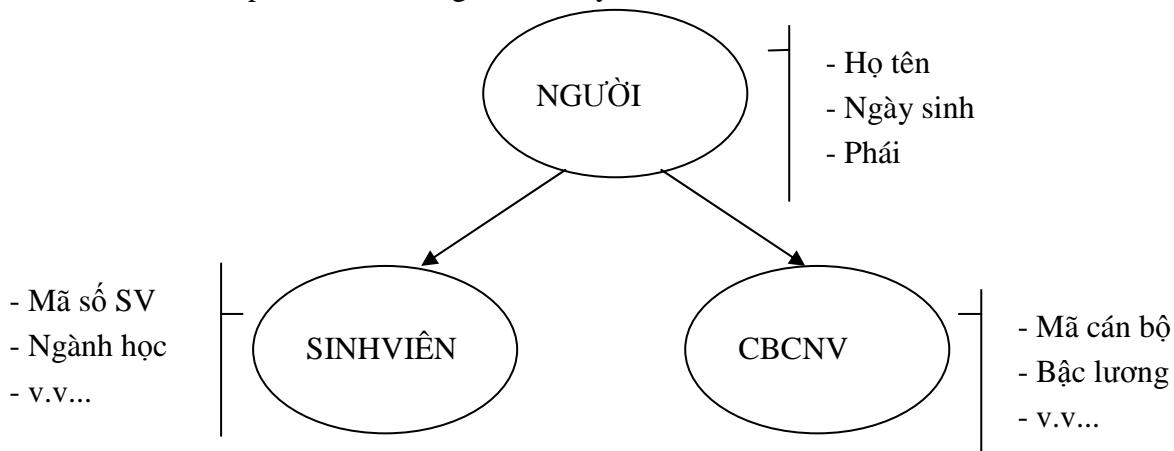
CHƯƠNG 5: LẬP TRÌNH XỬ LÝ SỰ KIỆN – CÁC THÀNH PHẦN TRONG GIAO DIỆN DELPHI

I. Lập trình xử lý sự kiện

I.1. Lớp (Class) và đối tượng (Object)

Trong các chương trước, các bạn đã được biết đến cách lập trình tuần tự ở chế độ Console Application và biết được một số khái niệm cơ sở. Ở chương này, các bạn sẽ được trình bày một phương pháp lập trình chính của giáo trình và rất phổ biến hiện nay đó là lập trình sự kiện dựa trên nền tảng của lập trình hướng đối tượng (OOP: Object-Oriented Programming).

Trong cuộc sống, bạn thấy có nhiều đối tượng được xếp chung vào một loại/lớp, chẳng hạn như những người còn đi học ta xếp chung với một lớp đó là lớp SINHVIÊN, những người đi làm việc thì xếp vào lớp CBCNV. Như vậy ta có hình ảnh như sau: tất cả mọi người đều được xếp chung vào một lớp trên cùng là lớp NGƯỜI. Ta cũng có thể phân chia những người trong lớp NGƯỜI này thành hai nhóm là SINHVIÊN và CBCNV dựa vào tính chất để phân biệt là đang đi làm hay còn đi học.



Hình 1: Mô hình các lớp

Cụ thể: Nếu xét một gia đình công chức gồm 4 thành viên là người A – cha, B – mẹ; đi làm; anh C, chị D: đang đi học chẳng hạn, thì như thế ta có 2 đối tượng A, B được tạo ra từ lớp CBCNV và 2 đối tượng C, D được sinh ra từ lớp SINHVIÊN và cả 4 đối tượng này đều có những tính chất chung được thừa hưởng từ lớp NGƯỜI, và như vậy 2 lớp SINHVIÊN và CBCNV là các lớp dẫn xuất từ lớp NGƯỜI.

I.1.1. Lớp (Class)

Lớp là một khái niệm được sử dụng rất nhiều trong cách lập trình OOP. Việc hiểu rõ các kiến thức về lớp thì nằm ngoài phạm vi của giáo trình này. Do vậy, giáo trình chỉ đưa ra các khái niệm rất cơ bản về lớp, mục đích để các bạn dễ hiểu hơn về đối tượng (Object), là một thành phần rất phổ biến trong lập trình của Delphi. Như vậy, lớp được dùng để định nghĩa ra kiểu dữ liệu cấu trúc mà nó bao gồm 3 thành phần cơ bản:

- Các trường (Fields): là các biến dữ liệu của lớp.

- Các phương thức (Methods): là các hàm hay thủ tục để thực hiện thao tác nào đó trên đối tượng. Có 2 phương thức đặc biệt trong lớp mà bạn cần lưu ý đó là phương thức tạo (Constructor) và phương thức hủy (Destructor) sẽ được dùng khi một đối tượng (Object) được tạo nên hoặc bị hủy.
- Các thuộc tính (Attributes) của một lớp bao gồm các biến, các hằng, hay tham số nội tại của lớp đó. Tính chất quan trọng nhất của các thuộc tính khi chúng là biến là có thể bị thay đổi trong quá trình hoạt động của một đối tượng. Thông thường để truy xuất các biến trường, ta dựa vào trạng thái của thuộc tính là đọc (read) hay ghi (write) để nhận hay là thiết lập giá trị cho chúng.

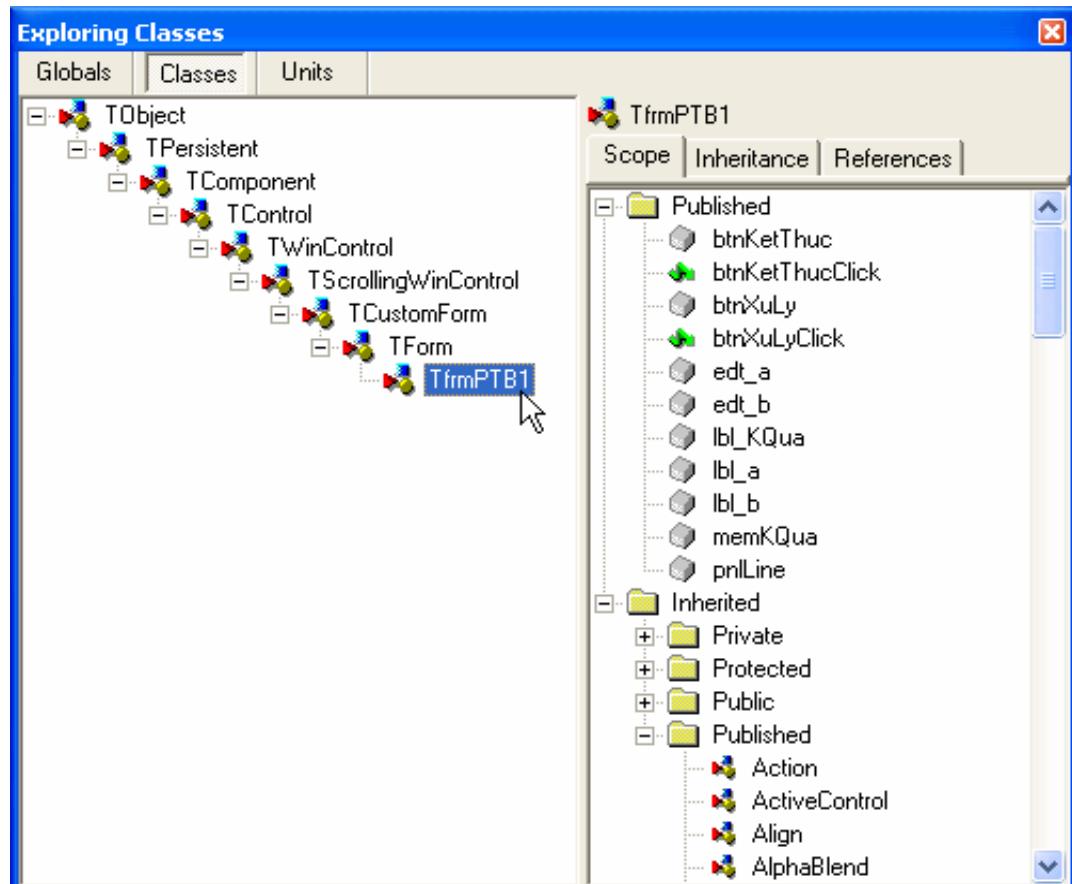
Lớp được khai báo theo cú pháp như sau:

```
type className = class (ancestorClass)
    memberList
end;
```

Trong đó:

- **ancestorClass** là lớp cha/tổ tiên của lớp dẫn xuất (descendant) là className, lớp className sẽ được thừa hưởng (inheritance) từ lớp cha. Trong Delphi lớp thủy tổ là lớp TObject.
- **memberList** là các thành phần của lớp là trường, phương thức và thuộc tính của lớp đã được trình bày ở trên.

Trong ví dụ giải phương trình bậc nhất ở trên, ta có một lớp TfrmPTB1 được dẫn xuất từ lớp cha là lớp TForm, lớp TForm lại được dẫn xuất từ lớp TCustomForm,... và cứ thế nó được dẫn xuất từ lớp thủy tổ trong Delphi là lớp TObject như hình sau:



Hình 2: Lớp TfrmPTB1 và các lớp tổ tiên (Ancestor Classes) của nó

Ví dụ 1: Khai báo lớp TfrmPTB1 cho form unit **untPtb1.Pas** ở ví dụ trên như sau:

```

type
  TfrmPTB1 = class(TForm)
    lbl_a: TLabel;
    lbl_b: TLabel;
    edt_a: TEdit;
    edt_b: TEdit;
    lbl_KQua: TLabel;
    memKQua: TMemo;
    pnlLine: TPanel;
    btnXuLy: TButton;
    btnKetThuc: TButton;
    procedure btnXuLyClick(Sender: TObject);
    procedure btnKetThucClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

```

I.1.2. Đối tượng (Object)

Đối tượng là thực thể (Instance) được sinh ra từ một lớp (Class), hay nói cách khác khi ta khai báo một biến thuộc một lớp nào đó, thì biến đó chính là một đối tượng mới được cấp phát. Đối tượng này được tạo ra bởi phương thức constructor và sẽ bị giải phóng bởi phương thức destructor.

* Quy cách đặt tên đối tượng:

Trong phong cách lập trình chuyên nghiệp, quy ước đặt tên cho đối tượng thường được ghép 2 phần: **tiền tố và tên gọi nhớ** của đối tượng đó. Tiền tố thường có 3 ký tự để xác định đối tượng đó thuộc loại thành phần nào. Trong giáo trình này gợi ý cho các bạn các tiền tố (prefix) cho các đối tượng của các thành phần như sau:

Đối tượng	Tiền tố	Ví dụ
Form	frm	frmWelcome, frmMain
Label	lbl	lblHeSo, lblHoTen
Edit	edt	edt_a, edtHoTen
Button	btn	btnGiai, btnXuLy
SpinEdit:	spe	speNhiетDo, speGia1TC
LabeledEdit	lbe	lbeĐonGia, lbeThanhTien
ListBox	lbx	lbxBaiHat, lbxFONT
CheckBox	chk	chkMatHang, chkDelphi
ComboBox	cbo	cboToanTu, cboPhai
GroupBox	gbx	gbxMonHoc, gbxMatHang

Đối tượng	Tiền tố	Ví dụ
CheckListBox	clb	clbTenMH, clbDMVT
ValueListEditor	vle	vleMatHang, vleBaoGia
Memo	mem	memDanhSach, memKQ
RadioButton	rbt	rbtNam, rbtNu
RadioGroup	rgp	rgpGioiTinh, rgpCourses
StringGrid	stg	stgAlphabet, stgMaTranA
Panel	pnl	pnlDieuKhien, pnlDrives
PopupMenu	pmu	pmuMauSac, pmuGames
MainMenu	mmu	mmuMainForm, mmuQLSV
MenuItem	mni	mniCopy, mniPaste

Bảng 1: Quy cách đặt tên các đối tượng

Ví dụ 2: Khai báo một thực thể (đối tượng) frmPTB1 như sau:

var

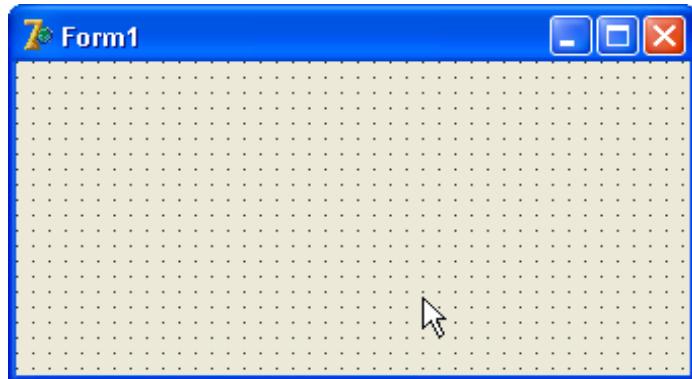
frmPTB1: TfrmPTB1; // đối tượng frmPTB1 của lớp TfrmPTB1

Lập trình hướng đối tượng là sự gắn kết giữa dữ liệu và các hàm xử lý dữ liệu tạo thành đối tượng. Mỗi đối tượng bao gồm:

- Thuộc tính (Properties): là dữ liệu của đối tượng.
- Phương thức (Methods): là những hành động mà đối tượng có thể thi hành.
- Sự kiện (Events): là những hành động của đối tượng đó đáp ứng lại thông qua thủ tục sự kiện do người lập trình cài đặt, khi người dùng tương tác với chương trình.

Ví dụ 3: Trong chương này, các bạn sẽ được học cách sử dụng các thuộc tính và các sự kiện của các đối tượng thông qua một chương trình đầu tiên là “**prjWelcome.dpr**”

- **Bước 1:** Bạn vào chức năng: **File/New/Application** để tạo ra dự án mới.



Hình 3: Hình ảnh của New Form.

Từ cửa sổ thuộc tính và sự kiện của đối tượng (Object Inspector), dựa vào thuộc tính name, bạn đặt tên cho form là frmWelcome để Delphi định nghĩa ra 1 lớp có tên là TfrmWelcome được dẫn xuất từ lớp cha là TForm như sau:

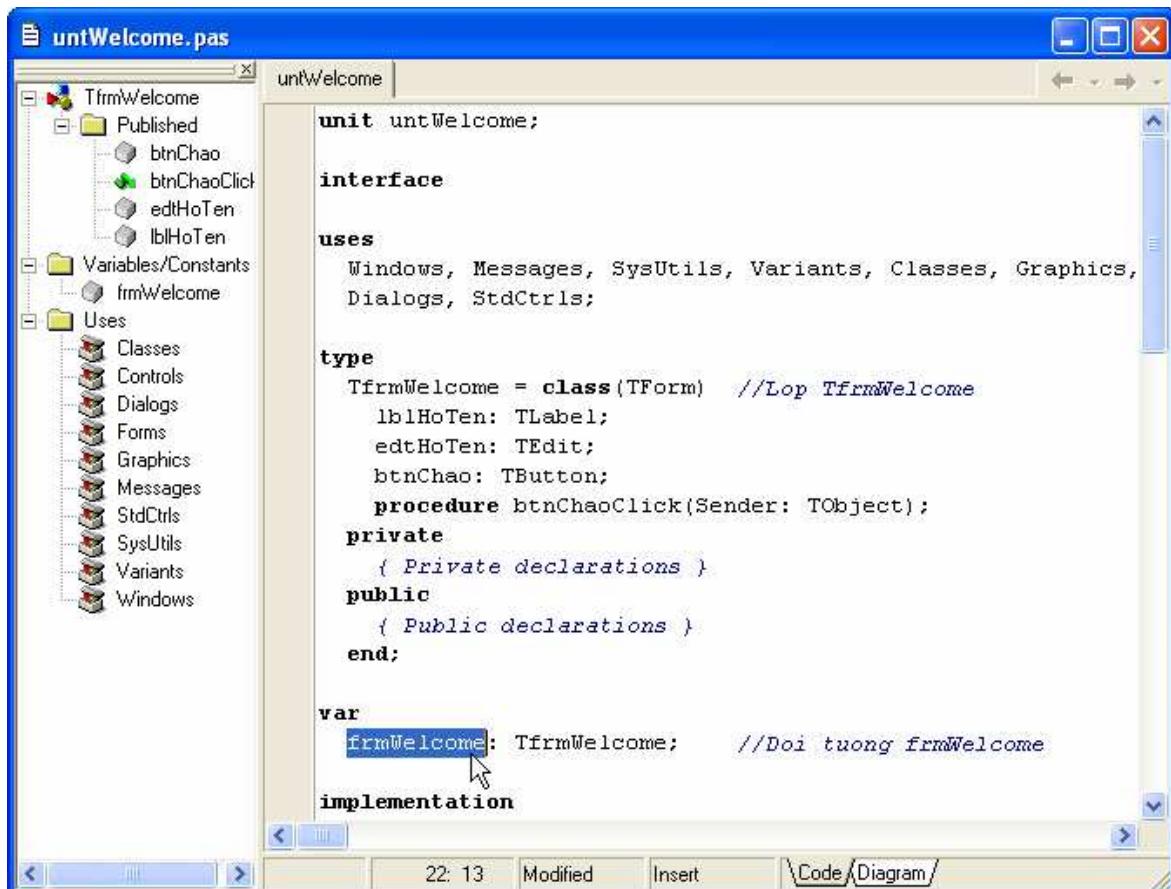
type

```
TfrmWelcome = class(TForm)
  // memberList
end;
```

Và khai báo một đối tượng có tên là frmWelcome thuộc kiểu lớp TfrmWelcome như sau:

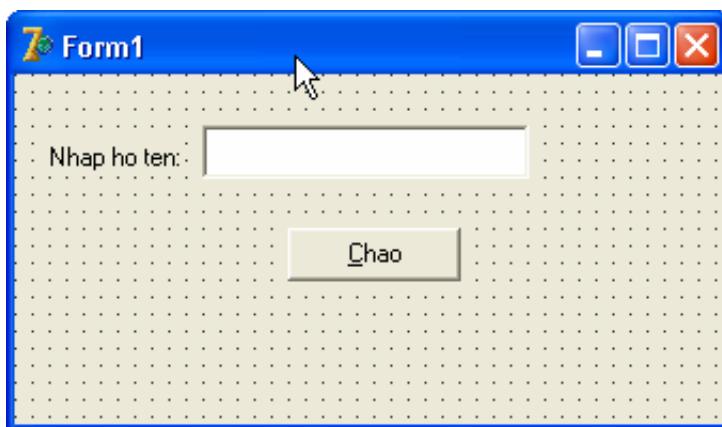
var

```
frmWelcome: TfrmWelcome;
```



Hình 4: Hình ảnh lớp TfrmWelcome và đối tượng frmWelcome

- **Bước 2:** Từ bảng các thành phần (Component Palette), click vào thẻ Standard rồi lần lượt đặt các thành phần TLabel, TEdit, và TButton lên Form:



Hình 5: Các đối tượng Label, Edit và Button đặt trên Form.

Trong môi trường lập trình IDE, các lớp trong Delphi thường được cài đặt thành các thành phần (TComponent) để người lập trình dễ dàng thửa hướng khi thiết kế và viết mã lệnh. Như vậy: Tên lớp cũng chính là tên thành phần.

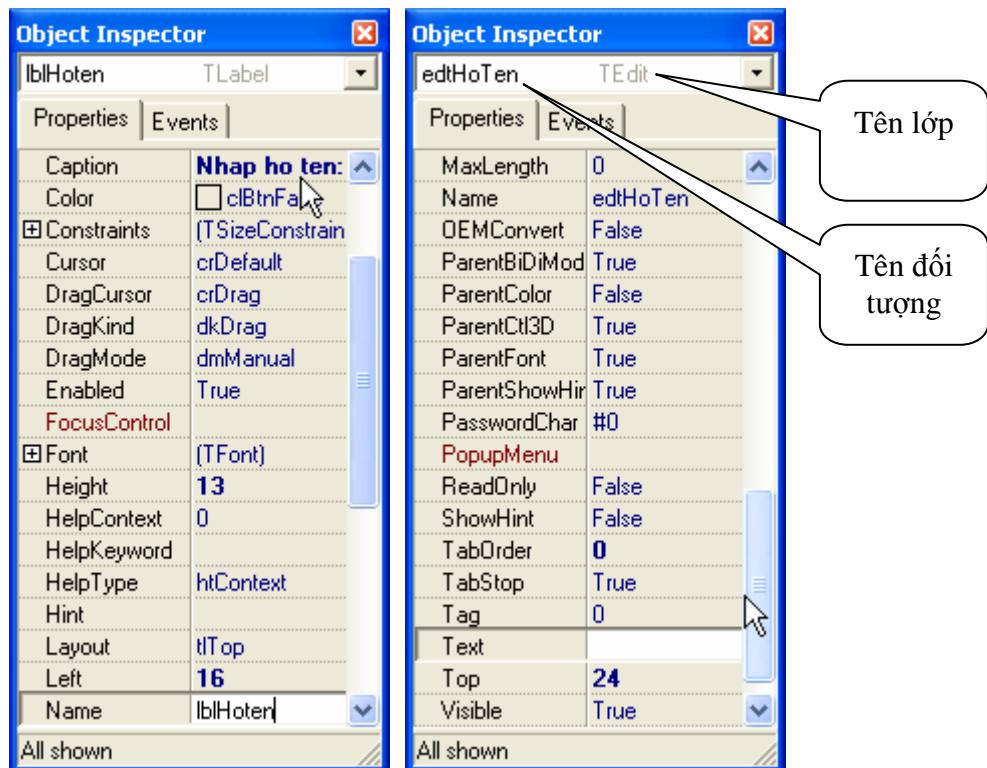
Từ đây về sau, thuật ngữ “**đối tượng A của một thành phần TA**” được viết ngắn gọn là “**đối tượng A**”.

Ví dụ 4: Câu lệnh **frmWelcome:TForm;** đọc đầy đủ là “đối tượng frmWelcome thuộc lớp (thành phần) TForm”. Tuy nhiên, ta có thể đọc gọn lại như: “**đối tượng frmWelcome**”, hoặc câu lệnh **btnGiai:TButton;** được đọc là “**nút lệnh btnGiai**” hay “**đối tượng btnGiai**”.

I.2. Thuộc tính

Mỗi thành phần đều có những thuộc tính để xác định những đặc tính của thành phần đó chẳng hạn như màu sắc (Color), vị trí (Position), kích thước và trạng thái (State) của chúng. Trong các thuộc tính này, bạn cần quan tâm nhất đến thuộc tính tên (Name) của đối tượng.

- **Bước 3:** Lần lượt thiết lập các thuộc tính Name, Caption cho đối tượng Label và các thuộc tính Name, Text cho đối tượng Edit:



Hình 6: Cửa sổ thuộc tính và sự kiện của đối tượng lblHoten và edtHoten

Tương tự, đối tượng thuộc thành phần TButton có thuộc tính Name là btnChao và Caption là “&Chao”, đối tượng của TForm có thuộc tính Name là frmWelcome và Caption là “Welcome to Delphi 7.0”

Cách truy xuất thuộc tính của đối tượng:

Tên_đối_tượng.Thuộc_tính

Ví dụ 5: Gán thuộc tính Enabled của lblHoten bằng True là:

```
lblHoten.Enabled := True;
```

I.3. Phương thức (Method)

Là thủ tục (Procedure) hoặc hàm (Function) đã được xây dựng sẵn cho công việc nào đó của đối tượng. Ta có thể gọi phương thức này trong quá trình viết mã (Coding) để chúng thi hành khi chương trình chạy.

Cách truy xuất phương thức đối tượng:

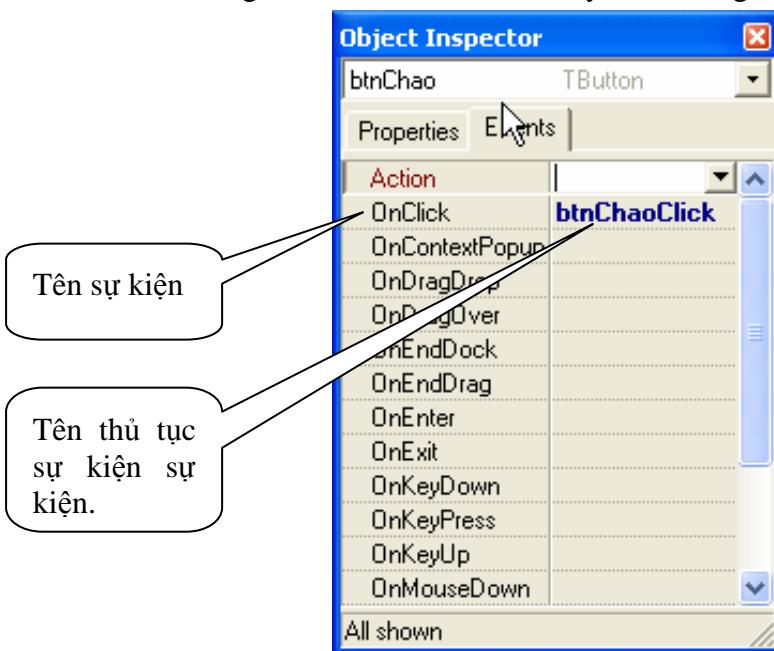
Tên đối tượng.Phương thức;

Ví dụ 6: Gọi phương thức Hide và Show để thực hiện việc ẩn và hiện của đối tượng frmWelcome là: frmWelcome.Hide;

và frmWelcome.Show;

I.4. Sự kiện (Event)

Đó chính là một thao tác của người sử dụng tác động lên đối tượng và nó sẽ đáp ứng lại bằng việc thực hiện một công việc nào đó. Các sự kiện thông thường là trên mouse và keyboard như: Click, RightClick, DoubleClick, KeyPress, Drag and drop,...



Hình 7: Sự kiện OnClick của btnChao

Như hình trên, đối tượng btnChao có nhiều sự kiện như: OnClick, OnContextPopup, OnDragDrop,... của tab Event tương ứng trên cột Action. Mỗi sự kiện này tương ứng với một thủ tục sự kiện để xử lý sự kiện đó (Event handlers) nằm ở cột bên phải tương ứng.

I.5. Xử lý sự kiện (Event Handlers)

Đó chính là thủ tục sự kiện, thân thủ tục sự kiện này bao gồm các câu lệnh mà người lập trình viết trong nó để thực hiện một công việc, chức năng nào đó.

Cách hình thành thủ tục sự kiện:

```
procedure Tên_Thành_Phần_Form.Tên_đối_tượngSự_kiện([DanhSáchCácThamSố]);
```

Ví dụ 7: Thủ tục sự kiện OnClick cho nút lệnh btnChao của frmWelcome (hay còn gọi là thủ tục sự kiện btnChaoClick của lớp TfrmWelcome) được hình thành như sau:

```
procedure TfrmWelcome.btnChaoClick(Sender: TObject);
```

tên thành phần hay
tên lớp form

tên nút lệnh
btnChao

sự kiện
OnClick

- **Bước 4:** Ở **hình 5**, nhấp đúp lên nút lệnh Chao, hoặc ở **hình 7** bạn nhấp đúp lên cột bên phải của sự kiện OnClick tương ứng để mở ra một thủ tục sự kiện có tên là TfrmWelcome.btnChaoClick (Sender: TObject) như sau:

```

untWelcome.pas
unit untWelcome;
interface
var
  frmWelcome: TfrmWelcome; //Doi tuong frmWelcome
implementation
{$R *.dfm}

procedure TfrmWelcome.btnChaoClick(Sender: TObject);
begin
{ ..... }
  {Than thu tuc su kien, ban go lenh vao day.}
{.....}
end;
end.

```

Hình 8: Thủ tục sự kiện btnChaoClick mới tạo của sự kiện OnClick

Tiếp đến bạn gõ một số lệnh vào trong thân của thủ tục sự kiện như hình sau:

The screenshot shows the Delphi IDE with the code editor open for the file 'untWelcome.pas'. The code is as follows:

```

var
  frmWelcome: TfrmWelcome; //Đối tượng frmWelcome

implementation

{$R *.dfm}

procedure TfrmWelcome.btnChaoClick(Sender: TObject);
begin
  frmWelcome.Hide; // phương thức Hide của frmWelcome
  ShowMessage('Chào bạn "'+edtHoTen.Text+'" đến với Delphi 7.0!...');

  frmWelcome.Show; // phương thức Show của frmWelcome
end;
end.

```

A tooltip window is displayed over the word 'Show' in the 'ShowMessage' line, listing several methods available for the 'frmWelcome' object:

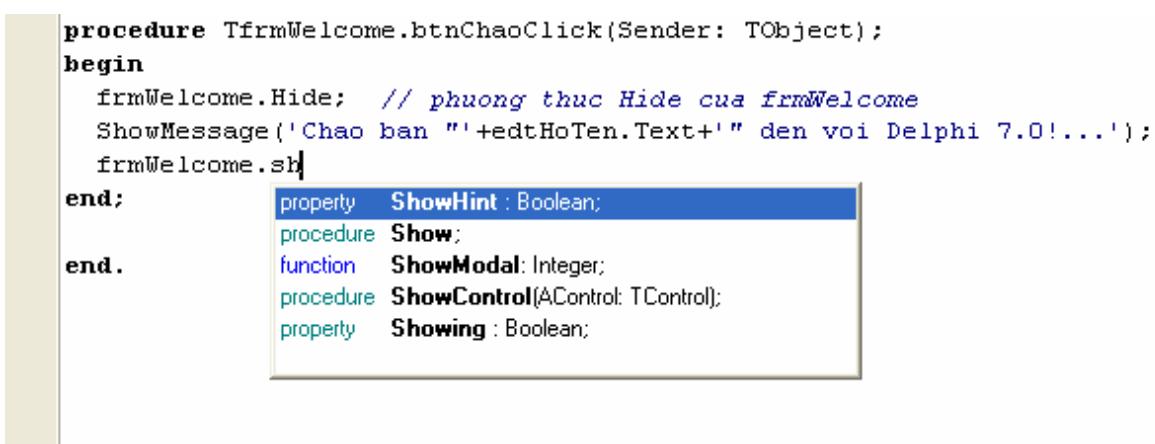
- property ShowHint : Boolean;
- procedure Show;
- function ShowModal: Integer;
- procedure ShowControl(AControl: TControl);
- property Showing : Boolean;

Hình 9: Viết code cho thủ tục sự kiện btnChaoClick

I.6. Trình hỗ trợ mã lệnh (Code Completion /IntelliSense)

Trong cách lập trình hướng đối tượng (OOP) và nhất là khi ngôn ngữ lập trình có hỗ trợ môi trường phát triển IDE thì số só thuộc tính, phương thức, sự kiện và các thủ tục sự kiện của từng đối tượng là vô cùng phong phú (do tính thừa hưởng). Việc nhớ chính xác cú pháp của chúng là một việc làm không tưởng đối với bất kỳ một lập trình viên nào (vì không chỉ có duy nhất một ngôn ngữ lập trình Delphi!). Để giúp cho người lập trình "dễ thở" hơn, Delphi có đưa ra menu ngữ cảnh rất hữu ích để hỗ trợ cho việc viết code này và được gọi là **Trình hỗ trợ mã lệnh (Code Completion)**.

Trong lúc bạn viết code, khi bạn gõ tên đối tượng rồi đến dấu chấm, thì Code Completion sẽ xuất hiện như hình sau:

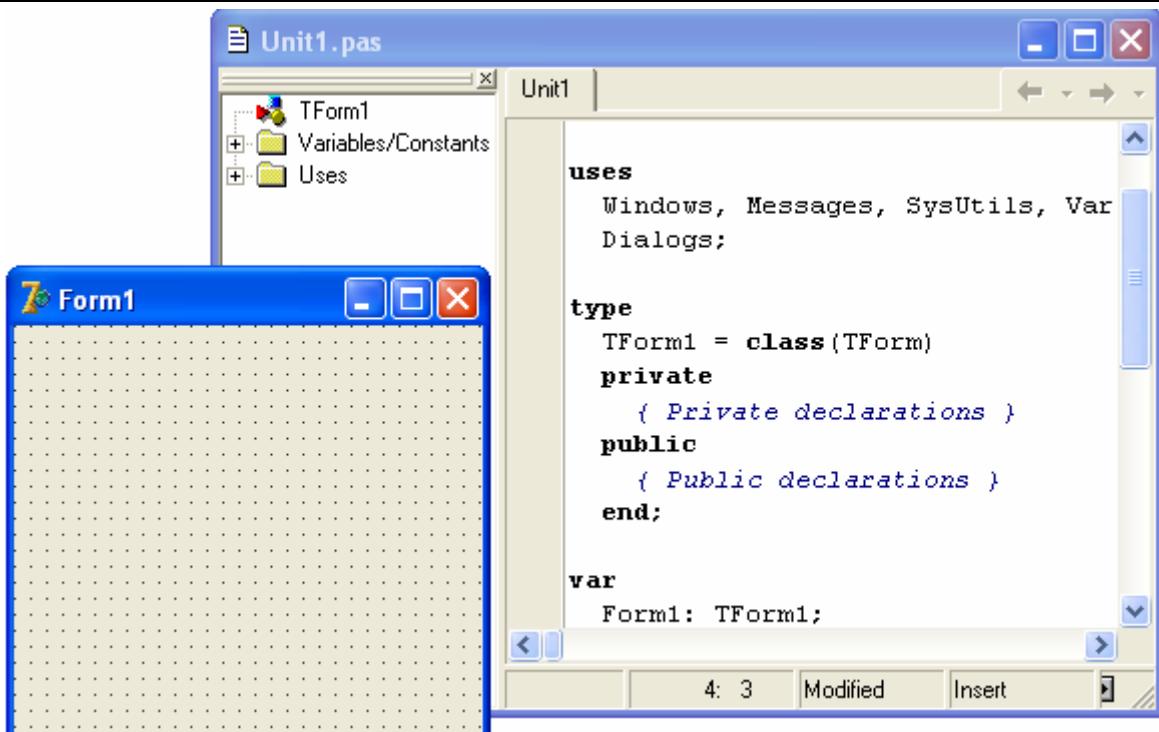


Hình 10: Trình hỗ trợ mã lệnh

II. Sinh mã tự động và một số cách sửa lỗi

II.1. Cách tự động sinh mã (generate code) trong Delphi

- Khi **mới** tạo mới một dự án (Form Application) như hình ảnh sau:



Hình 11: Mã lệnh sinh tự động trong dự án mới.

thì bạn thấy tên kiểu (Type) lớp TForm1 được xây dựng thông qua lớp (Class) dẫn xuất từ lớp cha là lớp TForm. Trong kiểu TForm1 (không kể sự thừa hưởng từ lớp TForm) chưa có thuộc tính (Properties), Phương thức (Methods) hoặc sự kiện (Events) nào của chính nó, như bạn thấy trong đoạn code sau đây:

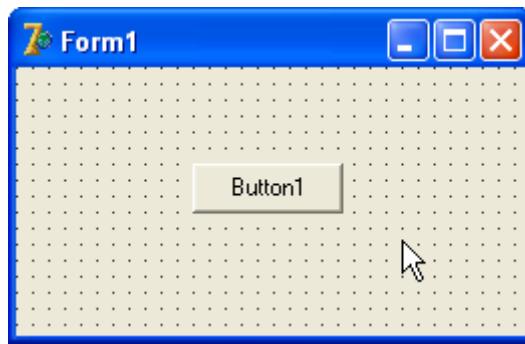
```

type
  TForm1 = class(TForm)
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

```

- Khi bạn đặt lên form (hoặc gỡ bỏ) một đối tượng (Object) của đối tượng bất kỳ như Button, Edit,... thì tên (name) của đối tượng đó sẽ được tự động thêm vào (hoặc xóa bỏ đi) trong phần thuộc tính (Property) của Form ở phần định nghĩa kiểu lớp Form. Chẳng hạn, bạn thêm vào đối tượng Button1 thuộc thành phần TButton (ngắn gọn là Button1: TButton) lên Form như hình sau:



Hình 12: Thêm nút lệnh Button1

thì đoạn code sẽ tự động thêm vào biến dữ liệu là Button1: TButton như sau:

```

type
  TForm1 = class(TForm)
    Button1: TButton;      //mới tự động sinh mã thêm
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

```

- Khi bạn tạo ra 1 thủ tục sự kiện dựa trên sự kiện nào đó, thì Delphi sẽ tự động khai báo (Declare) thêm phần nguyên mẫu (Prototype) của thủ tục sự kiện trong phần dữ liệu (thuộc tính – Properties) của lớp. Chẳng hạn, bạn tạo ra thủ tục (Procedure) của sự kiện (Event) nhấp chuột (OnClick) của nút lệnh có tên Button1, thì Delphi sẽ tự động thêm vào phần khai báo thủ tục sự kiện trong lớp TForm1 như đoạn code sau:

```

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject); //mới thêm vào
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

```

và sự cài đặt (Implementation) các câu lệnh cho thủ tục sự kiện mà ta mới tạo ở sau từ khóa **implementation**, như đoạn code sau:

```

type
    TForm1 = class(TForm)
        Button1: TButton;
        procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

{$R *.dfm}

//----- phần cài đặt thủ tục mới được tự động thêm vào -----
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
// -----

end. // Kết thúc Form Unit.

```

II.2. Cách sửa lỗi sinh mã trong Delphi

Khi bạn mới học sử dụng Delphi, nhất là khi mới làm quen với lập trình, thì việc phát sinh ra các lỗi (errors, bugs) do thao tác không đúng hoặc chưa hiểu đầy đủ kiến thức về OOP là dễ gặp phải. Ở đây, sẽ chỉ ra những sai sót thường gặp và cách sửa chúng.

a. Sai trong thao tác tạo thủ tục sự kiện: Nếu bạn làm đúng hướng dẫn trong phần "Xử lý sự kiện" (phần I.5) thì không bị sai. Tuy nhiên do chưa quen, bạn thường tự gõ vào phần cài đặt thủ tục sự kiện (TTSK) mà không biết hoặc quên đưa vào phần khai báo nguyên mẫu (Prototype), gồm từ khóa **procedure** Tên_TTSK([các_tham_số]); của thủ tục sự kiện.

Ví dụ như đoạn code sau:

```

type
    TForm1 = class(TForm)
        Button1: TButton;
        {----- thiếu khai báo nguyên mẫu này -----}
        procedure Button1Click(Sender: TObject);
        {----- -----}

```

```
private
  { Private declarations }
public
  { Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

// ----- phần bạn tự gõ vào -----
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Welcome to Delphi 7.0!... ');
end;
// -----
end. // Kết thúc Form Unit.
```

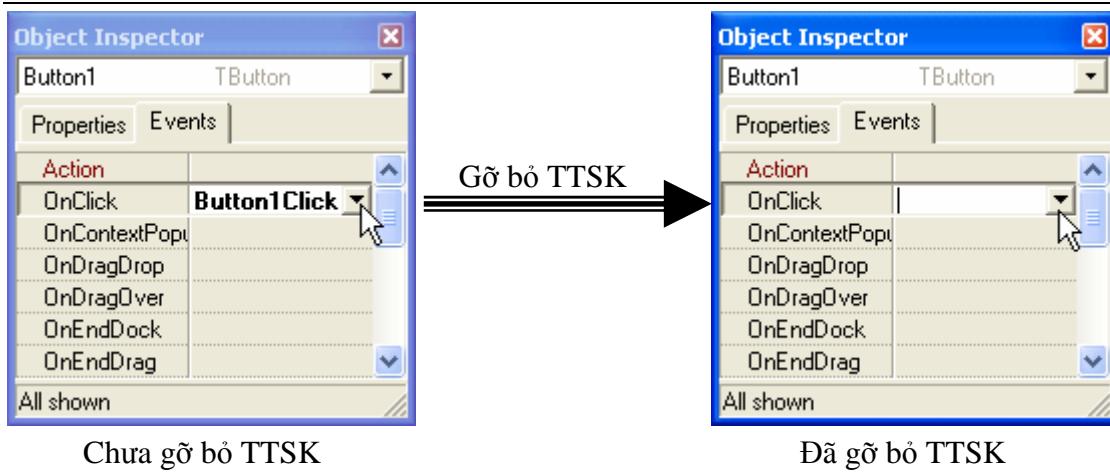
thì khi bạn biên dịch, Delphi sẽ thông báo lỗi như hình sau:



Hình 13: Báo lỗi do thiếu phần khai báo prototype

Khi xuất hiện hộp thông báo lỗi này, bạn có 3 chọn lựa:

- Chọn nút lệnh Yes, nếu muốn gỡ bỏ tên thủ tục sự kiện trong sự kiện tương ứng của đối tượng. Trong ví dụ này, bạn muốn gỡ bỏ TTSK Button1Click trong sự kiện OnClick của nút lệnh Button1.

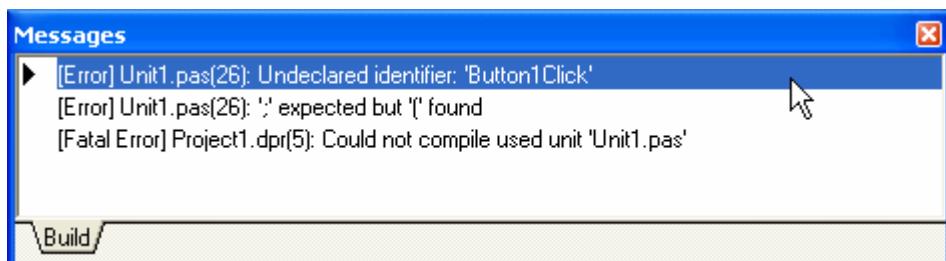


Hình 14: Minh họa tên TTSK **Button1Click** trong sự kiện **OnClick**

- Chọn nút lệnh No, nếu không gỡ bỏ tên thủ tục sự kiện trong sự kiện tương ứng của đối tượng. Trong ví dụ này, bạn vẫn muốn giữ lại TTSK Button1Click trong sự kiện OnClick của nút lệnh Button1.

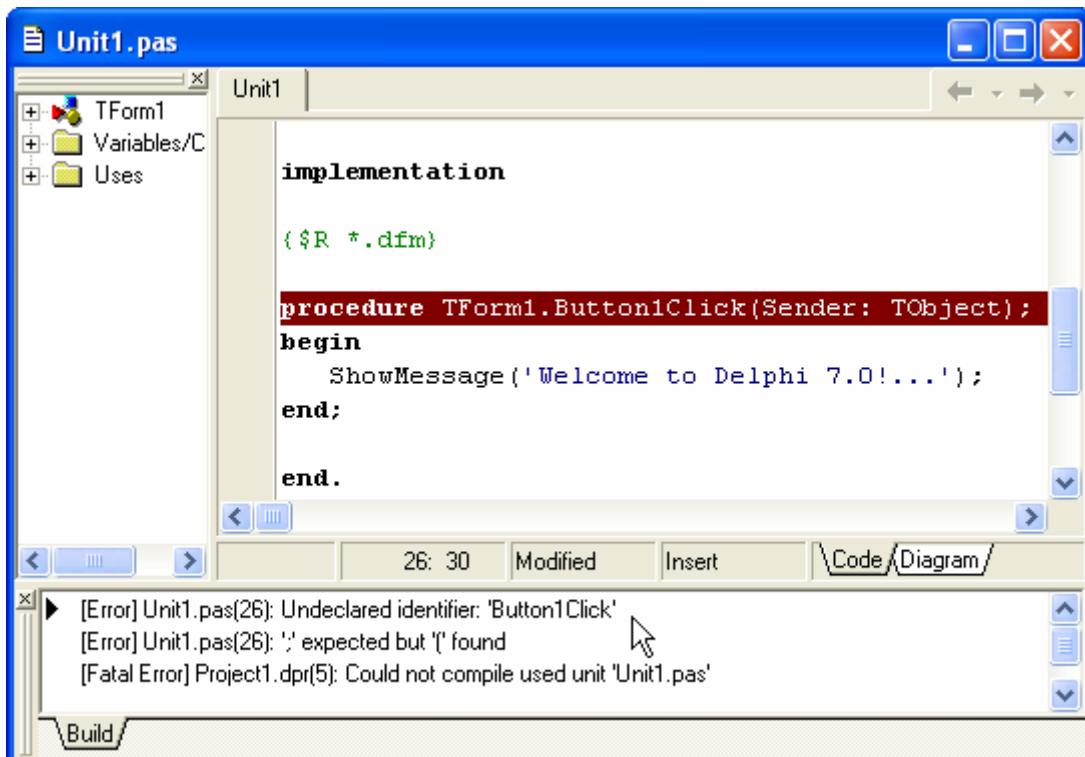
- Chọn nút lệnh Cancel thì hủy bỏ bảng thông báo lỗi này. Điều này cũng như chọn nút lệnh No.

Cả 3 chọn lựa trên vẫn không tự động sửa hết lỗi cho bạn. Nếu bạn nhấn phím Ctrl+F9 hoặc F9, thì Delphi sẽ nháy con trỏ (con nháy) đến đúng phần cài đặt TTSK và thông báo lỗi trong cửa sổ Messages ở phía dưới của sổ soạn thảo mã lệnh như hình sau:



Hình 15: Cửa sổ Messages

Hình ảnh phản ứng cửa sổ soạn thảo mã lệnh cùng với các cửa sổ khác.



Hình 16: Thông báo lỗi và con nháy nhảy đến TTSK bị lỗi.

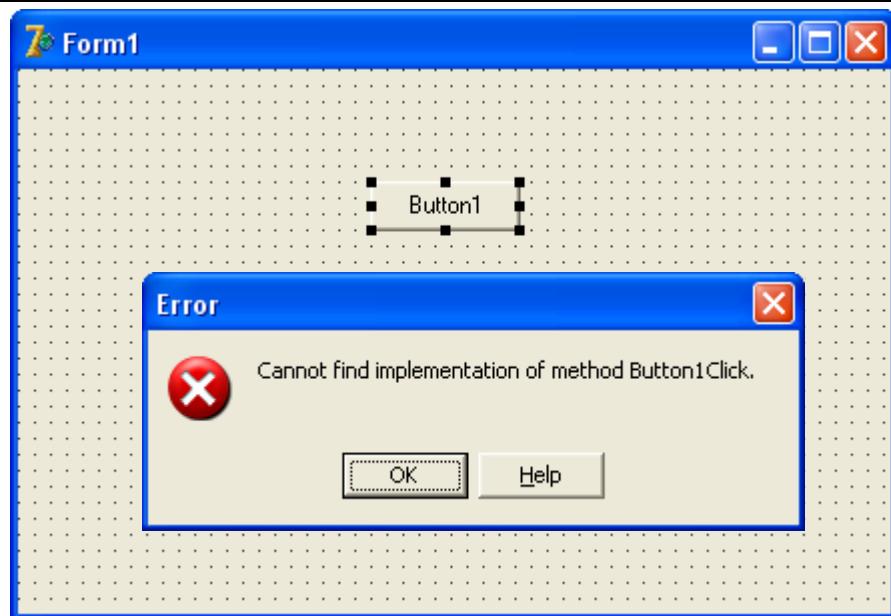
Để sửa lỗi này, bạn chuyển lên phần định nghĩa lớp TForm1, và thêm vào phần khai báo (Prototype) của thủ tục sự kiện này như sau:

```

type
    TForm1 = class(TForm)
        Button1: TButton;
        procedure Button1Click(Sender: TObject); //gõ thêm vào
    private
        { Private declarations }
    public
        { Public declarations }
    end;
    
```

Cần chú ý: Nếu bước trên bạn chọn Yes, thì sau khi hết lỗi bạn phải gắn tên TTSK vào sự kiện tương ứng cho đối tượng Button1; Còn nếu chọn No thì tên TTSK vẫn còn giữ nguyên (đã gắn sẵn) cho bạn.

- Một lỗi khác là bạn mở TTSK không được như khi bạn DClick lên nút lệnh Button1 thì có thông báo lỗi như sau:



Hình 17: Lỗi không thể tạo TTSK

trong trường hợp này, bạn chọn **OK**, sau đó nhấn phím chức năng F12 để chuyển vào cửa sổ soạn thảo mã lệnh để sửa lỗi.

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

end. //Nguyên nhân phát sinh lỗi, bạn xóa bỏ lệnh này đi.

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Welcome to Delphi 7.0! ');
end;
end.

```

- Một lỗi khác thường xảy ra là bạn xóa đi từ khóa **end.** trong tập tin unit thì sẽ nhận được thông báo:



Hình 18: Lỗi thiếu từ khóa **end.** trong unit form

Để sửa lỗi này, bạn nhấn phím chức năng F12 để chuyển sang cửa sổ mã lệnh, rồi thêm từ khóa **end.** vào cuối tập tin unit.

* **Tóm lại:** Có rất nhiều lỗi khác nhau phát sinh nếu bạn thao tác không chính xác. Cần phải chú ý 2 điểm:

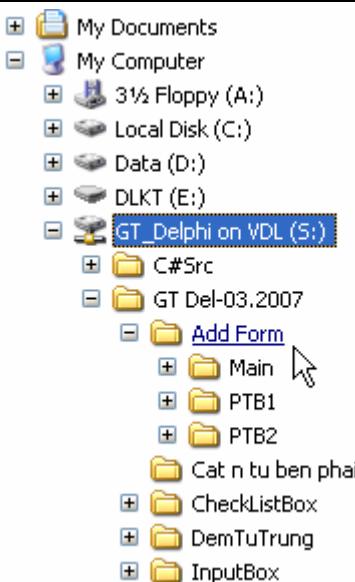
- Thao tác đúng theo sự hướng dẫn đã trình bày trong giáo trình.
- Một TTSK phải đầy đủ 2 thành phần:
 - + Phần khai báo tên thủ tục (Prototype) trong phần định nghĩa lớp của Form.
 - + Phần cài đặt đầy đủ TTSK sau phần cài đặt (Implementation).

II.3. Thêm tập tin unit vào dự án

Việc này giúp cho bạn, trưởng nhóm lập trình viên hay chủ dự án (group leader, leader team) có thể ghép nhiều form khác nhau ở những project khác nhau của những lập trình viên khác nhau viết, thành một dự án (thành phẩm) cuối cùng.

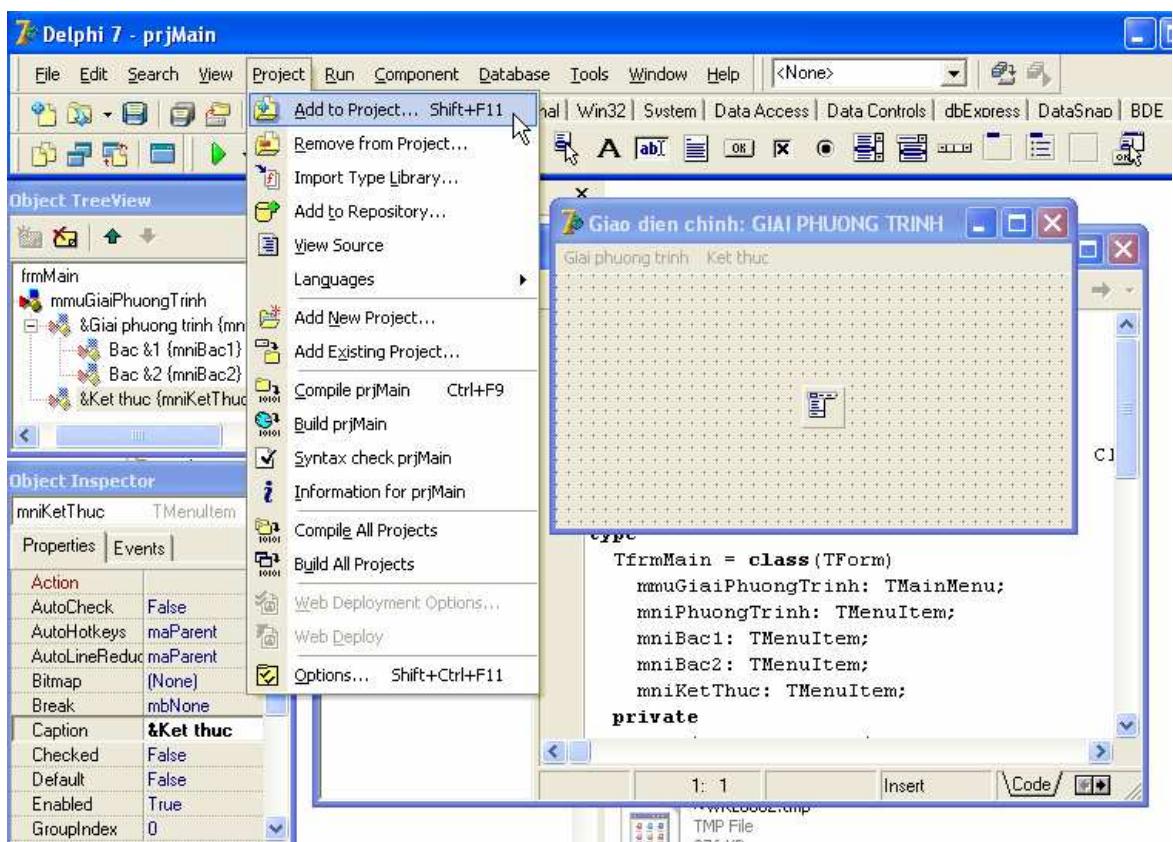
Ví dụ: Bạn thử tưởng tượng sự phân công công việc của 1 sản phẩm "Giải Phương Trình" có 2 phương trình bậc nhất và bậc hai. Trưởng nhóm, anh A, phân công cho lập trình viên B viết chương trình giải phương trình bậc nhất (tên form: frmPTB1, unit file: untPTB1.pas, project file: prjPTB1.dpr) và lập trình viên C viết chương trình bậc hai (tên form: frmPTB2, unit file: untPTB2.pas, project file: prjPTB2.dpr). Công việc của anh A là viết giao diện (Interface) cho gói sản phẩm này (tên form: frmMain, unit file: untMain.pas, project file: prjMain.dpr). Sau khi phần chương trình viết riêng biệt của mỗi anh A, B, C đã xong, thì anh A sẽ ghép thêm frmPTB1, frmPTB2 vào dự án của anh ta trong đó frmMain là giao diện của chương trình.

Tổ chức của sản phẩm "Giải Phương Trình" được đặt trong 3 thư mục con của thư mục S:\GT Del-03.2007\Add Form như hình sau:



Hình 19: Cấu trúc thư mục của sản phẩm "Giải Phương Trình"

Để làm được điều này, từ dự án đang mở của anh A, dùng lệnh **Project/ Add To Project... Shift+F11** để thêm frmPTB1, untPTB1.pas vào dự án prjMain.dpr như hình sau:



Hình 20: Thêm unit form untPTB1.pas vào dự án của anh A

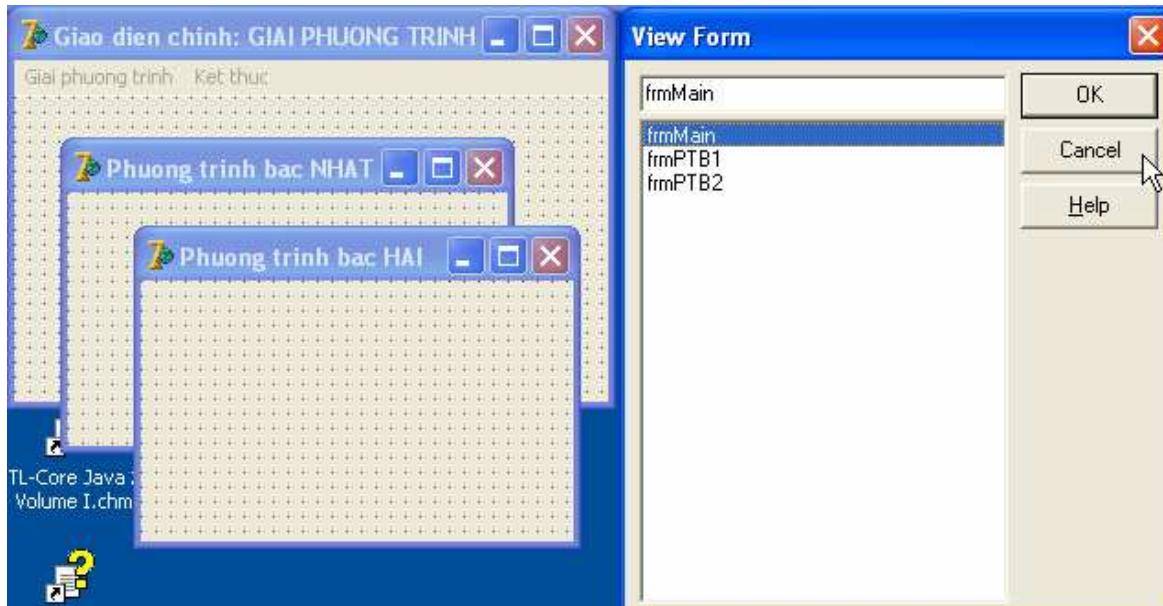
tiếp đến, bạn chọn tập tin unit form untPTB1.pas trong thư mục PTB1 (dự án của anh B) và nhấn nút lệnh Open để thêm vào.



Hình 21: Cửa sổ thêm untPTB1.pas vào dự án chính.

Hoàn toàn tương tự, bạn thêm vào unit form untPTB2.pas trong thư mục PTB2 (dự án của anh C).

Và như vậy trong sản phẩm của anh A bây giờ là 1 dự án có 3 form, như hình sau:



Hình 22: Dự án anh A đã chứa đầy đủ 3 form.

Trong tập tin dự án (Project file: prjMain.dpr) có sự tham chiếu (Reference/link) tới các tập tin unit form trong các folder của anh B và anh C như hình sau:

```

program prjMain;

uses
  Forms,
  untMain in 'untMain.pas' (frmMain),
  UntPTB1 in '..\PTB1\UntPTB1.pas' (frmPTB1),
  untPTB2 in '..\PTB2\untPTB2.pas' (frmPTB2);

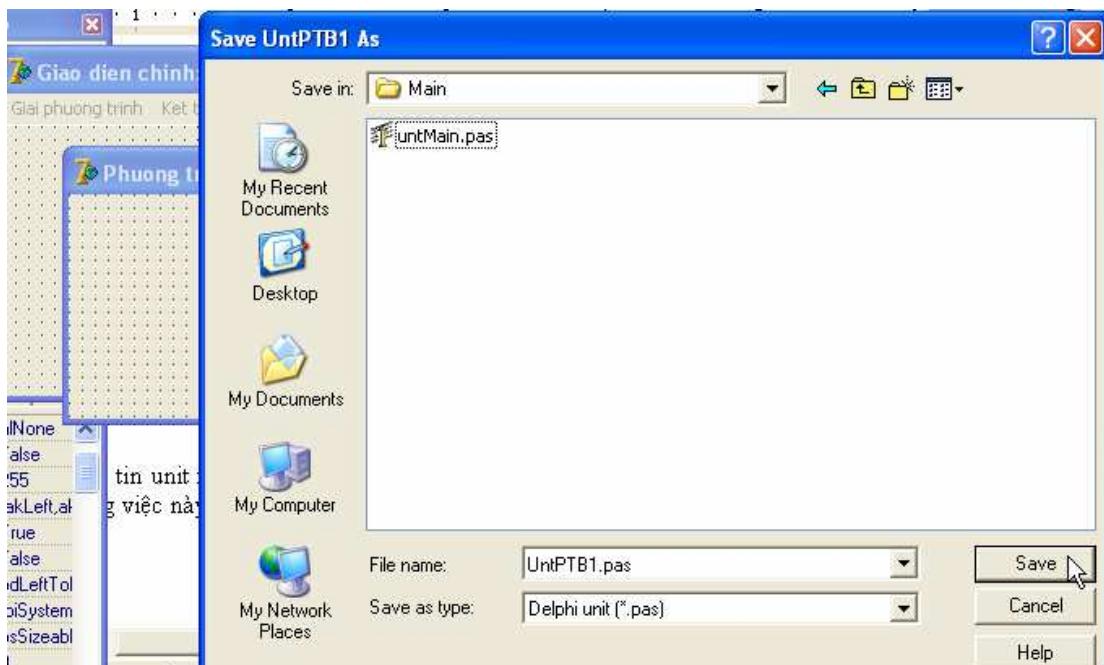
  {$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmMain, frmMain);
  Application.CreateForm(TfrmPTB1, frmPTB1);
  Application.CreateForm(TfrmPTB2, frmPTB2);
  Application.Run;
end.

```

Hình 23: Sự tham chiếu tới các unit form: untPTB1.pas và untPTB2.pas

Tới đây, câu hỏi được đặt ra là trong thư mục của anh A đã chứa đầy đủ các tập tin của sản phẩm chưa? Liệu bạn có thể xóa bỏ đi 2 thư mục PTB1 và PTB2 không? Câu trả lời là không. Một yêu cầu được đặt ra là anh A (trưởng nhóm) muốn tất cả các tập tin chỉ nằm trong 1 folder của anh ta (thư mục S:\GT Del-03.2007\Add Form\Main) thì làm như thế nào? Câu trả lời là anh A sẽ lưu lại unit file: untPTB1.pas (tương tự cho untPTB2.pas) vào thư mục Main bằng lệnh **File/Save As** như sau:



Hình 24: Lưu lại untPTB1.pas vào thư mục của anh A

Xem lại tập tin dự án, bạn thấy các tham chiếu tới các unit file đã nằm trong thư mục hiện hành (của anh A):

```

untMain
uses
  Forms,
  untMain in 'untMain.pas' (frmMain),
  UntPTB1 in 'UntPTB1.pas' (frmPTB1),
  untPTB2 in 'untPTB2.pas' (frmPTB2);

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TfrmMain, frmMain);
  Application.CreateForm(TfrmPTB1, frmPTB1);
  Application.CreateForm(TfrmPTB2, frmPTB2);
  Application.Run;
end.

```

Hình 25: Các tập tin unit form đã ở trong cùng 1 thư mục anh A

II.4. Viết lệnh sau khi thêm unit form vào dự án

Bây giờ, từ main form (frmMain), anh A muốn hiển thị (Show) form của anh B hoặc anh C thì sao? Chẳng hạn khi Click vào mục **Bac 1** trong thực đơn **Giai phuong trinh** để hiển thị form PTB1 lên như hình sau:



Hình 26: Hiển thị form "Phuong trinh bac nhat"

Để làm được thì anh A sẽ gõ thêm câu lệnh **frmPTB1.Show;** để hiển thị frmPTB1:

The screenshot shows the Delphi IDE interface with the file `untMain.pas` open. In the code editor, there is a code completion dropdown menu open over the word `Show`. The dropdown contains several suggestions, including `frmPTB1`. Below the code editor, a status bar displays the message "[Pascal Error] untMain.pas[1]: Unable to invoke Code Completion due to errors in source code".

```

implementation

( $R *.dfm)

procedure TfrmMain.mniBac1Click(Sender: TObject);
begin
    frmPTB1.      //Anh A muon lay phuong thuc Show
end;

end.

```

Hình 27: Không xuất hiện trình Code Completion

Tuy nhiên, trình hỗ trợ mã lệnh của Delphi (**Code Completion**) không tự động xuất hiện để anh A chọn phương thức `Show` mà lại xuất hiện thông báo lỗi trong cửa sổ Messages như sau:

"**[Pascal error] untMain.pas(1): Unable to invoke Code Completion due to errors in source code**".

Thông báo lỗi này phát sinh là do trong tập tin `untMain.Pas`, bạn **chưa** khai báo sử dụng unit `untPTB1` thông qua từ khóa `uses`

The screenshot shows the Delphi IDE interface with the file `untMain.pas` open. In the code editor, there is a code completion dropdown menu open over the word `Show`. The dropdown contains several suggestions, including `frmPTB1`. Below the code editor, a status bar displays the message "[Pascal Error] untMain.pas[1]: Unable to invoke Code Completion due to errors in source code".

```

unit untMain;

interface

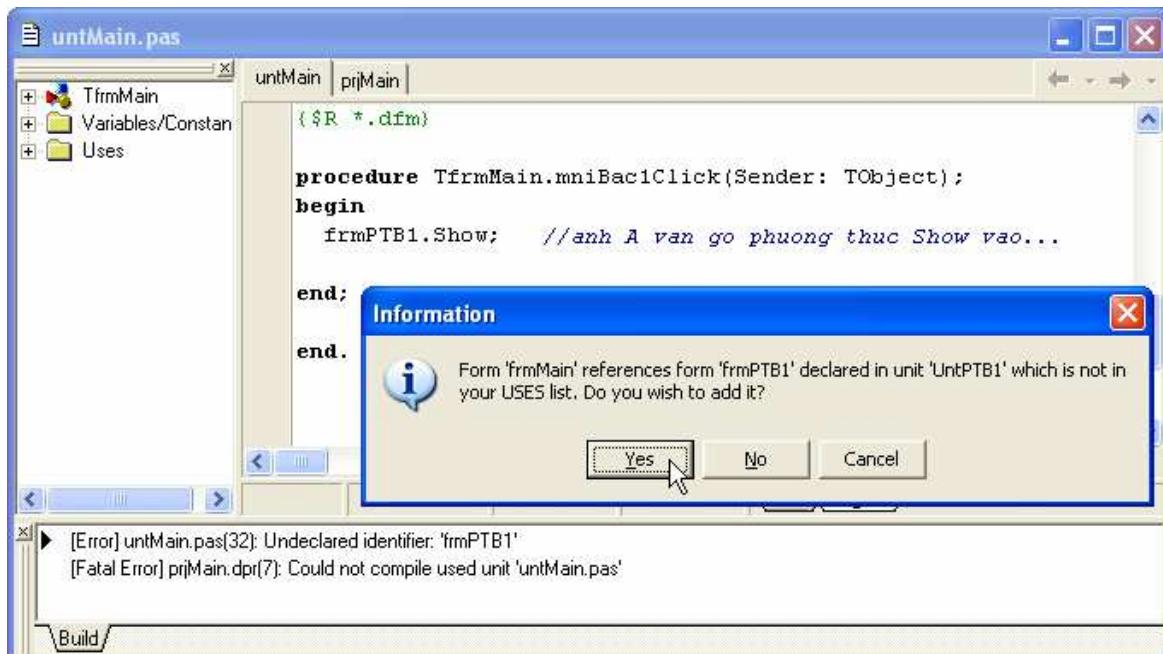
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Menus;

```

Hình 28: Chưa có khai báo unit `untPTB1` và `untPTB2`

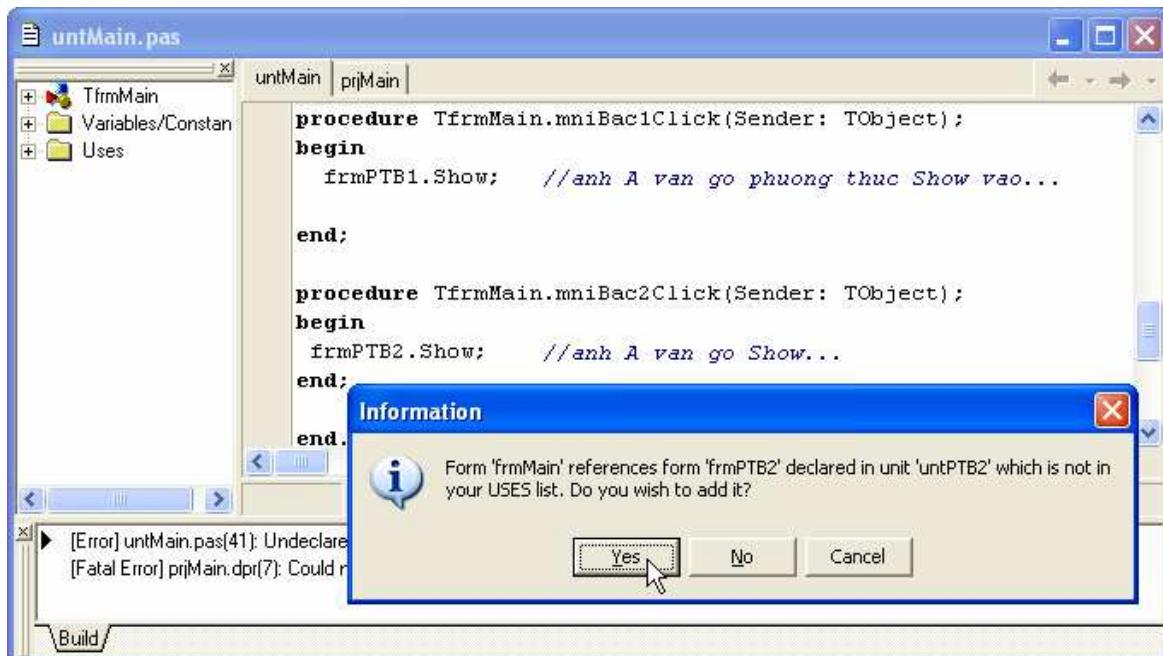
Có 02 cách để bạn sửa lỗi này:

- **Cách 1:** Nếu bạn biết chính xác tên phương thức mà bạn cần, thì bạn cứ việc gõ tiếp phương thức Show như sau: **frmPTB1.Show;** (bắt chấp thông báo lỗi – không được hỗ trợ bởi menu Code Completion). Sau đó, bạn biên dịch lại (nhấn tổ hợp phím Ctrl+F9), sẽ xuất hiện thông báo như sau:



Hình 29: Thông báo thiếu khai báo unit untPTB1

Bạn Click nút lệnh **Yes** để Delphi tự động khai báo thêm untPTB1 vào tập tin unit untMain. Tương tự cho unit frmPTB2, ta cũng Click nút lệnh **Yes** để thêm vào.



Hình 30: Thông báo thiếu khai báo unit untPTB2

Sau 2 bước trên, đoạn khai báo các unit sẽ như sau:

```

unit untMain;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus;

type
  TfrmMain = class(TForm)
    mmuGiaiPhuongTrinh: TMainMenu;
    mniPhuongTrinh: TMenuItem;
    mniBac1: TMenuItem;
    mniBac2: TMenuItem;
    mniKetThuc: TMenuItem;
    procedure mniBac1Click(Sender: TObject);
    procedure mniBac2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;

implementation

uses UntPTB1, untPTB2;

```

Hình 31: Đã khai báo xong 2 unit: untPTB1, untPTB2

- Cách 2: Vì trong unit untMain, anh A sử dụng frmPTB1 và frmPTB2. Do vậy, trước khi anh A viết code cho 2 TTSK trên, anh A phải gõ thêm khai báo việc sử dụng 2 unit: untPTB1 và untPTB2, rồi mới viết lệnh frmPTB1.Show; vào. Cách này tốt và chính xác hơn cách 1 vì có trình Code Completion hỗ trợ cho bạn khi viết lệnh như hình sau:

```

unit untMain;
interface
uses
  UntPTB1, untPTB2;      // Anh A moi khai bao them 2 unit vao
  {$R *.dfm}

procedure TfrmMain.mniBac1Click(Sender: TObject);
begin
  frmPTB1.sh| //anh A co trinh Code Completion ho tro....
end;

```

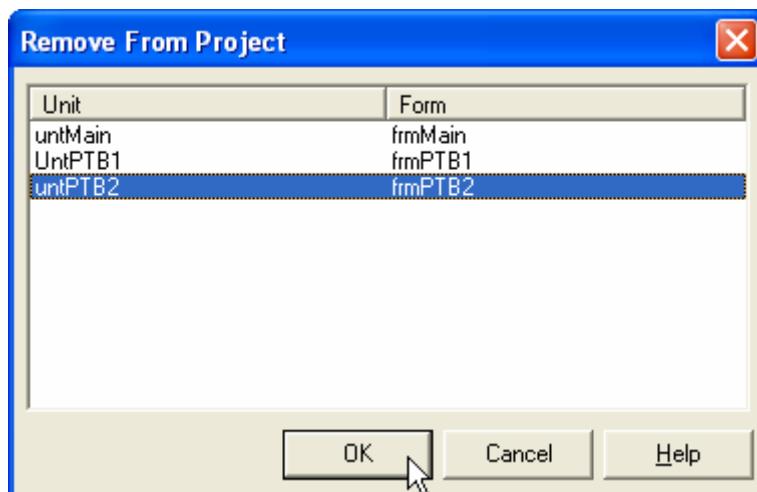
Hình 32: Thêm khai báo unit trước và có trình Code Completion

Chú ý: Bạn có thể sử dụng lệnh **File/Use Unit...** để thêm unit vào dự án hiện hành.

II.5. Gỡ bỏ tập tin unit form ra khỏi dự án

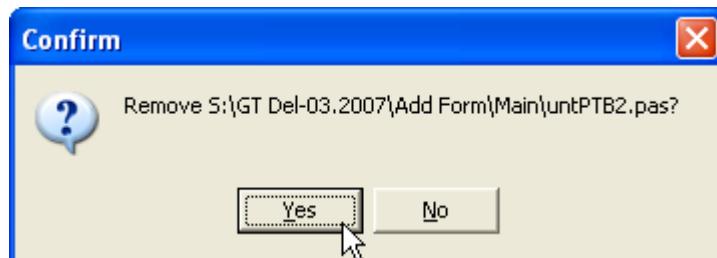
Công việc này giúp bạn loại bỏ ra khỏi dự án (Project) những form (unit file) không cần thiết, hoặc bị sai. Để gỡ bỏ, bạn chọn chức năng **Project/Remove from Project...**

Ví dụ 8: Anh A, trưởng nhóm, thấy phần chương trình của anh C – Phương trình bậc hai, viết còn chưa tốt (hay còn bugs). Anh A đề nghị anh C hoàn chỉnh và kiểm tra (testing) phần chương trình lại, sau đó gửi lại để anh A đóng gói thành phẩm. Như vậy, anh A tạm thời gỡ bỏ (remove) phần chương trình của anh C ra khỏi dự án hiện hành. Để làm việc đó, anh A ra lệnh Project/Remove from Project... thì Delphi sẽ xuất hiện hộp thoại "Remove From Project", rồi chọn unit untPTB2 và Click **OK** như hình sau:



Hình 33: Cửa sổ gỡ bỏ tập tin unit

Delphi yêu cầu bạn khẳng định lại công việc gỡ bỏ này ra khỏi dự án, bạn chọn **Yes** để gỡ bỏ:



Hình 34: Hộp xác nhận việc gỡ bỏ

III. Biểu mẫu (TForm)

Form là một đối tượng chuyên dụng trong môi trường lập trình Windows. Trong biểu mẫu form này, bạn có thể đặt các đối tượng thuộc thành phần khác như TLabel, TEdit, TButton,... lên nó để thiết kế giao diện của chương trình hay còn được gọi là Form chính (Main form) hoặc các biểu mẫu phụ khác trong dự án.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng form, tên này phải có tính gợi nhớ để giúp bạn dễ quản lý và lập trình, nhất là khi bạn tham gia vào những dự án lớn. (Xem lại Bảng 1: Quy cách đặt tên các đối tượng).
AutoSize	Có giá trị True hoặc False: tự động hay không tự động thay đổi kích thước của form sao cho kích thước của nó vừa đủ để chứa các thành phần (component) đặt trong nó.
Caption	Chuỗi ký tự được hiển thị trên thanh tiêu đề (Title bar) của form.
BorderStyle	Các kiểu dáng hiển thị form: <ul style="list-style-type: none"> - bsDialog: là hộp hội thoại, không thay đổi kích thước được. - bsSingle: Đường viền khung được kẻ đường 1 nét và không thay đổi được kích thước của form. - bsNone: Không có đường viền khung và không thay đổi được kích thước của form. - bsSizeable: Dạng chuẩn/ mặc nhiên của form. Đây là dạng một cửa sổ rất phổ biến trên môi trường lập trình Windows. - bsToolWindow: Giống bsSingle, nhưng thanh tiêu đề nhỏ hẹp hơn. - bsSizeToolWin: Giống bsSizeable, nhưng thanh tiêu đề nhỏ hẹp hơn.
BorderIcons	Xác định trên thanh tiêu đề có hộp điều khiển hay còn gọi là hộp “Control box”, “Control menu” và “System menu” ở bên trái thanh tiêu đề và các nút điều khiển ở bên phải thanh tiêu đề: <ul style="list-style-type: none"> - biSystemMenu: Form có hộp điều khiển. - biMinimize : Form có nút thu nhỏ (Minimize) - biMaximize : Form có nút phóng to (Maximize), phục chế (Restore) - biHelp : Form có nút trợ giúp nếu bạn thiết lập giá trị của biMinimize = False và biMaximize = False.
Cursor	Các kiểu dáng hiển thị con trỏ (Mouse) trên form.
Form style	Các kiểu form: (MDI: Multi Document Interface) <ul style="list-style-type: none"> - fsMDIForm: MDIForm (Multi Document Interface) là form MDI cha của tất cả các form khác (nếu có) trong chương trình. Khi bạn đóng MDIForm, thì đồng nghĩa với việc đóng tất cả các form khác và kết thúc chương trình. - fsMDIChild: Đây là form MDI con. Trong chương trình, bạn có thể mở ra nhiều cửa sổ MDIChild. <p>Để dễ hiểu, bạn có thể xem MDIForm chính là cửa sổ ứng dụng phần mềm Microsoft Word, còn MDIChild chính là các cửa sổ tập tin như Document1, Document2,... được tạo ra thông qua lệnh File/New</p> <ul style="list-style-type: none"> - fsNormal: Là loại form thường dùng nhất và đây cũng là giá trị mặc định (Default) khi bạn thêm 1 form mới. Loại form này không có tính chất của loại form MDI. - fsStayOnTop: giúp cho form luôn nổi lên trên nhất, ngay cả khi có form khác đang hoạt động (Active).

Tên thuộc tính	Ý nghĩa
Color	Màu nền
Font	Xác định các thuộc tính về font như: kiểu chữ, màu, cỡ,...
Icon	Biểu tượng của chương trình dự án. Biểu tượng này được nạp vào thông qua cửa sổ Picture Editor.
MainMenu	Xác định thanh thực đơn chính cho form thông qua thuộc tính name của thành phần TMainMenu đã được đặt lên form.
PopupMenu	Xác định menu đối tượng cho form thông qua thuộc tính name của thành phần TPopupMenu đã đặt được lên form.
Position	Xác định vị trí hoặc kích thước của form khi nó xuất hiện trên màn hình khi chạy chương trình: <ul style="list-style-type: none"> - poDefault: do hệ thống của Delphi quyết định - poDefaultPosOnly: Delphi chỉ quyết định vị trí của form còn kích thước thì vẫn giữ nguyên như lúc thiết kế. - poDefaultSizeOnly: Delphi chỉ quyết định kích thước của form còn vị trí như lúc thiết kế. - poDesigned: như tại thời điểm thiết kế. - poDesktopCenter: Vị trí của form nằm trung tâm màn hình. - poScreenCenter: Giống như poDesktopCenter (không xét đến ứng dụng đa màn hình, multi-monitor). - poMainFormCenter: thường được thiết lập cho các form không phải là form chính (Main form). Các form được thiết lập theo giá trị này sẽ được hiển thị ở trung tâm form chính. - poOwnerFormCenter: thường được thiết lập cho các form không phải là form chính. Khi form được thiết lập giá trị này, thì nó xuất hiện ở trung tâm của form chủ của nó (Owner).
WindowMenu	Dành cho MDI form.
Visible	Có giá trị True hoặc False: Có xuất hiện hay không khi chạy chương trình.
WindowState	Các giá trị có ý nghĩa như sau: <ul style="list-style-type: none"> - wsNormal: Ở trạng thái thông thường. Cách thức hiển thị cửa sổ form sẽ giống như các giá trị đã được thiết lập trong thuộc tính BorderStyle. - wsMinimized: Cửa sổ form sẽ thu nhỏ về thanh tác vụ (Task bar). - wsMaximized: Cửa sổ form sẽ được phóng to toàn bộ màn hình. Hai chức năng wsMinimized và wsMaximized chỉ có hiệu lực khi thuộc tính BorderStyle có giá trị là bsNone, bsSingle, hoặc bsSizeable.

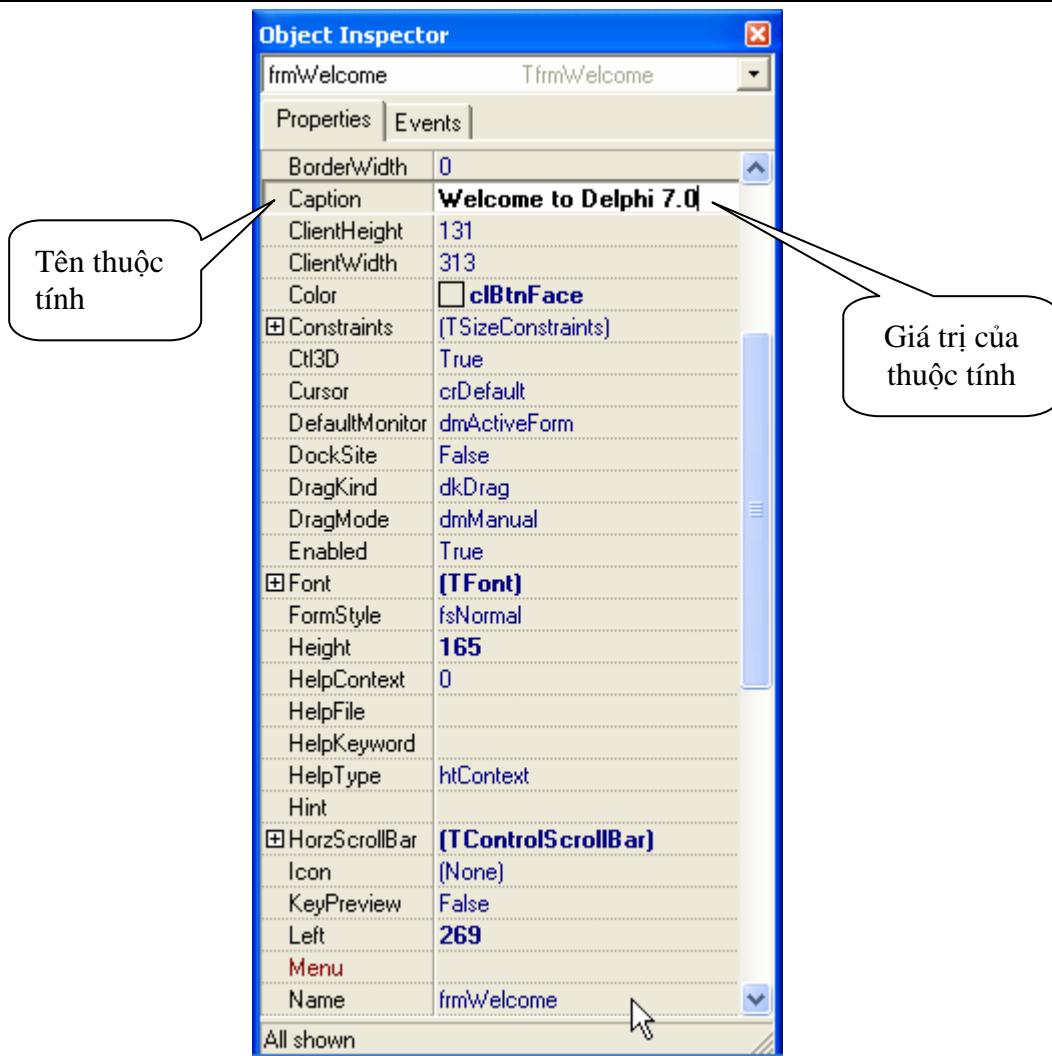
Bảng 2: Các thuộc tính của TForm.

* Một số sự kiện thường dùng:

Tên sự kiện	Sự kiện sẽ đáp ứng
ActiveControl	Xác định đối tượng của thành phần (Component) nào trên form nhận focus khi form hiển thị. Bạn xem chức năng này tương

Tên sự kiện	Sự kiện sẽ đáp ứng
	đương với việc đặt TabOrder của đối tượng này bằng 0.
ObjectMenuItem	Xác định MenuItem nào của MainMenu trở nên không kích hoạt được (Enabled = False) khi form được hiển thị. Chức năng này rất phổ biến trên Windows ví dụ như việc sao chép (Copy) và dán (Paste) chẳng hạn: Khi chưa thực hiện chức năng Copy thì chức năng Paste không kích hoạt được.
OnActivate	Khi form trở nên hoạt động (hiện hành)
OnCreate	Khi form được tạo ra. Đây là sự kiện được thực hiện trước hết mọi sự kiện của form.
OnShow	Khi form được hiển thị, dĩ nhiên là thuộc tính Visible của form là True.
OnClick	Khi người sử dụng nhấp chuột trái (Click) vào đối tượng
OnContextPopup	Khi người sử dụng nhấp chuột phải (RClick) vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng (PopupMenu)
OnDblClick	Khi người sử dụng nhấp đúp (Double Click) vào đối tượng
OnKeyDown	Khi người sử dụng nhấn phím bất kỳ.
OnKeyPress	Khi người sử dụng nhấn phím bất kỳ nhưng không phải là các phím điều khiển (ví dụ như: F1, Ctrl, Shift,...)
OnKeyUp	Khi một hoặc tổ hợp phím bất kỳ được thả ra.
OnMouseDown	Khi người sử dụng Click, RClick, mouse giũa (hay Scroll mouse).
OnMouseMove	Khi người sử dụng di chuyển mouse trên form.
OnMouseUp	Khi người sử dụng thả bất kỳ phím nào của mouse.
OnMouseWheel	Khi người sử dụng cuộn mouse giũa theo hướng lên hoặc xuống.
OnMouseWheelDown	Khi người sử dụng cuộn mouse giũa theo hướng xuống.
OnMouseWheelUp	Khi người sử dụng cuộn mouse giũa theo hướng lên.

Bảng 3: Các sự kiện của TForm.

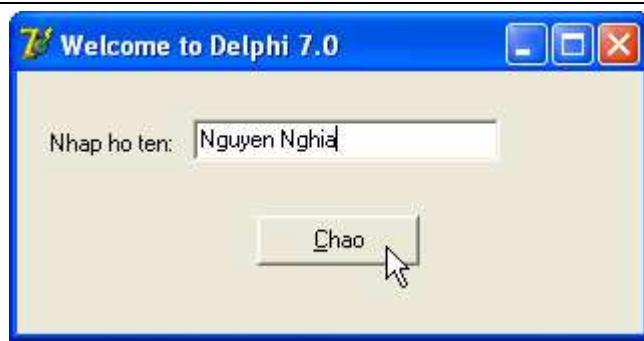


Hình 35: Thiết lập giá trị các thuộc tính của đối tượng frmWelcome

- **Bước 5:** Lưu dự án, biên dịch và chạy chương trình:

- + Chọn chức năng **File/Save All** để lưu dự án vào thư mục S:\Welcome
- + Nhấn Ctrl+F9 để biên dịch
- + Nếu biên dịch thành công, nhấn phím chức năng F9 hoặc chức năng Run/Run hoặc Click nút để chạy chương trình.
- + Kết thúc chương trình, bạn sử dụng một trong các cách:
 - Click nút Close trên Form.
 - Chọn chức năng Run/Program Reset
 - Sử dụng tổ hợp phím Ctrl+F2
 - Click chọn nút công cụ

Ta có kết quả chương trình như sau:



Hình 36: Form nhập liệu

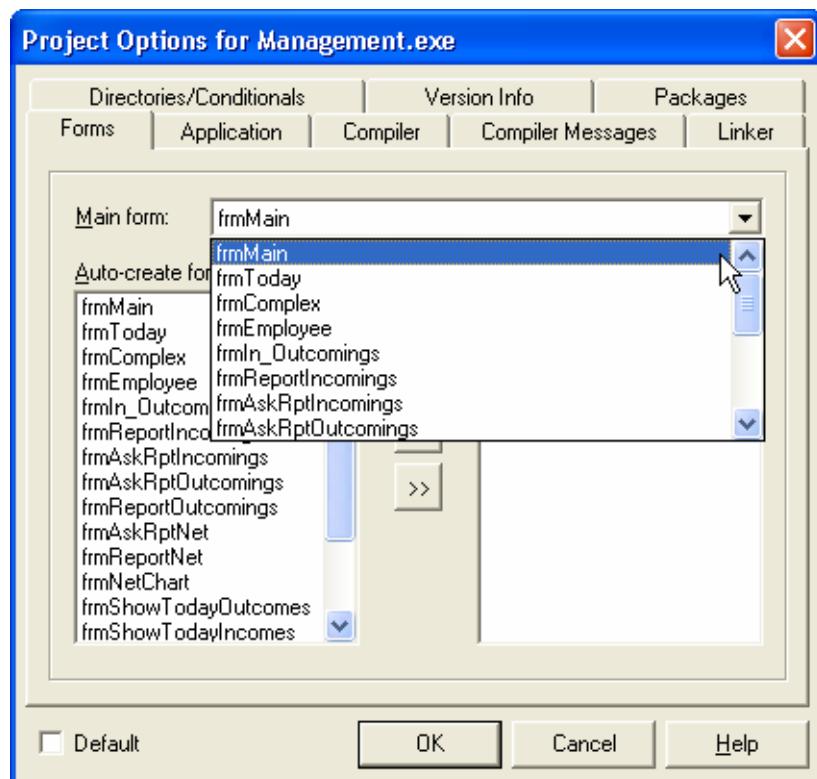
Nhập họ tên vào, chẳng hạn “Nguyen Nghia” rồi Click vào nút lệnh **Chao**, chương trình sẽ ẩn form chính và xuất hiện hộp thông báo “Welcome”:



Hình 37: Hộp thông báo Hello

Bạn click nút lệnh **OK** để đóng hộp thoại này và hiển thị lại form chính.

Trong dự án, bạn có thể có nhiều form, để thêm form vào dự án, bạn sử dụng chức năng: **File/New/Form**. Để chọn 1 form nào đó trong số các form làm form chính, bạn sử dụng chức năng **Project /Options/Forms** như hình sau:

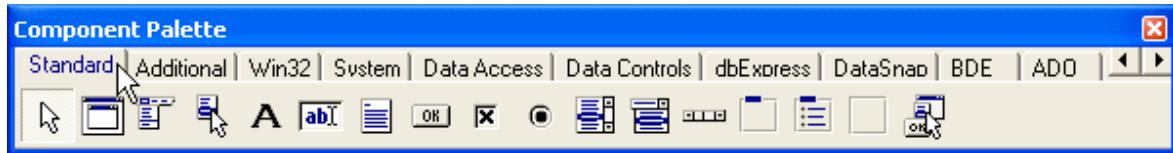


Hình 38: Chọn frmMain làm form chính.

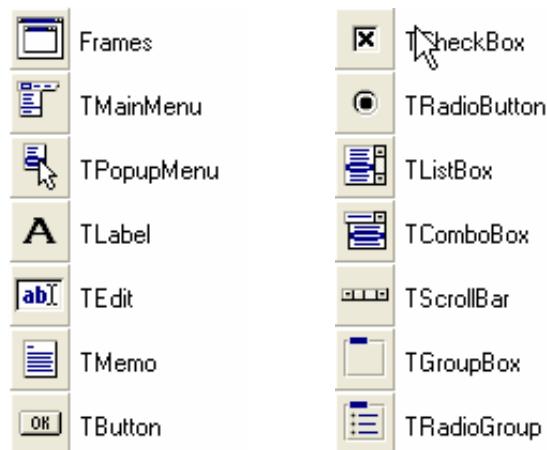
Để các đối tượng trên form làm việc, tương tác với nhau đúng như ý muốn, người lập trình cần phải thiết lập, đọc các thuộc tính cũng như sử dụng các phương thức, thủ tục sự kiện của chúng một cách hợp lý sao cho chúng gắn kết với nhau thành một khối thống nhất. Để làm được điều này, **người viết chương trình cần phải có ý tưởng rành mạch về dự án để thống nhất từ quá trình thiết kế (designing) đến viết mã (coding).**

IV. Các thành phần (Component) giao diện phổ biến

Các thành phần (điều khiển) trong Delphi rất phong phú và chúng được đặt trong các thẻ (tab) của bảng chứa các thành phần của Delphi (Component Palette) như thẻ Standard, Addition, Win32,... Trong giáo trình này, các bạn sẽ được giới thiệu nhiều đối tượng khác nhau phù hợp với trình tự của bài học. **Tại thời điểm thiết kế, ta Click vào thành phần cần sử dụng rồi drag and drop để đặt đối tượng đó lên form.**



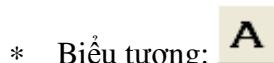
Hình 39: Bảng các thành phần



Hình 40: Các thành phần trên thẻ Standard

IV.1. Nhãn (TLabel)

Là một thành phần dùng để hiển thị một hoặc nhiều dòng **văn bản tĩnh** trong nó.



* Biểu tượng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng nhãn (Xem lại Bảng 1: Quy cách đặt tên các đối tượng)
AutoSize	Có giá trị True hoặc False: tự động hay không tự động thay đổi kích thước của Label khi chương trình chạy so với thiết kế ban đầu.
Caption	Chuỗi ký tự được hiển thị trong nó

Tên thuộc tính	Ý nghĩa
Visible	Có giá trị True hoặc False: có hiển thị hay không khi chạy chương trình.
Color	Màu nền
Font	Xác định các thuộc tính về font như: kiểu chữ, màu, cỡ,...
Transparent	Màu nền của nhãn là trong suốt/không màu.
WordWrap	Có giá trị True hoặc False: cho phép hiển thị nhiều dòng hay chỉ 1 dòng trong đối tượng Label.

Bảng 4: Các thuộc tính của TLabel

* **Một số sự kiện thường dùng:**

Tên sự kiện	Sự kiện sẽ đáp ứng
OnClick	Khi người sử dụng nhấp chuột trái (Click) vào đối tượng
OnContextPopup	Khi người sử dụng nhấp chuột phải (RClick) vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng (PopupMenu)
OnDblClick	Khi người sử dụng nhấp đúp (Double Click) vào đối tượng

Bảng 5: Các sự kiện của TLabel

- **Một số hàm được sử dụng trong ví dụ:**

+ Hàm trả về thời gian hiện hành của hệ thống máy tính:

function Time: TDateTime;

+ Hàm trả về ngày tháng năm hiện hành của hệ thống máy tính:

function Date: TDateTime;

+ Hàm chuyển đổi kiểu thời gian (time) sang kiểu chuỗi (string)

function TimeToStr(Time: TDateTime): string; overload;

+ Hàm chuyển đổi kiểu ngày tháng năm (date) sang kiểu chuỗi (string)

function DateToStr(Date: TDateTime): string; overload;

Ví dụ 9: Xem ngày giờ hiện hành của hệ thống.

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong Label
Position	poDesktopCenter

Bảng 6: Thiết lập các thuộc tính cho đối tượng Form.

- Đặt đối tượng Label lên form và thiết lập thuộc tính:

Thuộc tính	Giá trị
Name	lblDateTime
Caption	(để rỗng)
Align	alClient
Color	clSkyBlue
BorderStyle	bsDialog

Bảng 7: Thiết lập các thuộc tính cho Label

- Từ cửa sổ Object Inspector, click vào nút liệt kê thả để chọn đối tượng **frmMain**. Click chọn tab Events, DClick vào cột bên phải của sự kiện **OnCreate** tương ứng để sinh ra thủ tục sự kiện **FormCreate**, và gõ thêm các lệnh vào trong thân thủ tục sự kiện. Đoạn mã lệnh của chương trình hoàn chỉnh như sau:

```

unit untDateTime; //tên unit file: untDateTime.pas
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmMain = class(TForm)
    lblDateTime: TLabel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  lblDateTime.Font.Color := clPurple;
  lblDateTime.Font.Size:=15;
  lblDateTime.Caption:=' - The current time is ' +
                      TimeToStr(time);
  lblDateTime.Caption:=lblDateTime.Caption + Chr(10)+' - Today is
                      ' + DateToStr(Date);
end;
end.
```

- Lưu dự án vào thư mục S:\ViDuLabel
- Biên dịch dự án: Ctrl + F9
- Chạy chương trình: F9
- Kết quả chương trình như hình sau:



Hình 41: Ngày giờ hiện hành của hệ thống

IV.2. Hộp văn bản (TEdit)

Là một component chuẩn để nhập hay xuất một chuỗi ký tự trong nó khi chạy chương trình. Chú ý hộp văn bản chỉ nhập hoặc xuất được 1 dòng duy nhất.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho hộp văn bản.
AutoSize	Có giá trị True hoặc False: tự động hay không tự động thay đổi kích thước hộp văn bản khi chương trình chạy so với thiết kế ban đầu.
Text	Chuỗi ký tự trong hộp văn bản
CharCase	Xác định kiểu chữ in hoa (ecUpperCase), chữ nhỏ (ecLowerCase) hoặc kiểu chữ do người sử dụng nhập (ecNormal)
Font	Xác định các thuộc tính về font như: kiểu chữ, màu, cỡ,...
Visible	Có giá trị True hoặc False: có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
ReadOnly	Có giá trị True hoặc False: Dữ liệu trong nó là chỉ đọc hay có thể thay đổi.

Bảng 8: Các thuộc tính của TEdit

* Một số sự kiện thường dùng:

Tên sự kiện	Sự kiện sẽ đáp ứng
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng

Tên sự kiện	Sự kiện sẽ đáp ứng
OnEnter	Khi đối tượng nhận tiêu điểm (Got Focus) hay trở nên hoạt động (Active)
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 9: Các sự kiện của TEdit

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Show;	Thủ tục hiển thị đối tượng khi chạy chương trình.
Hide;	Thủ tục ẩn đối tượng khi chạy chương trình.
Clear;	Thủ tục xóa hết giá trị trong TEdit

Bảng 10: Các phương thức của TEdit

Ta có thể sử dụng các hàm chuẩn trong Delphi để nhập hoặc xuất dữ liệu thay cho TEdit.

- Hàm InputBox:

Cú pháp:

function InputBox(const ACaption, APrompt, ADefault: string): string;

Trong đó:

ACaption : là một chuỗi ký tự để làm tiêu đề của hộp hội thoại nhập.

APrompt : là một chuỗi ký tự để làm câu nhắc cho người nhập liệu

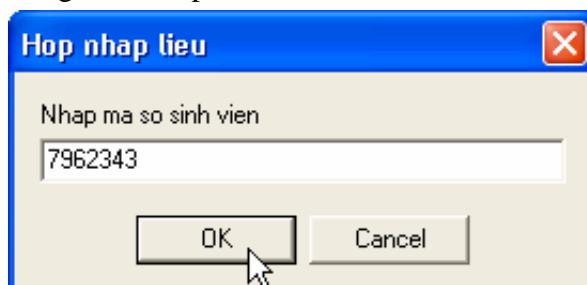
ADefault : là một chuỗi ký tự để làm giá trị mặc định ban đầu

Ý nghĩa: Dùng để nhập dữ liệu vào thông qua hộp thoại, kết quả trả về của hàm là kiểu chuỗi ký tự (String)

Ví dụ 10: Nhập mã số sinh viên từ hộp thoại như sau:

mssv := InputBox('Hop nhap lieu', 'Nhập mã số sinh viên','7962343');

Với biến mssv kiểu string, ta có hộp hội thoại như sau:



Hình 42: Hộp nhập liệu bằng hàm InputBox

- Một số khái niệm và hàm được sử dụng trong ví dụ:

+ **Biến riêng** (cục bộ): là biến chỉ có phạm vi hoạt động và tồn tại trong thủ tục sự kiện của nó.

+ Hàm chuyển đổi kiểu số dấu chấm động (Float) sang kiểu chuỗi:

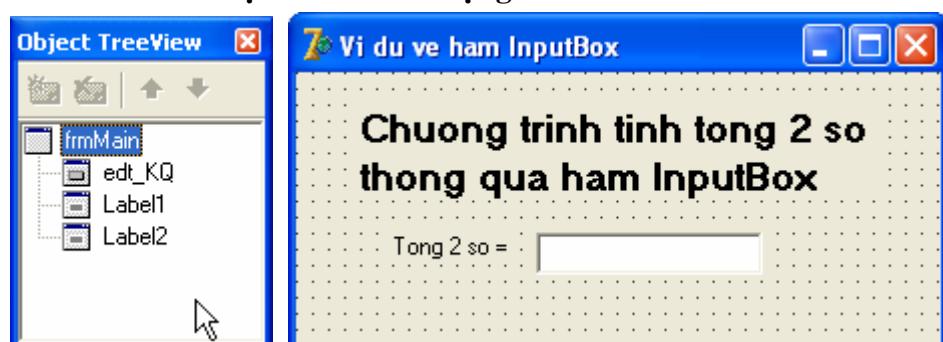
function FloatToStr(Value: Extended): **string; overload;**

+ Hàm chuyển đổi kiểu chuỗi sang kiểu dấu chấm động (Float):

function StrToFloat(const S: **string**): extended;

Ví dụ 11: Viết chương trình nhập 2 số bất kỳ bằng hàm InputBox thông qua sự kiện OnCreate của Form và cho biết kết quả của chúng thông qua đối tượng của thành phần TEdit.

- Thiết kế form và đặt tên các đối tượng như sau:



Hình 43: Thiết kế form và đặt tên các đối tượng

- Đoạn mã lệnh của chương trình:

```

unit untTinhTong;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmMain = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    edt_KQ: TEdit;
    procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
var
  frmMain: TfrmMain;

```

implementation

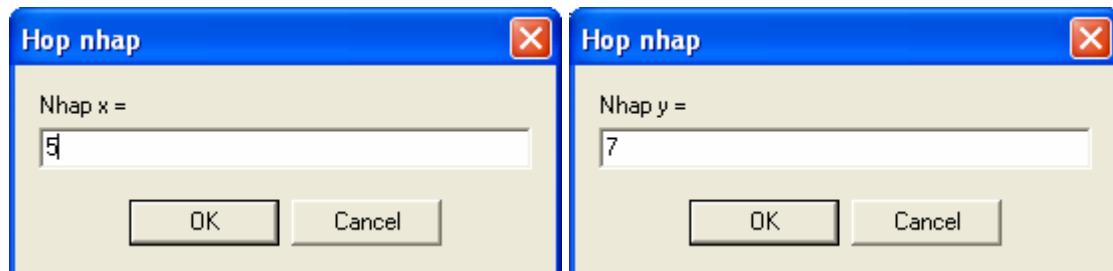
```

{$R *.dfm}

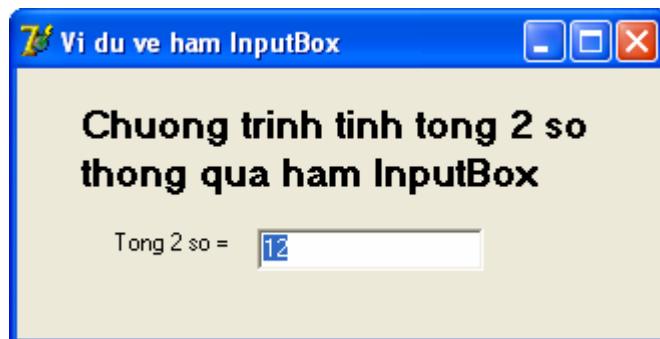
procedure TfrmMain.FormCreate(Sender: TObject);
var x, y: Extended;
begin
  x := StrToFloat(InputBox('Hop nhap', 'Nhập x =', '')); 
  y := StrToFloat(InputBox('Hop nhap', 'Nhập y =', '')); 
  edt_KQ.Text := FloatToStr(x+y);
end;
end.

```

- Lưu dự án vào thư mục S:\ViDuInputBox, biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 44: Nhập giá trị 2 biến x, y



Hình 45: Kết quả của chương trình

- **Hàm InputQuery:**

Cú pháp:

```
function InputQuery(const ACaption, APrompt: string; var Value: string);
  boolean;
```

Trong đó:

ACaption: là một **hằng chuỗi** ký tự để làm tiêu đề của hộp hội thoại nhập.

APrompt: là một chuỗi ký tự để làm câu nhắc cho người nhập liệu

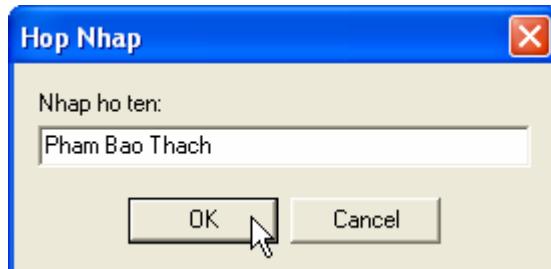
Value: là một **tham biến** nhận giá trị nhập của hàm.

Ý nghĩa: Dùng để nhập dữ liệu vào thông qua hộp thoại, kết quả trả về của hàm là kiểu boolean, và giá trị được nhập vào trong hộp thoại chứa trong tham biến Value nếu ta Click nút OK.

Ví dụ 12: Nhập họ tên từ hộp thoại như sau:

```
m_OK := InputQuery('Hop nhap lieu ', 'Nhap ho ten', m_Hoten);
```

Với biến m_OK kiểu **boolean**, m_Hoten kiểu **string**, ta có hộp hội thoại như sau:



Hình 46: Hộp nhập liệu bằng hàm InputQuery

Nếu ta click nút lệnh OK, thì m_OK = True còn m_Hoten = 'Pham Bao Thach', còn click nút Cancel thì m_OK = False.

- Thủ tục ShowMessage:

Cú pháp: **procedure ShowMessage(const Msg: string);**

Ý nghĩa: dùng để hiển thị thông điệp thông qua hàng chuỗi ký tự Msg

Ví dụ 13: Câu lệnh ShowMessage('Chao mung ban den voi Delphi phien ban 7.0!...'); để hiển thị thông báo: “Chào mừng bạn đến với Delphi phiên bản 7.0!...”



Hình 47: Hộp thông báo

- Một số khái niệm và hàm được sử dụng trong ví dụ:

+ Tham trị: Là cách truyền mà khi giá trị của tham số hình thức **thay đổi** thì cũng **không** ảnh hưởng đến giá trị của tham số thực.

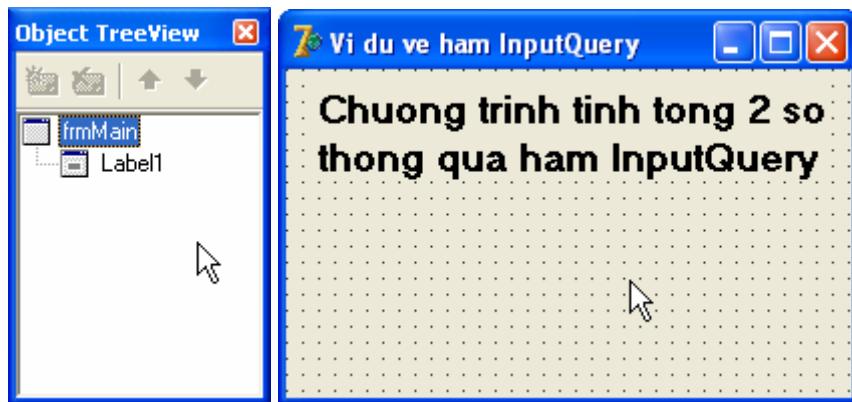
+ Tham biến: Là cách truyền mà khi giá trị của **tham số hình thức** thay đổi thì kéo theo giá trị của **tham số thực** cũng **thay đổi** theo.

+ Câu lệnh rẽ nhánh: **if** điều_kiện **then** công_việc_1 **else** công_việc_2;

Ý nghĩa: Nếu điều kiện đúng, có giá trị True, thì máy thực hiện công việc 1, ngược lại (nghĩa là điều kiện sai, có giá trị False) thì máy sẽ thực hiện công việc 2.

Ví dụ 14: Viết chương trình nhập 2 số bất kỳ bằng hàm InputQuery thông qua sự kiện OnActivate của Form và cho biết kết quả của chúng thông qua thủ tục MessageBox.

- Thiết kế form và đặt tên các đối tượng như sau:



Hình 48: Thiết kế form và đặt tên cho các đối tượng

- Đoạn mã lệnh của chương trình:

```

unit untInputQuery;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmMain = class(TForm)
    Label1: TLabel;
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;
implementation
{$R *.dfm}

procedure TfrmMain.FormActivate(Sender: TObject);
var x, y : string;
  mNhap_x, mNhap_y : boolean; // Kiem tra viec nhap x va y
begin
  mNhap_x := InputQuery('Hop nhap', 'Nhập x =', x);
  mNhap_y := InputQuery('Hop nhap', 'Nhập y =', y);
  if (mNhap_x = False) or (mNhap_y = False) then
    ShowMessage('Bạn đã bỏ qua - Cancel - việc nhập giá trị cho x
                hoặc y.')
end.

```

```

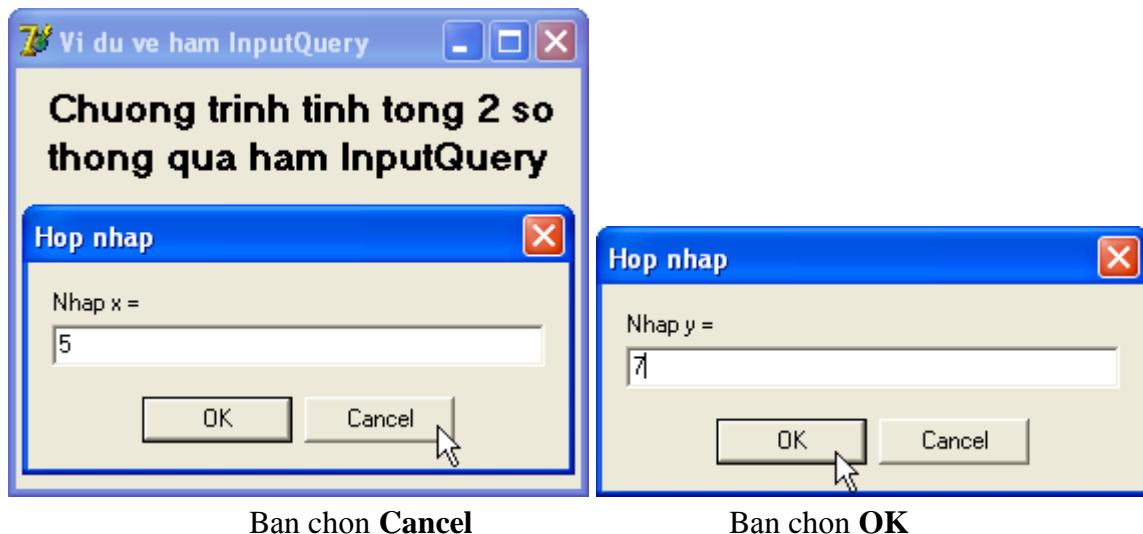
else
  ShowMessage( 'Tong cua ' + x + ' + ' + y + ' = ' +
               FloatToStr(StrToFloat(x) + StrToFloat(y)));
end;

end.

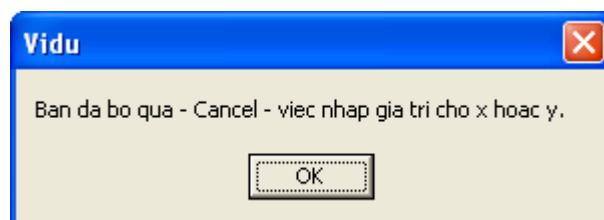
```

- Lưu dự án vào thư mục S:\ViDuInputQuery, biên dịch và chạy chương trình, ta có kết quả như sau:

+ Nếu nhập bỏ nhập trị cho x, và nhập trị y=7

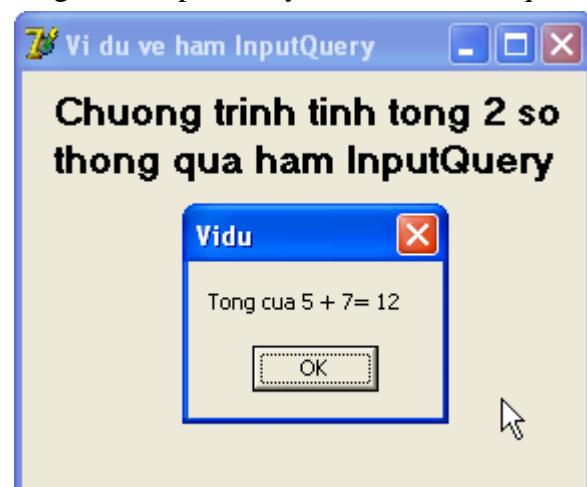


Hình 49: Bỏ nhập x, nhập y=7



Hình 50: Kết quả chương trình sẽ thông báo lỗi

+ Nếu nhập đúng, chẳng hạn nhập x = 5, y =7, thì ta có kết quả như sau:



Hình 51: Kết quả chương trình khi nhập trị cho x, y

IV.3. Nút lệnh (TButton)

Là một thành phần dùng thiết kế nút lệnh để thi hành một công việc nào đó mà bạn muốn.



* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho nút lệnh.
Caption	Chuỗi ký tự là tiêu đề cho nút lệnh. Sử dụng dấu & đê gạch dưới ký tự đại diện. Dùng tổ hợp phím Alt+Ký tự gạch dưới để kích hoạt nút lệnh.
Cancel	Nếu có giá trị True thì khi nhấn phím Esc thì sự kiện OnClick sẽ sinh ra; còn nếu có giá trị False thì không.
Visible	Có giá trị True hoặc False: Có hiển thị hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho hay không cho phép người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về nút lệnh khi bạn trỏ mouse đến nút lệnh đó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà nút lệnh đó được nhận focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 11: Các thuộc tính của TButton

Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnEnter	Khi đối tượng nhận tiêu điểm (Got Focus) hay trở nên hoạt động (Active).
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus).
OnKeyDown	Khi có phím bất kỳ được nhấn.
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 12: Các sự kiện của TButton

- Một hàm được sử dụng trong ví dụ:

+ Hàm thử chuyển đổi một chuỗi, S, sang kiểu số nguyên, out. Nếu thành công hàm trả về giá trị True, còn nếu đổi bị thất bại thì hàm sẽ trả về trị False.

function TryStrToInt(const S: string; out Value: integer): Boolean;

+ Tương tự như hàm trên, hàm này thử chuyển đổi một chuỗi, S, sang kiểu số chấm động, out.

```
function TryStrToFloat(const S: string; out Value: extended): boolean;
```

Ví dụ 15: Tính tổng 2 số:

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmTong
Caption	Ví dụ về đổi tuong Edit

Bảng 13: Thiết lập các thuộc tính Form

- Đặt 3 đối tượng Edit và Button lên form và thiết lập thuộc tính:

Đối tượng:	Thuộc tính	Giá trị
Edit thứ nhất:	Name	edt_a
	TabOrder	0
Edit thứ hai:	Name	edt_b
	TabOrder	1
Edit thứ ba:	Name	edtKQ
	TabOrder	3
	ReadOnly	True
Button	Name	btnTong
	Caption	Tính &tong
	TabOrder	2

Bảng 14: Thiết lập các thuộc tính cho Edit và Button

* Đoạn code hoàn chỉnh của chương trình:

```
unit untTinhTong;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmTong = class(TForm)
    edt_a: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    edt_b: TEdit;
  end;
```

```

edtKQ: TEdit;
Label3: TLabel;
btnTong: TButton;
procedure btnTongClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmTong: TfrmTong;

implementation
{$R *.dfm}

procedure TfrmTong.btnTongClick(Sender: TObject);
var a, b: extended;
begin
  if (TryStrToFloat(edt_a.Text,a) = True) and
  (TryStrToFloat(edt_b.Text,b)= True) then
    edtKQ.Text := FloatToStr(a+b)
  else edtKQ.Text := 'Du lieu nhap bi loi!...';
end;

end.

```

- Lưu dự án vào thư mục S:\ViDuEdit
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 52: Chương trình tính tổng

IV.4. Nhãn và Hộp nhập (TLabelEdit)

Có một thành phần kết hợp cả 2 thành phần TLabel và TEdit để trở thành một, đó chính là thành phần TLabelEdit.

* Biểu tượng:



* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng LabeledEdit.
EditLabel.Caption	Tiêu đề: chuỗi ký tự hiển thị trong phần nhãn của đối tượng.
LabelPosition	Vị trí của nhãn nằm phía trên, bên dưới, bên trái, bên phải so với hộp nhập. Các giá trị tương ứng là: lpAbove, lpBelow, lpLeft, lpRight
Text	Chuỗi trong hộp nhập của đối tượng LabeledEdit
Visible	Có giá trị True hoặc False: Có hiển thị hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về nút lệnh khi bạn trỏ mouse đến nút lệnh đó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà nút lệnh đó được nhận focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 15: Các thuộc tính của TLabeledEdit

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnEnter	Khi đối tượng nhận tiêu điểm (Got Focus) hay trở nên hoạt động (Active)
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 16: Các sự kiện của TLabeledEdit

IV.5. Hộp đánh dấu hay hộp kiểm (TCheckbox)

Là thành phần dùng để trình bày một tùy chọn cho người sử dụng chọn (đánh dấu) hoặc không chọn (bỏ đánh dấu). Thông thường chương trình sẽ có nhiều mục tùy chọn. Người sử dụng có thể không chọn mục nào hoặc chọn tất cả.

* Biểu tượng:



* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng CheckBox.
Caption	Chuỗi ký tự hiển thị trên nút lệnh. Sử dụng dấu & để gạch dưới ký tự đại diện. Dùng tổ hợp phím Alt+Ký tự gạch dưới để kích hoạt nút lệnh.
Checked	Có giá trị True và False: xác lập trạng thái cho Checkbox là được chọn hay chưa. Mặc nhiên có giá trị False.
State	Trạng thái của Checkbox: cbChecked, cbUnchecked hoặc cbGrayed
Visible	Có giá trị True hoặc False: Có hiển thị hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về nút lệnh khi bạn trỏ mouse đến nút lệnh đó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà nút lệnh đó được nhận focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 17: Các thuộc tính của TCheckBox

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnEnter	Khi đối tượng nhận tiêu điểm (Got Focus) hay trở nên hoạt động (Active)
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được gõ
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 18: Các sự kiện của TCheckBox

Chú ý: Để thuận tiện trong việc thiết kế cũng như lập trình, bạn nên dùng thành phần

TGroupBox (có biểu tượng ) để chứa các thành phần Checkbox.

* Một số khái niệm được sử dụng trong ví dụ:

+ Biến chung: là biến có tác động trong tất cả các thủ tục sự kiện và tồn tại trong suốt quá trình chương trình đang chạy (Run-time) và chỉ mất đi (giải phóng biến) khi kết thúc (Terminate/ end) chương trình/dự án.

+ Câu lệnh rẽ nhánh: **if** điều_kiện **then** công_viện_1 **else** công_viện_2;

Nếu điều kiện đúng, có giá trị true, thì máy thực hiện công việc 1; Ngược lại (nghĩa là điều kiện sai, có giá trị false) thì máy sẽ thực hiện công việc 2.

Ví dụ 16: Đánh dấu chọn vào danh sách các ngôn ngữ lập trình mà bạn biết.

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong CheckBox

Bảng 19: Thiết lập thuộc tính cho Form

- Đặt đối tượng GroupBox lên form và đặt thuộc tính như sau:

Thuộc tính	Giá trị
Name	gbxDanhSach
Caption	Danh sach:

Bảng 20: Thiết lập thuộc tính cho GroupBox

- Đặt các đối tượng Checkbox lên GroupBox, Edit lên form và thiết lập thuộc tính:

Đối tượng:	Thuộc tính	Giá trị
CheckBox 1:	Name	chkPascal
	Caption	Pascal
CheckBox 2:	Name	chkDelphi
	Caption	Delphi
CheckBox 3:	Name	chkVC
	Caption	C++/Visual C
CheckBox 4:	Name	chkVB
	Caption	Visual Basic
CheckBox 5:	Name	chkVisualFox
	Caption	Visual Fox
CheckBox 6:	Name	chkAssembler
	Caption	Assembler
CheckBox 7:	Name	chkCOBOL
	Caption	COBOL
CheckBox 8:	Name	chkJava
	Caption	Java

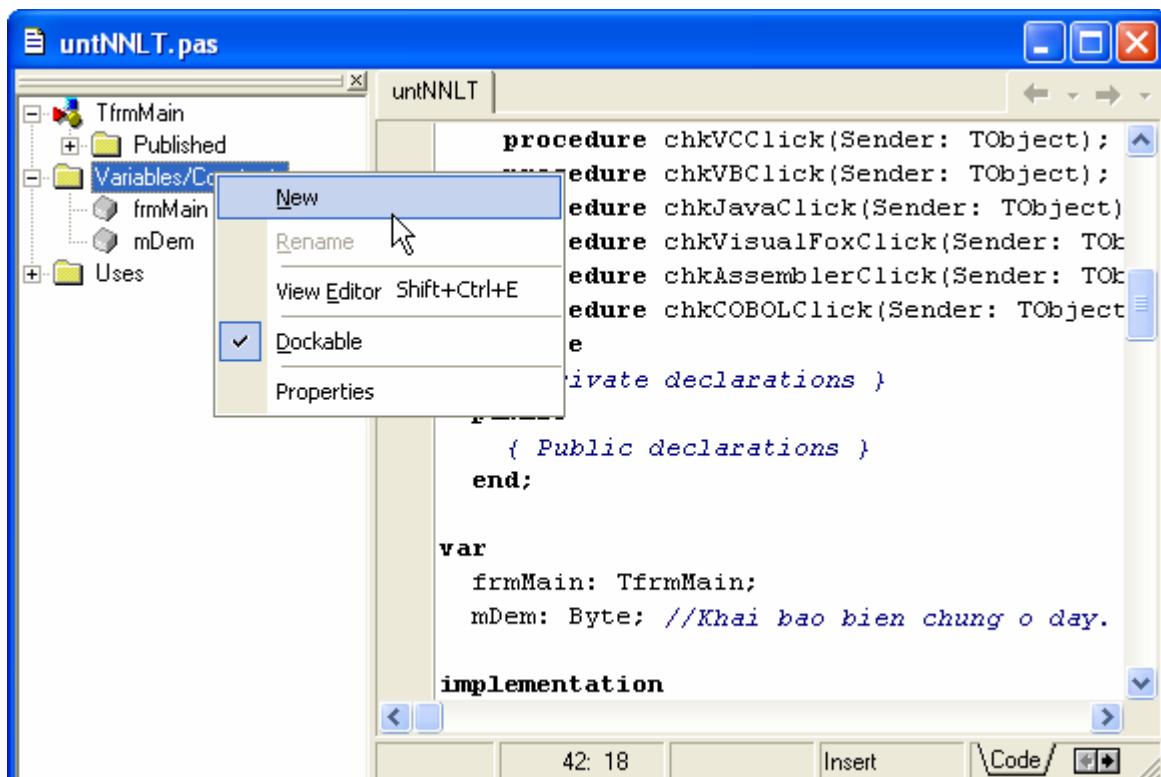
Edit	Name	edt_KQ
	Text	(để rỗng)
	ReadOnly	True

và 2 Label với các caption như mô tả trong form.

Bảng 21: Thiết lập các thuộc tính của CheckBox và Edit

- Viết mã cho chương trình:

Cách khai báo biến chung: Từ chế độ thiết kế Form, nhấn phím chức năng F12 để chuyển sang cửa sổ soạn thảo mã lệnh, rồi cuộn lên đầu chương trình và gõ vào sau từ khóa **var** là mDem: Byte; hoặc RClick vào mục Variables/Contants, chọn New (xem hình phía dưới) để khai báo 1 biến chung mới có tên mDem:Byte;



Hình 53: Cửa sổ thêm biến chung

Lần lượt nhập vào các câu lệnh cho các thủ tục sự kiện tương ứng cho mỗi đối tượng để hoàn thành chương trình như sau:

```
unit untNNLT;      //Các ngôn ngữ lập trình

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TfrmMain = class(TForm)
```

```

gbxDanhSach: TGroupBox;
chkPascal: TCheckBox;
chkDelphi: TCheckBox;
chkVC: TCheckBox;
Label1: TLabel;
chkVB: TCheckBox;
chkJava: TCheckBox;
chkAssembler: TCheckBox;
chkVisualFox: TCheckBox;
chkCOBOL: TCheckBox;
edt_KQ: TEdit;
Label2: TLabel;
procedure chkPascalClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure chkDelphiClick(Sender: TObject);
procedure chkVCClick(Sender: TObject);
procedure chkVBClick(Sender: TObject);
procedure chkJavaClick(Sender: TObject);
procedure chkVisualFoxClick(Sender: TObject);
procedure chkAssemblerClick(Sender: TObject);
procedure chkCOBOLClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;
  mDem: Byte; //Khai bao bien chung o day.

implementation
{$R *.dfm}

procedure TfrmMain.FormCreate(Sender: TObject);
begin
  mDem:=0;      //Khoi tao bien chung
end;

procedure TfrmMain.chkPascalClick(Sender: TObject);
begin
  if chkPascal.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

```

```
procedure TfrmMain.chkDelphiClick(Sender: TObject);
begin
  if chkDelphi.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

procedure TfrmMain.chkVCClick(Sender: TObject);
begin
  if chkVC.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

procedure TfrmMain.chkVBClick(Sender: TObject);
begin
  if chkVB.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

procedure TfrmMain.chkJavaClick(Sender: TObject);
begin
  if chkJava.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

procedure TfrmMain.chkVisualFoxClick(Sender: TObject);
begin
  if chkVisualFox.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

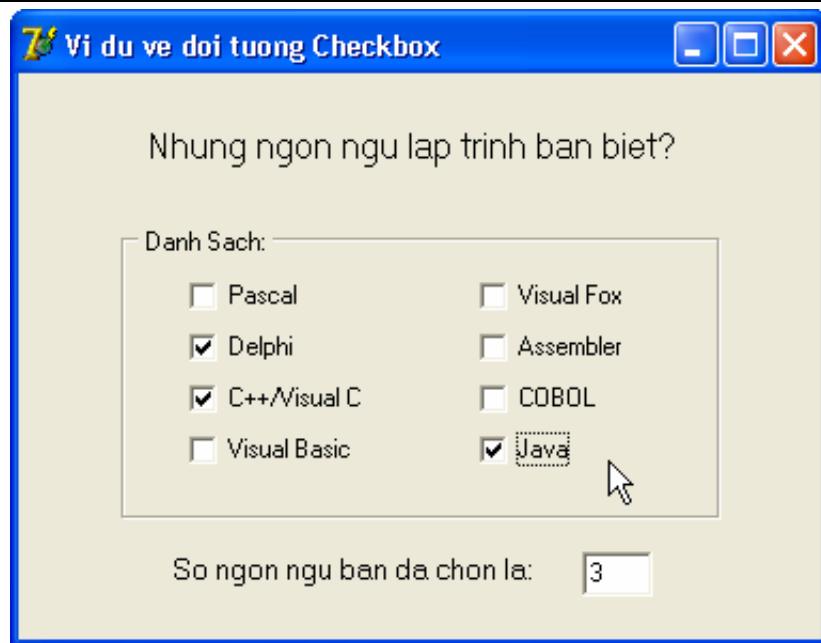
procedure TfrmMain.chkAssemblerClick(Sender: TObject);
begin
  if chkAssembler.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

procedure TfrmMain.chkCOBOLClick(Sender: TObject);
begin
  if chkCOBOL.Checked=True then mDem:=mDem+1 else mDem:=mDem-1;
  edt_KQ.Text:= IntToStr(mDem);
end;

end.
```

+ Lưu dự án vào thư mục **S:\ViDuCheckBox**

+ Biên dịch và chạy chương trình, ta có kết quả như sau:

**Hình 54:** Kết quả chương trình

IV.6. Nút tùy chọn (TRadioButton)

Là thành phần dùng để trình bày một tùy chọn cho người sử dụng chọn lựa (đánh dấu) hoặc không chọn (bỏ đánh dấu). Thông thường chương trình sẽ có nhiều mục tùy chọn. Khác với TCheckBox, người chỉ có thể chọn một mục duy nhất trong các tùy chọn đã cho.



* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho nút tùy chọn, tên này được đặt như tên biến và được sử dụng trong việc viết mã lệnh của chương trình.
Caption	Chuỗi ký tự hiển thị trên nút lệnh. Sử dụng dấu & để gạch dưới ký tự đại diện. Dùng tổ hợp phím Alt+Ký tự gạch dưới để kích hoạt nút lệnh.
Checked	Có giá trị True và False: Xác lập trạng thái cho RadioButton là được chọn hay chưa. Mặc nhiên có giá trị False.
Visible	Có giá trị True hoặc False: Có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về nút lệnh khi bạn trỏ mouse đến nút lệnh đó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà nút lệnh đó được nhận focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 22: Các thuộc tính của TRadioButton

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDoubleClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 23: Các sự kiện của TRadioButton

IV.7. Nhóm tùy chọn (TRadioGroup)

Là thành phần để chứa các RadioButton. Thay vì sử dụng nhiều RadioButton cùng lúc, thì bạn nên sử dụng RadioGroup để dễ thiết kế và lập trình gọn gàng hơn.



* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng RadioGroup
Items	Cho phép bạn nhập các phần tử vào trong đối tượng RadioGroup trong lúc thiết kế thông qua cửa sổ “String List Editor”.
Items[i]	Danh sách các phần tử trong RadioGroup. Phần tử thứ nhất có chỉ số i là 0, phần tử thứ 2 chỉ số i là 1,... trong danh sách.
ItemIndex	Trả về chỉ số phần tử hiện tại đang được chọn trong Items. Mặc nhiên có giá trị là -1 (chưa có phần tử nào được chọn)
Items.Count	Trả về số phần tử trong RadioGroup

Bảng 24: Các thuộc tính của TRadioGroup

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)

Bảng 25: Các sự kiện của TRadioGroup

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Items.Add(const S: string): Integer;	Hàm thêm phần tử kiểu string vào phía cuối và trả về chỉ số của phần tử này trong Items.
Items.Append(const S: string);	Thủ tục nối thêm phần tử kiểu string vào phía cuối Items.
Items.Delete(Index: Integer);	Thủ tục xóa một phần tử ở chỉ số index trong Items.
Items.Insert(Index: Integer; const S: string);	Thủ tục xóa một phần tử kiểu string tại chỉ số index trong Items. Thủ tục này tương đương câu lệnh gán vào thuộc tính Items như sau: Items[index] := S;
Items.Clear;	Thủ tục xóa hết các phần tử trong Items.

Bảng 26: Các phương thức của TRadioGroup

Ví dụ 17: Chọn khóa học Anh văn mà bạn đăng ký.

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong RadioGroup

Bảng 27: Thiết lập thuộc tính cho Form.

- Đặt đối tượng RadioGroup lên form và đặt thuộc tính như sau:

Thuộc tính	Giá trị
Name	rgpCourses
Caption	Courses:
Items	Click vào để chèn vào 5 dòng (phần tử): Level A Level B Level C IELTS TOEFL

Bảng 28: Thiết lập các thuộc tính RadioGroup

Đoạn mã lệnh của chương trình:

```
unit untDangKyAV;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TfrmMain = class(TForm)
    rgpCourses: TRadioGroup;
    Label1: TLabel;
    procedure rgpCoursesClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.rgpCoursesClick(Sender: TObject);
var mChiSo: integer;
begin
  mChiSo := rgpCourses.ItemIndex;
  ShowMessage('You have decided to enroll in '+
              rgpCourses.Items[mChiSo]+'.');
end;
end.
```

+ Lưu dự án vào thư mục **S:\ViDuRadioGroup**

+ Biên dịch và chạy chương trình, ta có kết quả như sau:

**Hình 55:** Kết quả chương trình của RadioGroup

IV.8. Vùng văn bản (TMemo)

Là một thành phần dùng để chứa các dòng văn bản. Thành phần này hữu ích cho những công việc liên quan đến văn bản có nhiều dòng như: đọc để hiển thị nội dung của một tập tin dạng text, soạn thảo văn bản,...

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng vùng văn bản.
Align	Xác định việc canh lè trái, phải, trên, dưới,... của Memo bên trong thành phần chứa nó. Nếu có giá trị bằng alClient thì kích thước đối tượng Memo sẽ chiếm toàn bộ bề mặt của thành phần chứa nó.
Alignment	Canh trái (taLeftJustify), canh phải (taRightJustify), canh giữa (taCenter) chuỗi văn bản trong Memo
Font	Xác định các thuộc tính về font như: kiểu chữ, màu, cỡ,...
Lines	Cho phép bạn nhập các dòng văn bản vào trong Memo trong lúc thiết kế thông qua cửa sổ “String List Editor”.
Lines[i]	Dòng văn bản ở chỉ số thứ i trong Memo. Dòng thứ nhất có chỉ số i là 0, dòng thứ 2 chỉ số i là 1,... trong danh sách.
Lines.Count	Trả về số dòng (line) trong Memo
SelText	Trả về khôi văn bản được chọn.
WantReturns	Có giá trị True và False: Cho hoặc không cho phép việc sử dụng phím Enter để xuống hàng (ký tự Carry Return) khi soạn thảo văn bản trong Memo.
WantTabs	Có giá trị True và False: Cho hoặc không cho phép việc sử dụng phím Tab khi soạn thảo văn bản trong Memo.
ReadOnly	Nếu giá trị True thì nội dung văn bản trong Memo là chỉ đọc.

Tên thuộc tính	Ý nghĩa
ScrollBars	Có thanh cuộn đứng (ssVertical), ngang (ssHorizontal), cả hai thanh cuộn (ssBoth) trong Memo hoặc không có thanh cuộn nào (ssNone).
Visible	Có giá trị True hoặc False: Có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về Memo khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà Memo đó được nhận focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True và WantTabs là False.
WordWrap	Nếu có giá trị True: Chuỗi văn bản được nhập vào trên mỗi dòng mà dài hơn độ rộng của vùng soạn thảo Memo thì sẽ được tự động xuống dòng mới, ngược lại sẽ không tự động xuống dòng mới.

Bảng 29: Các thuộc tính của TMemo

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnChange	Khi người sử dụng thay đổi, cập nhật dữ liệu trong đối tượng Memo
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 30: Các sự kiện của TMemo

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Lines.Add(const S: string): Integer;	Hàm thêm dòng kiểu string vào phía cuối và trả về chỉ số của phần tử này trong Lines.
Lines.Append(const S: string);	Thủ tục nối thêm phần tử kiểu string vào phía cuối Lines.
Lines.Delete(Index: Integer);	Thủ tục xóa một dòng ở chỉ số index trong Lines.

Cú pháp của phương thức	Ý nghĩa
Lines.Insert(Index: Integer; const S: string);	Thủ tục xóa một phần tử kiểu string tại chỉ số index trong Lines. Thủ tục này tương đương câu lệnh gán vào thuộc tính Items như sau: Lines[index] := S;
CopyToClipboard;	Thủ tục sao chép khôi văn bản được chọn và lưu vào vùng nhớ Clipboard của Windows.
CutToClipboard;	Thủ tục cắt khôi văn bản được chọn và lưu vào vùng nhớ Clipboard của Windows.
PasteFromClipboard;	Thủ tục dán vùng văn bản trong Clipboard vào Memo.
Clear;	Thủ tục xóa hết các dòng trong Lines.
ClearSelection;	Thủ tục xóa vùng văn bản được chọn trong Memo.

Bảng 31: Các phương thức của TMemo

Ví dụ 18: Chương trình soạn thảo NotePad đơn giản, với 4 nút lệnh:

Clear: Xóa nội dung trong vùng soạn thảo

Quét khôi văn bản: sử dụng thao tác Drag and drop.

Copy: Sao chép; Cut: Cắt vùng văn bản đã được quét khôi

Paste: Dán khôi văn bản vừa Copy hoặc Cut vào tại vị trí con nháy.

Lưu ý:

Khi mới chạy chương trình, các nút Copy, Cut và Paste bị mờ đi (Disable)

Khi thực hiện việc quét khôi thì các nút lệnh Copy, Cut mới tác động (Enable).

Khi thực hiện lệnh Copy hoặc Cut thì nút lệnh Paste mới tác động.

- Tạo một dự án mới File/New/Application

- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmNotePad
Caption	Vi du ve doi tuong Memo

Bảng 32: Thiết lập thuộc tính Form

- Đặt các thành phần lên form và đặt thuộc tính chúng như sau:

Đối tượng	Thuộc tính	Giá trị
Memo	Name	memNotePad
	Lines	rỗng (chưa có dòng nào)
	TabOrder	0

Đối tượng	Thuộc tính	Giá trị
Button thứ 1:	Name	btnClear
	Caption	Clea&r
Button thứ 2:	Name	btnCopy
	Caption	&Copy
Button thứ 3:	Name	btnCut
	Caption	C&ut
Button thứ 4:	Name	btnPaste
	Caption	&Paste

và một Label với Caption như mô tả trong form.

Bảng 33: Thiết lập các thuộc tính thành cho các đối tượng.

Đoạn mã lệnh của chương trình:

```

unit untNotePad;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmNotePad = class(TForm)
    memNotePad: TMemo;
    btnCopy: TButton;
    btnCut: TButton;
    btnPaste: TButton;
    btnClear: TButton;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure memNotePadMouseUp(Sender: TObject; Button:
      TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure btnCopyClick(Sender: TObject);
    procedure btnCutClick(Sender: TObject);
    procedure btnPasteClick(Sender: TObject);
    procedure btnClearClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

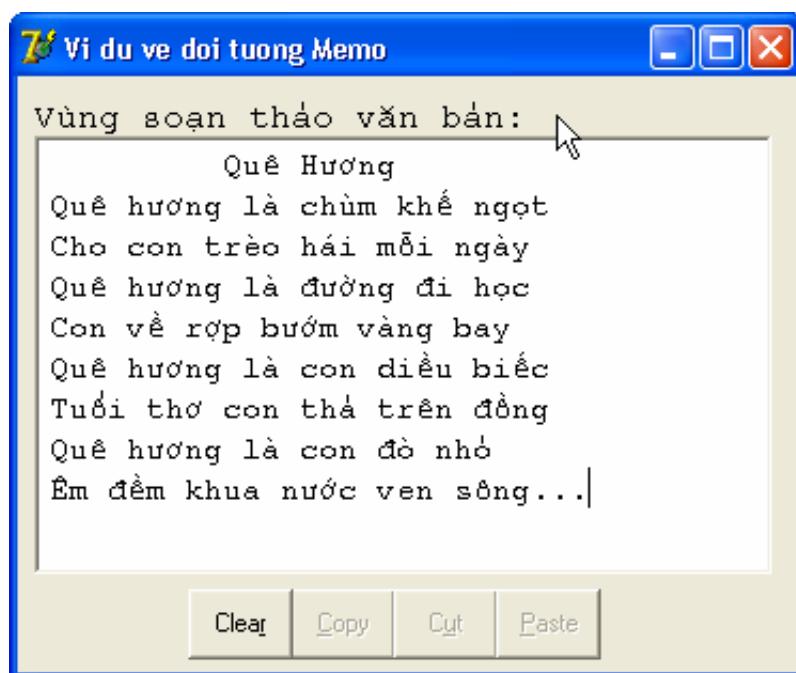
var
  frmNotePad: TfrmNotePad;

```

```
implementation  
{$R *.dfm}  
  
procedure TfrmNotePad.FormCreate(Sender: TObject);  
begin  
    btnCopy.Enabled:=False;  
    btnCut.Enabled:=False;  
    btnPaste.Enabled:=False;  
end;  
  
procedure TfrmNotePad.memNotePadMouseUp(Sender: TObject;  
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);  
begin  
    if length(memNotePad.SelText)>0 then  
        begin  
            btnCopy.Enabled:=true;  
            btnCut.Enabled:=true;  
        end;  
    end;  
  
procedure TfrmNotePad.btnCopyClick(Sender: TObject);  
begin  
    memNotePad.CopyToClipboard;  
    btnPaste.Enabled:=True;  
end;  
  
procedure TfrmNotePad.btnCutClick(Sender: TObject);  
begin  
    memNotePad.CutToClipboard;  
    btnPaste.Enabled:=True;  
end;  
  
procedure TfrmNotePad.btnPasteClick(Sender: TObject);  
begin  
    memNotepad.PasteFromClipboard;  
    memNotePad.SetFocus;  
end;  
  
procedure TfrmNotePad.btnClearClick(Sender: TObject);  
begin  
    memNotePad.Clear;  
end;  
  
end.
```

- Lưu dự án vào thư mục **S:\ViDuMemo**

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 56: Kết quả chương trình Memo

IV.9. Hộp danh sách (TListBox)

Là thành phần liệt kê một danh sách các phần tử. Các phần tử này **không** thể nhập trực tiếp khi chạy chương trình.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho hộp danh sách.
Align	Xác định việc canh lè trái, phải, trên, dưới,... của Listbox bên trong thành phần chứa nó. Nếu có giá trị bằng alClient thì kích thước ListBox sẽ chiếm toàn bộ bề mặt của thành phần chứa nó.
Columns	Số cột trong ListBox
Font	Xác định các thuộc tính về font như: kiểu chữ, màu, cỡ,...
Items	Cho phép bạn nhập các phần tử vào ListBox lúc thiết kế thông qua cửa sổ “String List Editor”.
Items[i]	Phần tử ở chỉ số thứ i trong Listbox. Chỉ số i bắt đầu từ 0 và tiếp tục là 1, 2,... trong hộp danh sách. Cách này thường được dùng viết mã lệnh.
ItemIndex	Trả về chỉ số phần tử hiện tại đang được chọn trong Items. Mặc nhiên có giá trị là -1 (chưa có phần tử nào được chọn)
Count	Trả về số phần tử có mặt trong ListBox
SelCount	Trả về số phần tử có đã được chọn trong Listbox
MultiSelect	Có giá trị True hoặc False: Cho phép Click chọn cùng lúc nhiều phần

Tên thuộc tính	Ý nghĩa
	tử (kết hợp với phím Ctrl hoặc Shift) hay chỉ 1 phần tử.
Sorted	Có giá trị True hoặc False: Sắp xếp các phần tử theo thứ tự chữ cái hoặc không sắp xếp.
Visible	Có giá trị True hoặc False: có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về ListBox khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà ListBox đó được nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 34: Các thuộc tính của TListBox

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 35: Các sự kiện của TListBox

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Items.Add(const S: string): Integer;	Hàm thêm phần tử kiểu string vào phía cuối và trả về chỉ số của phần tử này trong Items.
Items.Append(const S: string);	Thủ tục nối thêm phần tử kiểu string vào phía cuối Items.
Items.Delete(Index: Integer);	Thủ tục xóa một phần tử ở chỉ số index trong Items.
Items.Insert(Index: Integer; const S: string);	Thủ tục xóa một phần tử kiểu string tại chỉ số index trong Items. Thủ tục này tương đương câu lệnh gán vào thuộc tính Items như sau: Items[index] := S;

Cú pháp của phương thức	Ý nghĩa
Clear;	Thủ tục xóa hết các phần tử trong ListBox.
ClearSelection;	Thủ tục xóa các phần tử đã chọn.

Bảng 36: Các phương thức của TListBox

Ví dụ 19: Đếm số phần tử mà bạn đã chọn trong ListBox

- Tạo một dự án mới File/New/Application
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong ListBox

Bảng 37: Thiết lập thuộc tính Form.

- Đặt các đối tượng lên form và đặt thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
ListBox	Name	lbxSo
	Items	rỗng (0 line)
	Columns	2
	MultiSelect	True
Edit	Name	edt_pt
	Text	rỗng
	TabOrder	0
Button 1:	Name	btnThem
	Caption	&Them phan tu vao Listbox
Button 2:	Name	btnDem
	Caption	&Dem so phan tu chon
và 2 Label với Caption như mô tả trong form.		

Bảng 38: Thiết lập thuộc tính cho các đối tượng.

Đoạn mã lệnh chương trình:

```
unit untDanhSach;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, Qt;
```

```

type
  TfrmMain = class(TForm)
    lbxSo: TListBox;
    edt_pt: TEdit;
    btnThem: TButton;
    btnDem: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure btnThemClick(Sender: TObject);
    procedure btnDemClick(Sender: TObject);
    procedure edt_ptKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMain: TfrmMain;
implementation
{$R *.dfm}

procedure TfrmMain.btnThemClick(Sender: TObject);
begin
  lbxSo.Items.Add(edt_pt.Text);
  edt_pt.Clear;
  edt_pt.SetFocus;
end;

procedure TfrmMain.btnDemClick(Sender: TObject);
begin
  ShowMessage('So phan tu ban da chon la: ' +
    IntToStr(lbxSo.SelCount));
end;

{TTSK sử dụng phím Enter để đưa giá trị nhập trong Edit vào
ListBox }

procedure TfrmMain.edt_ptKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if Key = 13 then //Go phím Enter
  begin
    lbxSo.Items.Add(edt_pt.Text);
    edt_pt.Clear;
  end;
end;
end.

```

- Lưu dự án vào thư mục S:\ViDuListBox
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 57: Kết quả chương trình ListBox

IV.10. TSpinEdit

Thành phần này nằm trong thẻ Sample. Nó cho phép bạn nhập vào một số nguyên bất kỳ như thành phần TEdit. Tuy nhiên, TSpinEdit còn cho phép bạn thay đổi (tăng, giảm) giá trị này theo một bước nhảy (step/increment) nào đó bằng cách Click vào nút (nút UpDown) để tăng hoặc giảm.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng.
Value	Kiểu integer: Giá trị của đối tượng SpinEdit
MinValue	Kiểu integer: Giá trị thấp nhất của SpinEdit, mặc nhiên có giá trị 0: có nghĩa là không giới hạn và có thể có giá trị nhỏ nhất trong phạm vi kiểu số nguyên là -2147483648
MaxValue	Kiểu integer: Giá trị cao nhất của TSpinEdit, mặc nhiên có giá trị 0: có nghĩa là không giới hạn và có thể có giá trị lớn nhất trong phạm vi kiểu số nguyên là 2147483647
Increment	Kiểu integer: Giá trị bước nhảy (step). Mặc nhiên có giá trị là 1.
MaxLength	Kiểu integer: Độ dài lớn nhất (hay số lượng ký số nhiều nhất) của số nguyên trong SpinEdit. Chẳng hạn bạn giới hạn chỉ nhập số tối đa đến hàng ngàn thì bạn cho thuộc tính này bằng 4.
Text	Kiểu String: Chuỗi ký số bên trong đối tượng. Thuộc tính này tương đương với thuộc tính Value nhưng khác kiểu dữ liệu.
ReadOnly	Kiểu boolean: Có giá trị là True, nếu KHÔNG cần thay đổi giá trị của thuộc tính Value. Còn ngược lại, bạn có thể thay đổi.

Tên thuộc tính	Ý nghĩa
Visible	Kiểu boolean: Có giá trị True hoặc False: Có xuất hiện SpinEdit và các thành phần bên nó hay không khi chạy chương trình.
Enabled	Kiểu boolean: Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến các thành phần mà nó chứa hay không.
Hint	Dòng gợi ý ngắn gọn về TPanel khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà SpinEdit đó được nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 39: Các thuộc tính của TSpinEdit

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi bạn gõ phím bất kỳ
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...
OnChange	Khi bạn thay đổi giá trị trong đối tượng của thành phần TSpinEdit

Bảng 40: Các sự kiện của TSpinEdit

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Show;	Thủ tục hiển thị đối tượng khi chạy chương trình.
Hide;	Thủ tục ẩn đối tượng khi chạy chương trình.
Clear;	Thủ tục xóa hết giá trị trong TSpinEdit

Bảng 41: Các phương thức của TSpinEdit

IV.11. Hộp danh sách đánh dấu (TCheckListBox)

Là thành phần kết hợp cả chức năng của TListBox và TCheckBox mà các bạn đã học. Trong TListBox, mỗi phần tử tương ứng với một dòng. Còn trong TCheckListBox, mỗi phần tử tương ứng với mỗi hộp kiểm. Tính năng của này cho người lập trình chọn phần tử nào trong danh sách chỉ cần đánh dấu chọn trong hộp kiểm là xong. Thành phần này nằm ở trong thẻ Additional.

* Biểu tượng:



* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho hộp danh sách đánh dấu.
Align	Xác định việc canh lề trái, phải, trên, dưới,... của TCheckListBox bên trong thành phần chứa nó. Nếu có giá trị bằng alClient thì kích thước CheckListBox sẽ chiếm toàn bộ bề mặt của thành phần chứa nó.
Columns	Số cột trong đối tượng CheckListBox
Items	Cho phép bạn nhập các phần tử có dạng là hộp kiểm vào CheckListBox trong lúc thiết kế thông qua cửa sổ “String List Editor”.
Items[i]	Phần tử ở chỉ số thứ i trong CheckListBox. Chỉ số i bắt đầu từ 0 và tiếp tục là 1, 2,... trong hộp danh sách. Cách này thường được dùng viết mã lệnh.
Checked[i]	Phần tử thứ i trong TCheckListBox được đánh dấu chọn. Chỉ số i bắt đầu từ 0 và tiếp tục 1, 2,...
ItemIndex	Trả về chỉ số phần tử hiện tại đang được chọn trong Items. Mặc nhiên có giá trị là -1 (chưa có phần tử nào được chọn)
Count	Trả về số phần tử có mặt trong CheckListBox
SelCount	Trả về số phần tử có đã được chọn trong CheckListBox
MultiSelect	Có giá trị True hoặc False: Cho phép Click chọn cùng lúc nhiều phần tử (kết hợp với phím Ctrl hoặc Shift) hay chỉ 1 phần tử.
Sorted	Có giá trị True hoặc False: Sắp xếp các phần tử theo thứ tự chữ cái hoặc không sắp xếp.
Visible	Có giá trị True hoặc False: có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi tả ngắn gọn về CheckListBox khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà CheckListBox đó được nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 42: Các thuộc tính của TCheckListBox

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.

Tên sự kiện	Đáp ứng của sự kiện
OnDblClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 43: Các sự kiện của TCheckListBox

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Items.Add(const S: string): Integer;	Hàm thêm phần tử kiểu string vào phía cuối và trả về chỉ số của phần tử này trong Items.
Items.Append(const S: string);	Thủ tục nối thêm phần tử kiểu string vào phía cuối Items.
Items.Delete(Index: Integer);	Thủ tục xóa một phần tử ở chỉ số index trong Items.
Items.Insert(Index: Integer; const S: string);	Thủ tục xóa một phần tử kiểu string tại chỉ số index trong Items. Thủ tục này tương đương câu lệnh gán vào thuộc tính Items như sau: Items[index] := S;
Clear;	Thủ tục xóa hết các phần tử trong TCheckListBox.
ClearSelection;	Thủ tục xóa các phần tử đã chọn.

Bảng 44: Các phương thức của TCheckListBox

Ví dụ 20: Viết chương trình Đăng ký và tính học phí cho sinh viên

- Tạo một dự án mới File/New/Application
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmVidu
Caption	Vi du ve CheckListBox

Bảng 45: Thiết lập thuộc tính Form.

- Đặt các đối tượng lên form và đặt thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
CheckListBox	Name	clbTenMH

Đối tượng	Thuộc tính	Giá trị
	Items	Gồm 15 phần tử như sau: Thuc hanh Tin hoc can ban LT Microsoft Access TH Microsoft Access LT Lap trinh Can ban B - Delphi TH Lap trinh can ban B Anh van Can ban Dai so Tuyen tinh Toan cao cap Xac xuat thong ke Ke toan dai cuong Suc ben Vat lieu Ly thuyet mang Cai dat & Quan tri mang Thiet ke Web
ListBox	Name	IbxSoTC
	Items	Gồm 15 phần tử như sau: 2 3 2 3 2 2 3 3 5 4 5 5 3 5
SpinEdit	Name	speGia1TC
	MinValue	15000
	MaxValue	30000
	Value	15000
	Increment	1000
Memo	Name	memTienHP
	Lines	0 line (rỗng)
Button 1	Name	btnTinhTien
	Caption	Tinh &tien hoc phi
Button 2	Name	btnThemMH
	Caption	Them ten &mon hoc moi
và một số Label như thiết kế ở form		

Đoạn mã lệnh chương trình:

```

unit untViDu;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, CheckLst, Spin, ExtCtrls;
type
  TfrmViDu = class(TForm)
    clbTenMH: TCheckListBox;
    btnTinhTien: TButton;
    Label1: TLabel;
    lbxSoTC: TListBox;
    Label2: TLabel;
    speGia1TC: TSpinEdit;
    Label3: TLabel;
    btnThemMH: TButton;
    memTienHP: TMemo;
    Label4: TLabel;
    Label5: TLabel;
    procedure btnThemMHClick(Sender: TObject);
    procedure btnTinhTienClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmViDu: TfrmViDu;

implementation
{$R *.dfm}

procedure TfrmViDu.btnThemMHClick(Sender: TObject);
var mTenMH:string; mSoTC:integer;
begin
  mTenMH := InputBox('Hop nhap','Nhap ten mon hoc','');
  if TryStrToInt(InputBox('Hop nhap','Nhap so tin chi','3'),
    mSoTC) and (mSoTC>=1) and (mTenMH<>'') then
  begin
    clbTenMH.Items.Add(mTenMH);
    lbxSoTC.Items.Add(IntToStr(mSoTC));
  end
  else ShowMessage('Ban nhap SAI du lieu ve mon hoc.');
end;

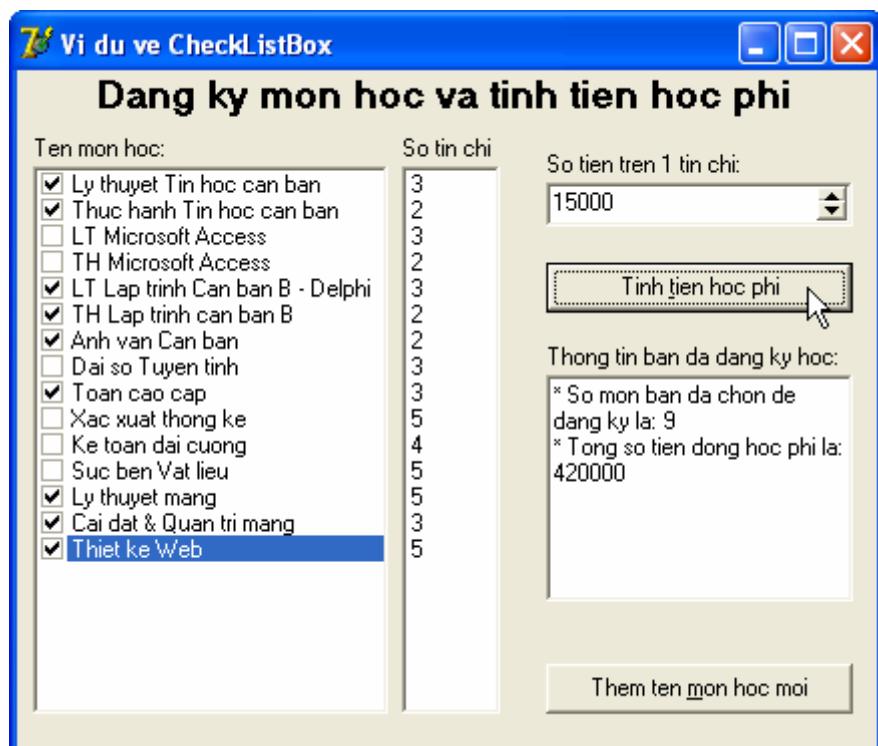
```

```

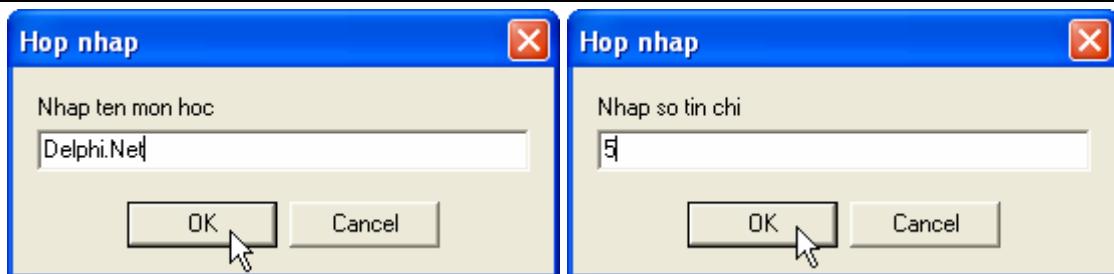
procedure TfrmViDu.btnAddTienClick(Sender: TObject);
var i, mSoMH, mTongSoTC, mTienHP:integer;
begin
  mSoMH:=0; mTongSoTC:=0;
  memTienHP.Lines.Clear;
  for i:=0 to clbTenMH.Items.Count - 1 do
    if clbTenMH.Checked[i] then
    begin
      inc(mSoMH);
      inc(mTongSoTC,StrToInt(lbxSoTC.Items[i]));
    end;
  mTienHP:=mTongSoTC*speGia1TC.Value;
  memTienHP.Lines.Add('* So mon ban da chon de dang ky la: '+
                      IntToStr(mSoMH));
  memTienHP.Lines.Add('* Tong so tien dong hoc phi la: '+
                      IntToStr(mTienHP));
end;
end.
```

- Lưu dự án vào thư mục S:\ViDuCheckListBox

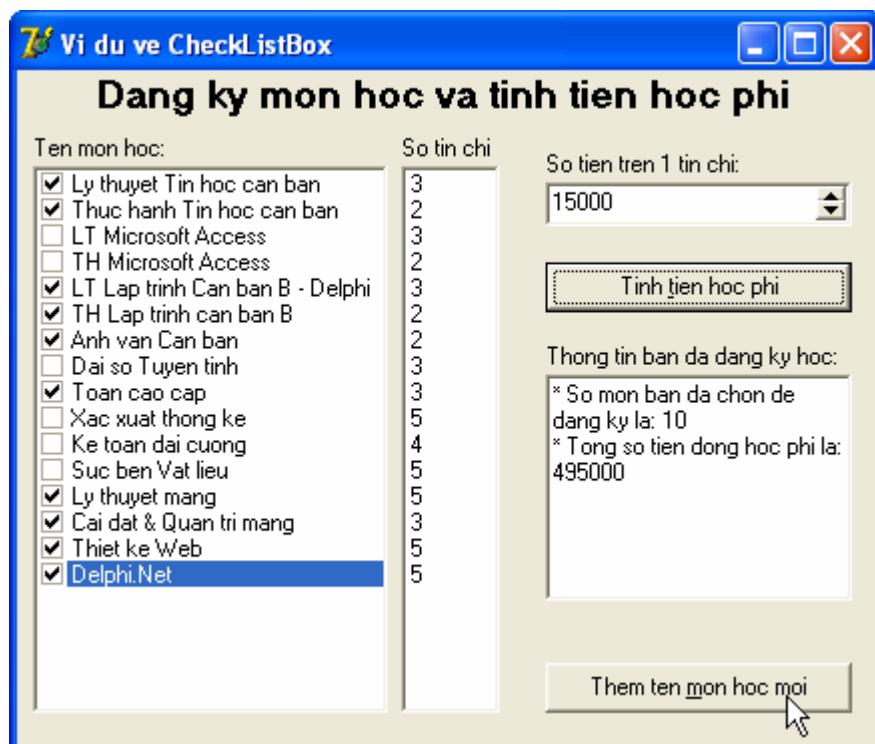
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 58: Chọn tên môn học và chọn Tính tiền học phí



Hình 59: Thêm tên môn học mới và số tín chỉ



Hình 60: Sau khi thêm tên môn học mới và tính lại tiền học phí

IV.12. Hộp danh sách các khoá (TValueListEditor)

Thành phần này gần giống như TListBox, nhưng mỗi phần tử/dòng thì có 2 phần: phần **tên khoá** (Key name) và phần **trị khoá** (Key value). Hai phần này được thể hiện trên 2 cột và tiêu đề của 2 cột này mặc nhiên là **Key** và **Value**. Tuy nhiên ta có thể thay đổi tên tiêu đề này. Thành phần này nằm ở trong thẻ Additional.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng ValueListEditor, tên này được đặt như tên biến và được sử dụng trong việc viết mã lệnh của chương trình.
TitleCaptions	Tiêu đề cho các cột của ValueListEditor. Bạn Click vào nút để thay đổi tên tiêu đề nếu bạn cần thông qua cửa sổ “String List Editor”
DisplayOptions	doColumnTitles: Có hiển thị (True) hay không (False) hiển thị hàng tiêu đề. Nếu có hàng tiêu đề, thì nó là hàng đầu tiên trong

Tên thuộc tính	Ý nghĩa
	ValueListEditor nên có chỉ số hàng là 0.
Strings	Cho phép bạn nhập các phần tử hay hàng của TValueListEditor, với mỗi phần tử có 2 phần: Tên khoá và trị khoá vào TValueListEditor trong lúc thiết kế thông qua cửa sổ “Value List Editor” khi bạn Click vào nút 
RowCount	Trả về số hàng (có tính cả hàng tiêu đề) có mặt trong ValueListEditor.
ColCount	Trả về số cột trong ValueListEditor.
Strings.Count	Số hàng/phần tử không tính hàng tiêu đề có trong ValueListEditor
Keys[i]	Là tên khoá (kiểu String) thứ i trong ValueListEditor. Chỉ số i bắt đầu từ 0 và tiếp tục là 1, 2,... Cách này thường được dùng viết mã lệnh (Coding). Bạn cần chú ý là: Nếu không hiển thị hàng tiêu đề, thì phần tử đầu tiên sẽ có chỉ số bắt đầu là 0. Còn nếu có hiển thị hàng tiêu đề thì phần tử đầu tiên có chỉ số hàng bắt đầu từ 1.
Values[i]	Là trị khoá (kiểu String) thứ i của ValueListEditor. Trong đó i lên tên của khoá của phần tử thứ i.
Strings[i]	Trả về trị khoá (kiểu String) của phần tử thứ i. Chỉ số i luôn luôn bắt đầu từ 0 rồi 1, 2,... Bạn cần chú ý là: Thuộc tính này KHÔNG tính hàng tiêu đề. Nếu có hiển thị hàng tiêu đề thì phần tử đầu tiên có chỉ số là 0 và phần tử cuối cùng là RowCount-1 (hay Strings.Count). Còn nếu không hiển thị hàng tiêu đề, thì phần tử đầu tiên có chỉ số là 0 và phần tử cuối cùng là RowCount-2 (hay Strings.Count-1).
Row	Trả về chỉ số hàng hiện hành của phần tử được chọn khi chạy chương trình. Nếu không hiển thị hàng tiêu đề thì chỉ số hàng đầu tiên bắt đầu từ 0 rồi 1, 2, ... Còn nếu có hiển thị hàng tiêu đề thì chỉ số hàng đầu tiên có chỉ số hàng bắt đầu từ 1.
Col	Trả về chỉ số cột hiện hành của phần tử được chọn khi chạy chương trình. Chỉ số cột bắt đầu từ 0 rồi 1, 2, ...
Cell[i,j]	Trả về giá trị trong ô ở chỉ số cột i và hàng j.
KeyOptions	Các tùy chọn trên cột khoá (Key): Cho phép (True) hay không cho phép (False): sửa tên khoá - keyEdit, thêm tên khoá - keyAdd, xoá tên khoá - keyDelete, kiểm tra tên khoá duy nhất (không trùng khoá) - keyUnique.
Options	Các tùy chọn trên cột giá trị (Value) của khoá: Cho phép (True) hay không cho phép (False): thay đổi trị khoá- goEditing, sử dụng phím Tab và tổ hợp phím Shift + Tab - goTabs, chọn cả hàng – goRowSelect,...
Visible	Có giá trị True hoặc False: có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi tả ngắn gọn về CheckListBox khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.

Tên thuộc tính	Ý nghĩa
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà đối tượng này nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 46: Các thuộc tính của TCheckListBox

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...
OnSelectCell	Khi một ô bất kỳ trong TValueListEditor được Click chọn.

Bảng 47: Các sự kiện của TValueListEditor

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
InsertRow(const KeyName, Value: string ; Append: Boolean): Integer;	Hàm chèn thêm vào một hàng (phân tử) gồm tên khoá và trị khoá. Nếu muốn chèn phía trên hàng hiện hành thì truyền trị False cho tham số Append; Còn nếu muốn chèn phân tử vào cuối danh sách thì cho Append bằng True.
DeleteRow(ARow: Integer);	Thủ tục xoá một hàng bất kỳ.
FindRow(const KeyName: string ; var Row: Integer): Boolean;	Hàm tìm một phân tử dựa vào tên khoá (Key). Nếu tìm thấy thì hàm sẽ trả về giá trị True và tham biến Row sẽ chứa chỉ số hàng của phân tử đầu tiên được tìm thấy. Còn ngược lại hàm trả về giá trị False.
Strings.Add(const S: string): Integer;	Hàm thêm một trị khoá (cột Value) kiểu string vào phía cuối và trả về chỉ số của phân tử này.
Strings.Append(const S: string);	Thủ tục nối thêm một trị khoá ở cột Value.
Strings.Delete(Index: Integer);	Thủ tục xóa một hàng/phân tử ở chỉ số index trong Items.
Strings.Clear;	Thủ tục xóa hết các hàng/phân tử trong TValueListEditor ngoại trừ hàng tiêu đề.

Bảng 48: Các phương thức của TValueListEditor

Ví dụ 21: Viết chương trình nhập vào danh sách các mặt hàng và đơn giá. Tìm kiếm theo mặt hàng.

- Tạo một dự án mới **File/New/Application**

- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve ValueListEditor

Bảng 49: Thiết lập thuộc tính Form.

- Đặt các đối tượng lên form và đặt thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị	
ValueListEditor	Name	vleMatHang	
	TitleCaptions	2 lines tương ứng là Key và Value: Ten Giao Trinh: Don Gia:	
	Strings	Key	Value
		Ly thuyet Access	150
		Thuc hanh Access	145
		Ly thuyet Delphi	165
		Thuc hanh Delphi	140
LabeledEdit thứ 1	KeyOptions	KeyEdit = True, KeyAdd=True KeyDelete = False, KeyUnique=True	
	Name	lbeTenHang	
	EditLabel.Caption	Nhập &Ten giao trinh moi:	
	LabelPosistion	lpAbove	
LabeledEdit thứ 2	Text	(rỗng)	
	Name	lbeDonGia	
	EditLabel.Caption	Nhập &Don gia:	
	LabelPosistion	lpAbove	
LabeledEdit thứ 3	Text	(rỗng)	
	Name	lbeKQTK	
	EditLabel.Caption	Kết quả tìm kiếm.	
	LabelPosistion	lpBelow	
Button thứ 1	Text	(rỗng)	
Button thứ 1	Name	btnThem	

Đối tượng	Thuộc tính	Giá trị
	Caption	Them vao & danh sach
Button thứ 2	Name	btnTimKiem
	Caption	Tim & kiem

và 1 Label như thiết kế ở form

Đoạn mã lệnh chương trình:

```

unit untViDu;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, ValEdit, StdCtrls, ExtCtrls;

type
  TfrmMain = class(TForm)
    vleMatHang: TValueListEditor;
    lbeTenHang: TLabeledEdit;
    lbeDonGia: TLabeledEdit;
    btnThem: TButton;
    Label1: TLabel;
    btnTimKiem: TButton;
    lbeKQTK: TLabeledEdit;
    procedure btnThemClick(Sender: TObject);
    procedure btnTimKiemClick(Sender: TObject);

private
  { Private declarations }

public
  { Public declarations }
  end;

var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.btnThemClick(Sender: TObject);
var mDGia:Single; mVitri:integer;
begin
  if (lbeTenHang.GetTextLen>0) and TryStrToFloat(lbeDonGia.Text,mDGia) and
  (mDGia>0) then

```

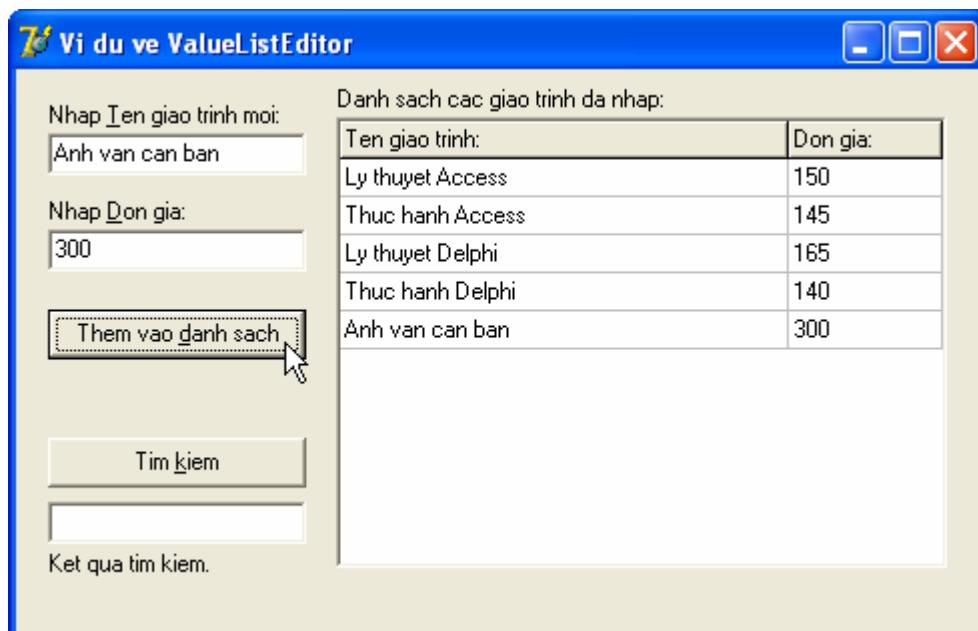
```

if vleMatHang.FindRow(lbeTenHang.Text,mViTri) = True then
    ShowMessage('Sorry!... Giao trinh nay (Key) da ton tai roi.')
else //Chua ton tai, thi them vao
    vleMatHang.InsertRow(lbeTenHang.Text, lbeDonGia.Text,True)
else
    ShowMessage('Ban nhap du lieu khong hop le!...');
end;

procedure TfrmMain.btnTimKiemClick(Sender: TObject);
var mOK, mTimThay:Boolean; mTenHang: string; mViTri:Integer;
begin
    lbeKQTK.Clear;
    mOK := InputQuery('Hop Nhaph', 'Nhaph chinh xac ten giao trinh can tim: ',mTenHang);
    if mOK then
        begin
            mTimThay := vleMatHang.FindRow(mTenHang, mViTri);
            if mTimThay then lbeKQTK.Text := 'Tim thay o vi tri thu ' + IntToStr(mViTri)
            else lbeKQTK.Text := 'Tim KHONG thay.';
        end
        else ShowMessage('Ban bo qua viec tim kiem.');
    end;
end.

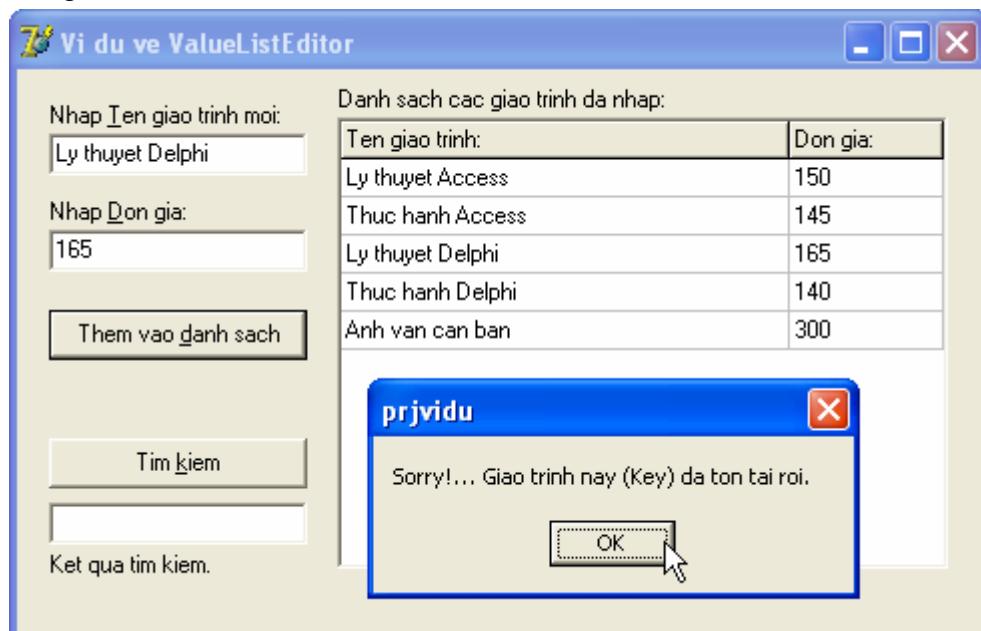
```

- Lưu dự án vào thư mục **S:\ViDuValueListEditor**
- Biên dịch và chạy chương trình, ta có kết quả như sau:

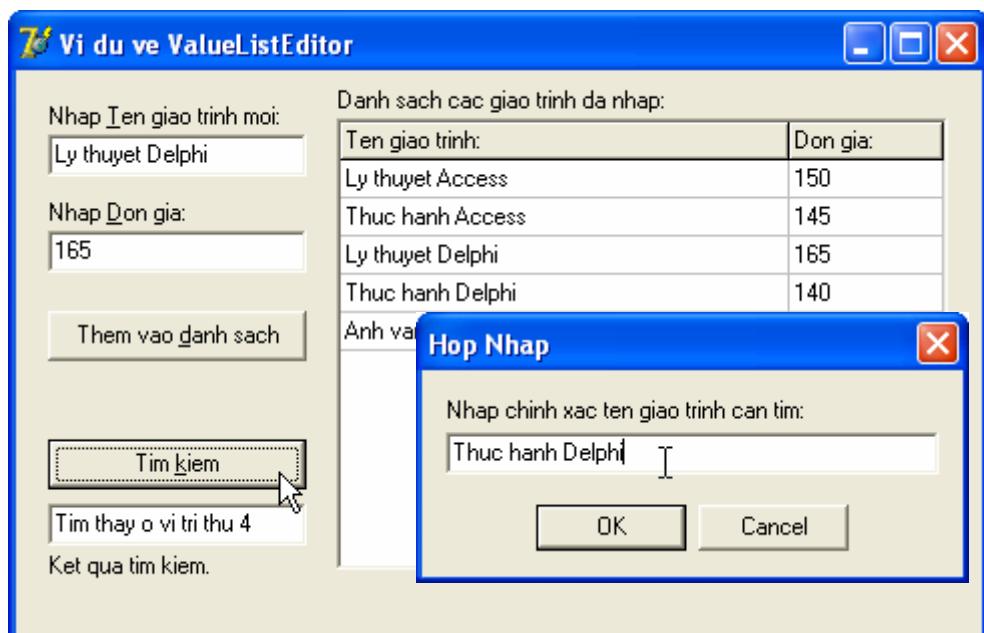


Hình 61: Thêm giáo trình "Anh văn căn bản"

- Nếu bạn nhập thêm vào tên giáo trình đã có, ví dụ như: "Ly thuyet Delphi" thì sẽ nhận được thông báo như sau:



Hình 62: Nhập bị trùng



Hình 63: Tìm kiếm theo tên giáo trình

IV.13. Hộp liệt kê thả (TComboBox)

Là thành phần liệt kê một danh sách các phần tử thả xuống. Các phần tử này **có thể** được nhập trực tiếp hoặc **không** thể nhập trực tiếp khi chạy chương trình là tùy thuộc vào kiểu (Style) mà bạn thiết lập nó ở quá trình thiết kế.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng hộp liệt kê thả.
AutoDropDown	Có giá trị True hoặc False: Nút liệt kê thả tự động thả xuống hay không khi nhấn một phím.
Items	Cho phép bạn nhập các phần tử vào ComboBox lúc thiết kế thông qua cửa sổ “String List Editor”.
Items[i]	Phần tử ở chỉ số thứ i trong ComboBox. Chỉ số i bắt đầu từ 0 và tiếp tục là 1, 2,... trong hộp liệt kê thả. Cách này thường được dùng viết mã lệnh.
ItemIndex	Xác định chỉ số phần tử được chọn, chỉ số này có giá trị bắt từ 0, 1, 2,... Nếu ItemIndex = -1 thì chưa có phần tử nào được chọn.
Items.Count	Tổng số các phần tử có trong hộp liệt kê thả.
DropDownCount	Số lượng các phần tử được liệt kê trong hộp liệt kê thả tại mỗi thời điểm.
Style	Có 2 giá trị mà bạn cần phân biệt đó là: - csDropDown Cho phép bạn nhập trực tiếp thêm một phần tử mới vào trong TComboBox. - csDropDownList chỉ cho bạn chọn từ danh sách các phần tử có sẵn trong ComboBox.
Text	Xác định chuỗi văn bản (hay phần tử) đang hiển thị trong TComboBox
Sorted	Có giá trị True hoặc False: Sắp xếp các phần tử theo thứ tự chữ cái hoặc không sắp xếp.
Visible	Có giá trị True hoặc False: Có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về TComboBox khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà ComboBox đó được nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 50: Các thuộc tính của TComboBox

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng

Tên sự kiện	Đáp ứng của sự kiện
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 51: Các sự kiện của TComboBox

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Items.Add(const S: string): Integer;	Hàm thêm phần tử kiểu string vào phía cuối và trả về chỉ số của phần tử này trong Items.
Items.Append(const S: string);	Thủ tục nối thêm phần tử kiểu string vào phía cuối Items.
Items.Delete(Index: Integer);	Thủ tục xóa một phần tử ở chỉ số index trong Items.
Items.Insert(Index: Integer; const S: string);	Thủ tục xóa một phần tử kiểu string tại chỉ số index trong Items. Thủ tục này tương đương câu lệnh gán vào thuộc tính Items như sau: Items[index] := S;
Clear;	Thủ tục xóa hết các phần tử trong ListBox.

Bảng 52: Các phương thức của TComboBox

- Một số khái niệm được sử dụng trong ví dụ:

+ Câu lệnh rẽ nhánh: **case <biến> of**

Giá trị_i: công việc_i;

end;

+ Hàm Trunc(x): trả về phần số nguyên của một số lẻ thập phân x

Ví dụ 22: Máy tính đơn giản sử dụng thành phần ComboBox.

- Tạo một dự án mới File/New/Application

- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong ComboBox
BorderIcons	[]
Position	poDesktopCenter

Bảng 53: Thiết lập các thuộc tính TForm.

- Đặt các đối tượng lên form và thiết lập các thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
ComboBox	Name	cboToantu
	Items	+
		-
		*
		/
Edit thứ 1	Div	
	Mod	
	Style	csDropDownList
Edit thứ 2	ItemIndex	-1
	Name	edt_ToanHang1
	Text	0
Edit thứ 3	TabOrder	0
	Name	edt_ToanHang2
	Text	0
Edit thứ 4	TabOrder	1
	Name	Edt_Kq
	Text	0
Button	ReadOnly	True
	Name	btnKetThuc
	Caption	&Kết thúc
	và 4 Label với Caption như mô tả trong form.	

Bảng 54: Thiết lập các thuộc tính của các đối tượng

- **Lần lượt nhập vào các câu lệnh cho các thủ tục sự kiện như sau:**

```

unit untMaytinh;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TfrmMain = class(TForm)
    cboToantu: TComboBox;
    edtToanHang1: TEdit;
    edtToanHang2: TEdit;

```

```

edt_Kq: TEdit;
btnKetThuc: TButton;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
procedure cboToantuChange(Sender: TObject);
procedure btnKetThucClick(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
frmMain: TfrmMain;
implementation
{$R *.dfm}

procedure TfrmMain.cboToantuChange(Sender: TObject);
var a, b:extended;
begin
if (TryStrToInt(edtToanHang1.Text,a)) and
(TryStrToInt(edtToanHang2.Text,b)) then
case cboToantu.ItemIndex of
0: edt_Kq.Text := FloatToStr(a + b);
1: edt_Kq.Text := FloatToStr(a - b);
2: edt_Kq.Text := FloatToStr(a * b);
3: if b<>0 then
    edt_Kq.Text := FloatToStr(a / b)
else edt_Kq.Text := 'Loi!... Mau so bang khong.';
4: if b<>0 then
    edt_Kq.Text := IntToStr(Trunc(a) Div Trunc(b))
else edt_Kq.Text := 'Loi!... Mau so bang khong.';
5: if b<>0 then
    edt_Kq.Text := IntToStr(Trunc(a) Mod Trunc(b))
else edt_Kq.Text := 'Loi!... Mau so bang khong.';

```

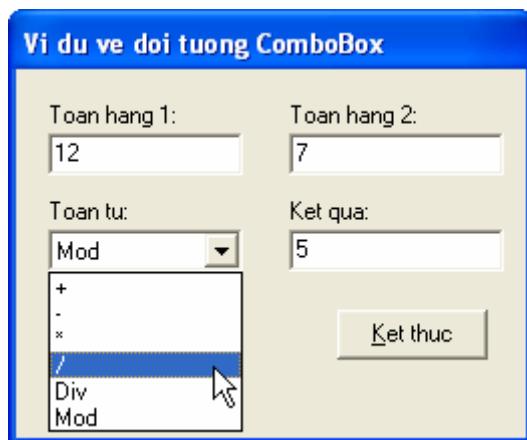
```

end // end case
else // else if
  edt_Kq.Text :='Du lieu nhap vao sai!...';
end;

procedure TfrmMain.btnKetThucClick(Sender: TObject);
begin
  Close;
end;
end.

```

- Lưu dự án vào thư mục S:\ViDuComboBox
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 64: Kết quả chương trình ComboBox

IV.14. Lưới chuỗi (TStringGrid)

Là thành phần được thiết kế theo dạng bảng gồm các ô chứa dữ liệu kiểu chuỗi. Các ô này được xác định bởi đánh chỉ số cột và chỉ số hàng. Trong TStringGrid, chỉ số cột đầu tiên (cột bên trái nhất) và hàng đầu tiên (hàng trên nhất) được đánh số mặc định là 1. Tuy nhiên, bạn có thể thay đổi chỉ số này về 0 để phù hợp chỉ số bắt đầu của mảng (Array) trong Delphi là 0 (zero) dựa vào các thuộc tính của TStringGrid được mô tả chi tiết dưới đây.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng StringGrid.
FixedCols	Xác định chỉ số cột đầu tiên (bên trái nhất) trong lưới, các giá trị này có thể bắt đầu từ 0, 1, 2,... Mặc định có giá trị là 1.
FixedRows	Xác định chỉ số hàng đầu tiên (hàng trên nhất) trong lưới, các giá trị

Tên thuộc tính	Ý nghĩa
	này có thể bắt đầu từ 0, 1, 2,... Mặc định có giá trị là 1.
ColCount	Xác định số cột trong lưới, số cột được đánh chỉ số bắt đầu từ 0 đến ColCount-1
RowCount	Xác định số hàng trong lưới, số hàng được đánh chỉ số bắt đầu từ 0 đến RowCount-1
Col	Chỉ số cột của ô hiện hành
Row	Chỉ số hàng của ô hiện hành
Cells[i,j]	Giá trị trong ô ở chỉ số cột i và hàng j
Options	Là các tùy chọn cho lưới, dưới đây trình bày là một số tùy chọn: - goEditing có giá trị True hoặc False, cho hay không phép soạn thảo trong các ô lưới. - goTabs có giá trị True hoặc False, cho hay không cho sử dụng phím Tab để di chuyển qua lại giữa các ô. - goRowSelect có giá trị True hoặc False, cho hay không phép chọn tất cả các ô trên toàn bộ một hàng.
ScrollBars	Xác định số thanh cuộn cần hiển thị, các giá trị của nó như sau: - ssBoth: Hiển thị cả 2 thanh cuộn đứng và ngang - ssHorizontal: Hiển thị thanh cuộn ngang - ssVertical: Hiển thị thanh cuộn đứng - ssNone: Không hiển thị thanh cuộn nào cả
Visible	Có giá trị True hoặc False: Có xuất hiện hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến nó.
Hint	Dòng gợi ý ngắn gọn về StringGrid khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà StringGrid nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 55: Các thuộc tính TStringGrid

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi bạn Click vào đối tượng
OnContextPopup	Khi bạn RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi bạn DClick vào đối tượng
OnExit	Khi bạn đưa con nháy vừa rời khỏi đối tượng
OnKeyDown	Khi bạn gõ phím bất kỳ
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các

Tên sự kiện	Đáp ứng của sự kiện
	phím điều khiển như: F1, Ctrl, Shift,...
OnSelectCell	Khi bạn Click vào ô được chọn

Bảng 56: Các sự kiện TStringGrid

- Một số khái niệm được sử dụng trong ví dụ:

+ Câu lệnh bao nhóm: **with tên_đối_tượng do**

begin

các thuộc tính

và phương thức của đối tượng

end;

+ Câu lệnh lặp: **for i := trị_đầu to trị_cuối do công_viện;**

Nếu **trị_cuối >= trị_đầu** thực hiện **công_viện** với **(trị_cuối - trị_đầu +1)** lần.

Ví dụ 23: Hiển thị bảng 26 ký tự chữ cái. Click vào ô tương ứng trong lưới để hiển thị ký tự trong ô bạn vừa chọn.

- Tạo một dự án mới **File/New/Application**

- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmAlphabet
Caption	Ví dụ về đối tượng TStringGrid
Position	poDesktopCenter

Bảng 57: Thiết lập các thuộc tính cho Form.

- Đặt các đối tượng lên form và thiết lập các thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
StringGrid	Name	stgAlphabet
	Name	lblAlphabet
Label	Caption	BANG 26 KY TU CHU CAI
	Font.Size	12

Bảng 58: Thiết lập các thuộc tính cho các đối tượng

Đoạn mã lệnh chương trình:

```
unit untAlphabet;
```

```
interface
```

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, ExtCtrls, Grids, StdCtrls;

type

```
TfrmAlphabet = class(TForm)
  stgAlphabet: TStringGrid;
  Label1: TLabel;
  procedure FormActivate(Sender: TObject);
  procedure stgAlphabetSelectCell(Sender: TObject; ACol, ARow: Integer;
    var CanSelect: Boolean);
```

private

{ Private declarations }

public

{ Public declarations }

end;

var

frmAlphabet: TfrmAlphabet;

implementation

{\$R *.dfm}

procedure TfrmAlphabet.FormActivate(Sender: TObject);

var i, j, k : Integer;

begin

k := 0;

with stgAlphabet **do**

begin

FixedCols:=0; // Cột trái nhất

FixedRows:=0; // Hàng trên nhất

ColCount:=5; RowCount:=6;

Options := Options+ [goEditing]; //cho phép cập nhật

for i := FixedCols **to** ColCount - 1 **do**

for j:= FixedRows **to** RowCount - 1 **do**

begin

k := k + 1;

if k <= 26 **then** Cells[i,j] := Chr(k+64);

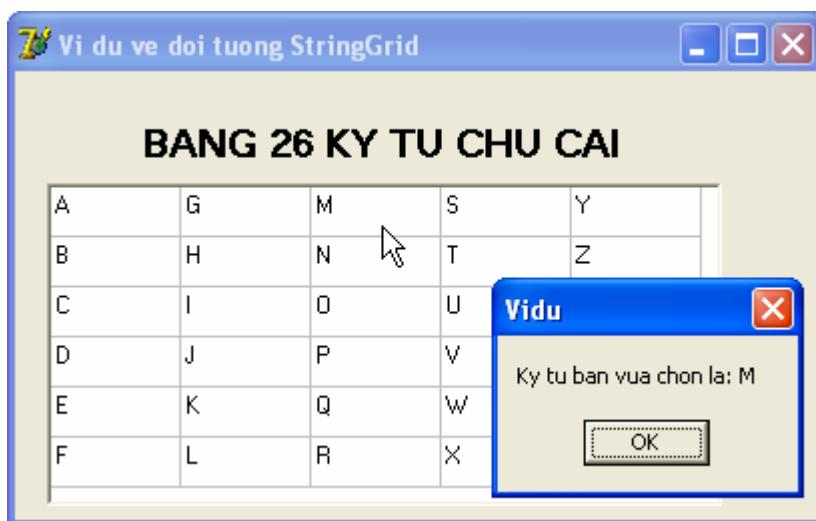
```

end;
end;
end;

procedure TfrmAlphabet.stgAlphabetSelectCell(Sender: TObject; ACol,
      ARow: Integer; var CanSelect: Boolean);
begin
  ShowMessage('Ky tu ban vua chon la: '+ stgAlphabet.Cells[ACol,ARow]);
  {với tham số ACol tương đương với thuộc tính Col, và ARow tương đương Row }
end;
end.

```

- Lưu dự án vào thư mục **S:\ViduStringGrid**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 65: Kết quả chương trình StringGrid

Ví dụ 24: Viết chương trình hiển thị 256 ký tự của bảng mã ASCII. Sử dụng bảng có 24 cột và 11 hàng; độ cao và độ rộng mặc nhiên của ô là 24.

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmBangASCII
Caption	Bang ma ASCII
Position	poDesktopCenter

Bảng 59: Thuộc tính của frmBangASCII

- Đặt các đối tượng lên form và thiết lập các thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
StringGrid	Name	grdASCII
	DefaulColWidth	24
	DefaulRowHeight	124
	FixedCols	0
	FixRows	0

và đối tượng nhãn Label.

Bảng 60: Thuộc tính các đối tượng trên form

- Đoạn mã lệnh chương trình:

```

unit untBangASCII;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, Grids, StdCtrls;
type
  TfrmBangASCII = class(TForm)
    grdASCII: TStringGrid;
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```



```

var
  frmBangASCII: TfrmBangASCII;

implementation

{$R *.dfm}

procedure TfrmBangASCII.FormCreate(Sender: TObject);
var i,k,j:integer;
begin
  k:=0;
  for i:=0 to grdASCII.ColCount-1 do
    for j:=0 to grdASCII.RowCount-1 do
      begin
```

Chương 5: Lập trình xử lý sự kiện – Các thành phần của Delphi

```

if k<=256 then grdASCII.Cells[i,j]:=chr(k);
k:=k+1;
end;
end;

end.

```

- Lưu dự án vào thư mục S:\ViduStringGrid2
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 66: Hiển thị bảng mã ASCII

IV.15. Bảng chứa các thành phần (TPanel)

Thường được dùng để nhóm các thành phần khác nhau cùng đặt bên trong nó để gom lại thành nhóm riêng biệt cho dễ quản lý, lập trình và thao tác.

* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho đối tượng Panel.
Align	Xác định việc canh lề trái, phải, trên, dưới,... của Panel bên trong thành phần chứa nó. Nếu có giá trị bằng alClient thì kích thước Panel sẽ chiếm toàn bộ bề mặt của thành phần chứa nó.
Alignment	Canh trái (taLeftJustify), canh phải (taRightJustify), canh giữa (taCenter) tiêu đề.
Caption	Tiêu đề của TPanel

Tên thuộc tính	Ý nghĩa
BorderStyle	Không kẻ khung (bsNone), có kẻ khung (bsSingle)
Visible	Có giá trị True hoặc False: Có xuất hiện Panel và các thành phần bên nó hay không khi chạy chương trình.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến các thành phần mà nó chứa hay không.
Hint	Dòng gợi ý ngắn gọn về Panel khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.
TabOrder	Được đánh thứ tự bắt đầu từ 0, 1, 2,... để xác định thứ tự mà Panel đó được nhận Focus khi bạn nhấn phím Tab. Chức năng này chỉ có tác động khi thuộc tính TabStop có giá trị True.

Bảng 61: Các thuộc tính của TPanel

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi người sử dụng Click vào đối tượng
OnContextPopup	Khi người sử dụng RClick vào đối tượng, hoặc thực hiện việc bật lên menu đối tượng.
OnDblClick	Khi người sử dụng DClick vào đối tượng
OnExit	Khi đối tượng mất tiêu điểm (Lost Focus)
OnKeyDown	Khi có phím bất kỳ được nhấn
OnKeyPress	Gần như sự kiện OnKeyDown, nhưng không đáp ứng cho các phím điều khiển như: F1, Ctrl, Shift,...

Bảng 62: Các sự kiện của TPanel

* Một số phương thức (Method) thường dùng:

Cú pháp của phương thức	Ý nghĩa
Show;	Thủ tục hiển thị đối tượng khi chạy chương trình.
Hide;	Thủ tục ẩn đối tượng khi chạy chương trình.

Bảng 63: Các phương thức của TPanel

IV.16. Thanh thực đơn chính (TMainMenu)

Thường được sử dụng khi chương trình của bạn có nhiều chức năng khác nhau. Để giúp cho người sử dụng dễ dàng sử dụng chương trình hơn, người ta thường thiết kế một thanh thực đơn chính. Thanh thực đơn này bao gồm nhiều thực đơn (Menu Item) ghép lại.

* Biểu tượng: 

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho thực đơn chính, tên này được đặt như tên biến và được sử dụng trong việc viết mã lệnh của chương trình và gắn kết vào các đối tượng khác (nếu có).
Items	Cho phép bạn nhập các thực đơn (MenuItem) vào MainMenu tại thời điểm thiết kế thông qua chức năng “Menu Designer”. Bạn cũng có thể RClick, DClick vào biểu tượng MainMenu hoặc tên đối tượng ở trong cửa sổ Object TreeView để thiết kế thanh thực đơn này. Bạn chú ý là mỗi thực đơn được tạo ra bởi thành phần TMenuItem được dẫn xuất từ lớp cha (Ancestor class) là TComponent. Lớp TMenuItem này được trình bày ở phần tiếp theo.
Images	Danh sách các hình ảnh (đông qua thành phần TImageList trên tab Win32) được gắn vào các thực đơn của thanh menu. Bạn cũng có thể gắn từng hình ảnh vào từng thực đơn thông qua thuộc tính Bitmap của mỗi thực đơn (TMenuItem)

Bảng 64: Các thuộc tính của TMainMenu

* **Cách sử dụng:** Thường một thanh thực đơn được đặt trong một cửa sổ chính (giao diện chương trình). Để gắn TMainMenu này vào form chính (MainForm), bạn mở MainForm ở chế độ thiết kế, và từ thuộc tính Menu bạn Click chọn tên (name) của MainMenu cần gắn lên.

IV.17. Thực đơn (TMenuItem)

* Biểu tượng: Không có. Nó được tự động tạo ra bởi chức năng “Menu Designer” trong quá trình thiết kế thanh thực đơn chính (Main Menu).

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho thực đơn, tên này được đặt như tên biến và được sử dụng trong việc viết mã lệnh của chương trình.
AutoCheck	Có giá trị True thì lúc bạn chạy chương trình và Click vào thực đơn thì Delphi sẽ tự động bật/tắt dấu Check lên phía trước của thực đơn; ngược lại thì có giá trị False.
Checked	Có giá trị True và False: Bạn đánh dấu check vào thực đơn này ngay thời điểm thiết kế.
Bitmap	Gắn hình ảnh vào thực đơn
Caption	Tiêu đề của thực đơn
ImageIndex	Là chỉ mục hình ảnh trong danh sách hình của ImageList. Mặc nhiên có giá trị là -1 : không sử dụng hình ảnh trong ImageList.
Enabled	Có giá trị True hoặc False: Cho phép hay không người sử dụng truy xuất đến các thành phần mà nó chứa hay không.
Hint	Dòng gợi tả ngắn gọn về MenuItem khi bạn trỏ mouse đến nó. Chức năng này chỉ có tác động khi thuộc tính ShowHint là True.

Tên thuộc tính	Ý nghĩa
ShortCut	Xác định (tổ hợp) phím nóng cho thực đơn.
Visible	Có giá trị True hoặc False: Cho hiển thị hay không cho hiển thị TMenuItem.

Bảng 65: Các thuộc tính của TMenuItem

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnClick	Khi bạn Click chọn vào thực đơn lúc chạy chương trình.

Bảng 66: Sự kiện của TMenuItem

Ví dụ 25: Chương trình tiện ích

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của main form (Giao diện chính) như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong MainMenu
Position	po MainFormCenter

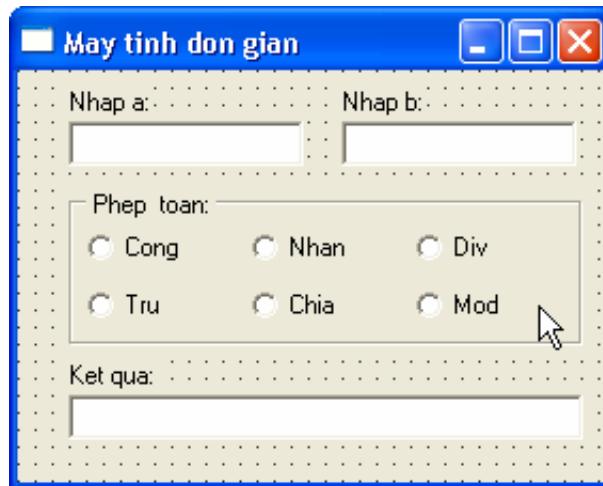
Bảng 67: Thiết lập các thuộc tính cho form frmMain

- Đặt các thành phần lên form và thiết lập thuộc tính như sau. Chú ý TMainMenu chỉ có 1 thực đơn (TMenuItem) là **Tien ich**, trong thực đơn **Tien ich** thì có 4 thực đơn cấp con (TMenuItem) là **May tinh**, **Xem ngay gio**, **Đường phân cách** (dấu -) và **Thoat**

Đối tượng	Thuộc tính	Giá trị				
MainMenu	Name	mmuMainForm				
		Có 5 TMenuItem như sau:				
	Items	Thứ	Name	Caption	Shortcut	Bitmap
		1	mniTienIch	&Tien ich		
		2	mniMayTinh	&May tinh	Ctrl+M	Calculator.bmp
		3	mniXemngaygio	Xem &ngay gio	Ctrl+D	Clock.bmp
		4	mniN1	-		
		5	mniThoat	&Thoat	Ctrl+Q	Close.bmp

Bảng 68: Thiết lập các thuộc tính cho các đối tượng trên Form

- Tạo thêm form “May tinh don gian” như hình sau:



Hình 67: Thiết kế form frmMayTinh

Thuộc tính	Giá trị
Name	frmCalculator
Caption	May tinh don gian

Bảng 69: Thiết lập các thuộc tính frmMayTinh

Đối tượng	Thuộc tính	Giá trị
RadioGroup	Name	rgpPhepToan
	Items	Cong Tru Nhan Chia Div Mod
	Name	edt_a
	Name	edt_b
	Name	edt_Kq
		và một số nhãn theo form.

Bảng 70: Các đối tượng trên frmMayTinh

- Tạo thêm form “Ngay gio hien hanh” như hình và các thuộc tính sau:



Hình 68: Thiết kế form frmDateTime

Đối tượng	Thuộc tính	Giá trị
Form	Name	frmDateTime
	Caption	Ngay gio hien hanh
	Position	poMainFormCenter
Label	Name	lblDateTime

Bảng 71: Thiết lập các thuộc tính frmDateTime và lblDateTime

- Đoạn code của chương trình unit file **untMain** như sau:

```

unit untMain;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, Menus;

type
    TfrmMain = class(TForm)
        mmuMainForm: TMainMenu;
        mniTienIch: TMenuItem;
        mniMayTinh: TMenuItem;
        mniXemngaygio: TMenuItem;
        mniN1: TMenuItem;
        mniThoat: TMenuItem;
        procedure mniMayTinhClick(Sender: TObject);
        procedure mniXemngaygioClick(Sender: TObject);
        procedure mniThoatClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmMain: TfrmMain;

implementation

uses untMaytinh,untDateTime;
{$R *.dfm}

procedure TfrmMain.mniMayTinhClick(Sender: TObject);
begin
    frmCalculator.ShowModal;
end;

```

Chương 5: Lập trình xử lý sự kiện – Các thành phần của Delphi

```
procedure TfrmMain.mniXemngaygioClick(Sender: TObject);
begin
  frmDateTime.ShowModal;
end;

procedure TfrmMain.mniThoatClick(Sender: TObject);
begin
  Close;
end;

end.
```

- Đoạn code của chương trình unit file **untMayTinh** như sau:

```
unit untMayTinh;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TfrmCalculator = class(TForm)
    rgpPhepToan: TRadioGroup;
    edt_a: TEdit;
    edt_b: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    edt_Kq: TEdit;
    Label3: TLabel;
    procedure rgpPhepToanClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmCalculator: TfrmCalculator;

implementation
{$R *.dfm}

procedure TfrmCalculator.rgpPhepToanClick(Sender: TObject);
var a, b: extended;
begin
```

```

if TryStrToFloat(edt_a.Text,a) and TryStrToFloat(edt_b.Text,b)
then
begin
    Case rgpPhepToan.ItemIndex of
        0: edt_Kq.Text := FloatToStr(a + b);
        1: edt_Kq.Text := FloatToStr(a - b);
        2: edt_Kq.Text := FloatToStr(a * b);
        3: if b<> 0 then edt_Kq.Text := FloatToStr(a / b)
            else edt_Kq.Text := 'Loi! Phep toans chia 0.';
        4: edt_Kq.Text := FloatToStr(Trunc(a) DIV Trunc(b));
        5: edt_Kq.Text := FloatToStr(Trunc(a) MOD Trunc(b));
    end;      // end case
end          // end begin
else ShowMessage('Loi! Du lieu nhap sai.');
end;

procedure TfrmCalculator.FormActivate(Sender: TObject);
begin
    edt_a.Clear;
    edt_b.Clear;
    edt_Kq.Clear;
end;

end.

```

- Đoạn code của chương trình unit file **untDateTime**; như sau:

```

unit untDateTime;

interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmDateTime = class(TForm)
        lblDateTime: TLabel;
        procedure FormActivate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmDateTime: TfrmDateTime;

```

```

implementation

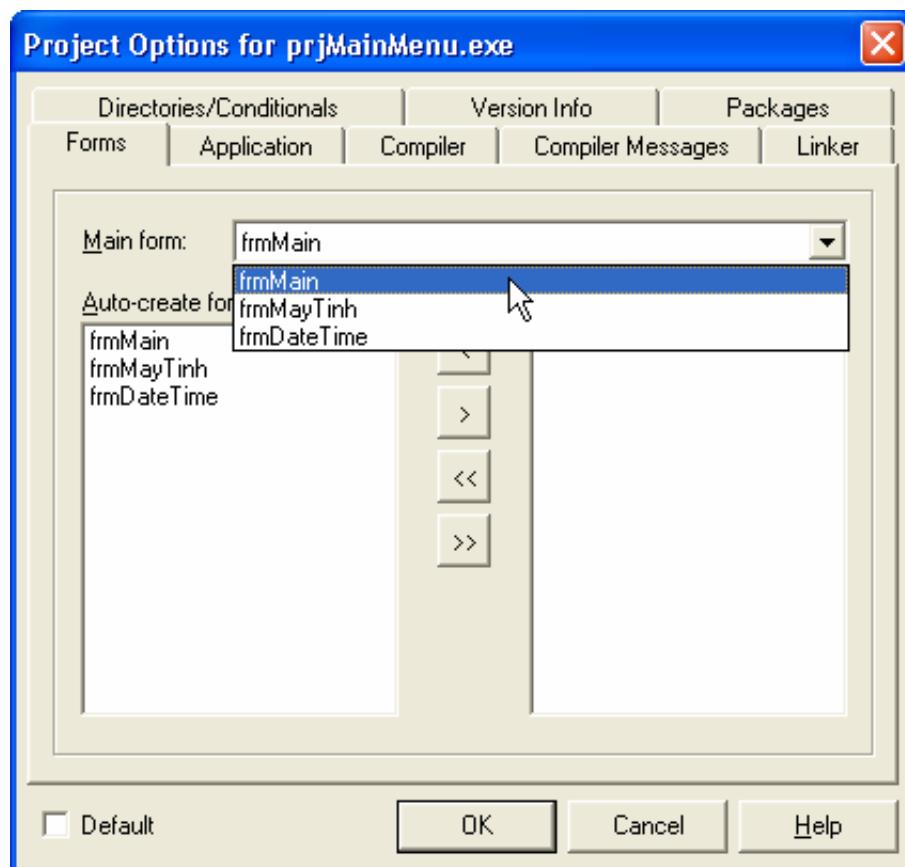
{$R *.dfm}

procedure TfrmDateTime.FormActivate(Sender: TObject);
begin
  lblDateTime.Font.Color := clBlue;
  lblDateTime.Font.Size:=15;
  lblDateTime.Alignment:=taCenter;
  lblDateTime.Caption:=DateToStr(Date)+ Chr(32)+ TimeToStr(time);
end;

end.

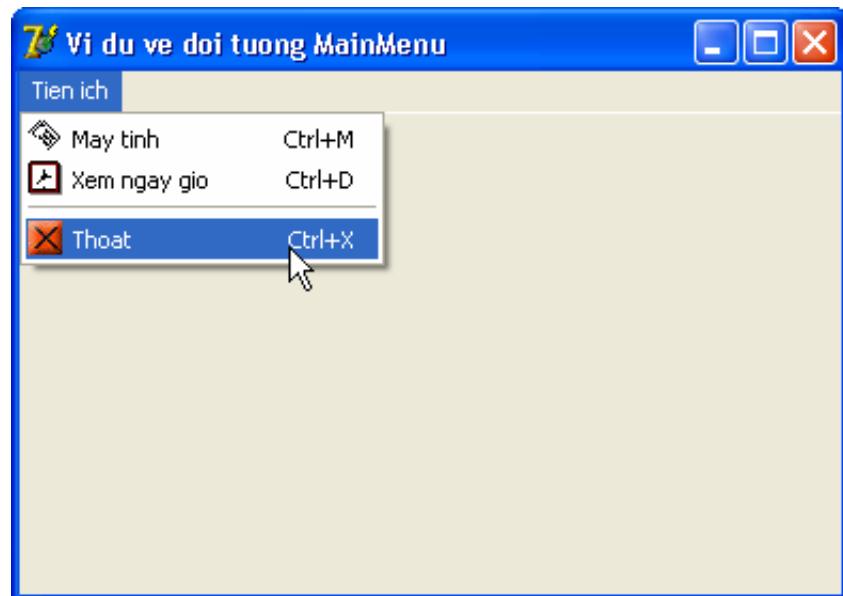
```

- Vào chức năng **Project/Options/tab Forms** chọn Main Form là frmMain như hình:



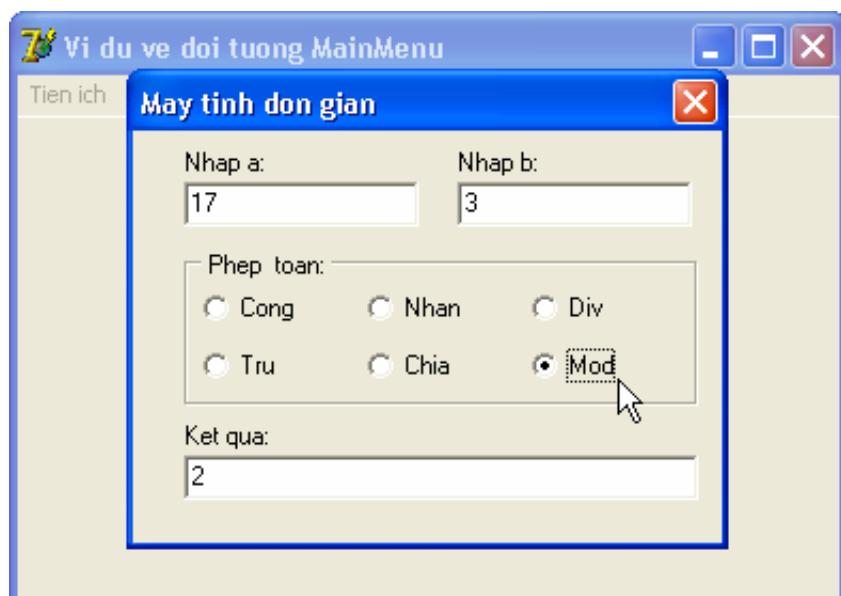
Hình 69: Chọn frmMain làm form chính.

- Lưu dự án vào thư mục S:\ViduMainMenu
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 70: Kết quả chương trình của frmMain

- Click chọn chức năng **May tinh**, ta có hình như sau:



Hình 71: Hiển thị form frmMayTinh

- Click chọn chức năng **Xem ngay gio**, ta có hình như sau:



Hình 72: Hiển thị form frmDateTime

Ở ví dụ trên, chúng ta sử dụng các form bình thường (fsNormal) để tạo giao diện chính (Main form). Tuy nhiên, khi bạn đã khá hiểu về Delphi thì có một công cụ hữu ích là

MDIForm (Multi Document Interface) để tạo giao diện chính và quản lý dự án của bạn hiệu quả và chuyên nghiệp hơn.

IV.18. Menu đối tượng (TPopupMenu)

Còn được gọi là trình đơn ngữ cảnh, hay menu tắt. Chức năng thường được sử dụng vào mục đích bật một menu tại bất kỳ nơi nào trên đối tượng chứa nó khi bạn sử dụng thao tác RClick .



* Biểu tượng:

* Một số thuộc tính thường dùng:

Tên thuộc tính	Ý nghĩa
Name	Xác định tên cho menu đối tượng, tên này được đặt như tên biến và được sử dụng trong việc viết mã lệnh của chương trình và gắn kết vào các đối tượng khác (nếu có).
Alignment	Vị trí xuất hiện của TPopupMenu ở góc trên bên trái (paLeft), góc trên bên phải (paRight), góc trên ở giữa (paCenter) của mouse.
AutoPopup	Có giá trị True và False: Tự động bật hay không khi người sử dụng RClick.
Items	Cho phép bạn nhập các thực đơn (MenuItem) vào PopupMenu tại thời điểm thiết kế thông qua chức năng “Menu Designer”. Bạn cũng có thể Rclick, DClick vào thành phần TMainMenu hoặc tên đối tượng ở trong cửa sổ Object TreeView để thiết kế thanh thực đơn này. Mỗi thực đơn trong PopupMenu được tạo ra bởi thành phần TMenuItem như đã trình bày ở phần trên.
Images	Danh sách các hình ảnh (đông qua thành phần ImageList trên tab Win32) được gắn vào các thực đơn của thanh thực đơn. Bạn cũng có thể gắn từng hình ảnh vào từng thực đơn thông qua thuộc tính Bitmap của mỗi thực đơn (TMenuItem)

Bảng 72: Các thuộc tính của TPopupMenu

* Một số sự kiện thường dùng:

Tên sự kiện	Đáp ứng của sự kiện
OnPopup	Khi bạn RClick, chú ý rằng sự kiện này sẽ xảy ra trước khi PopupMenu bật lên.

Bảng 73: Sự kiện của TPopupMenu

Ví dụ 26: Sử dụng TPopupMenu để thay đổi màu nền của form:

- Tạo một dự án mới **File/New/Application**
- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmMain
Caption	Vi du ve doi tuong PoupMenu
PopupMenu	{Được xác lập ở bước sau}

Bảng 74: Thiết lập các thuộc tính cho TForm

- Đặt các đối tượng lên form và thiết lập các thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị		
PopupMenu	Name	pmuMauSac		
	Có 6 TMenuItem như sau:			
	Items	Thứ	Name	Caption
		1	mniMauXanh	Mau &Xanh
		2	mniMauDo	Mau &Do
		3	mniMauTim	Mau &Tim
		4	mniMauVang	Mau &Vang
		5	mniN1	-
		6	mniMauBandau	Mau &Ban dau

Bảng 75: Thiết lập các thuộc tính cho các đối tượng trên form.

- Trở lại thuộc tính PopupMenu của form **frmMain**, bạn gắn **pmuMauSac** vào form.

- Đoạn code của chương trình như sau:

```
unit untPopup;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, Menus;
type
  TfrmPopup = class(TForm)
    pmuMauSac: TPopupMenu;
    mniMauXanh: TMenuItem;
    mniMauDo: TMenuItem;
    mniMauTim: TMenuItem;
    mniMauVang: TMenuItem;
    mniN1: TMenuItem;
    mniMauBandau: TMenuItem;
    procedure mniMauXanhClick(Sender: TObject);
    procedure mniMauDoClick(Sender: TObject);
    procedure mniMauTimClick(Sender: TObject);
  end;
```

```
procedure mniMauVangClick(Sender: TObject);
procedure mniMauBandauClick(Sender: TObject);

private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmPopup: TfrmPopup;

implementation
{$R *.dfm}

procedure TfrmPopup.mniMauXanhClick(Sender: TObject);
begin
  frmPopup.Color := clBlue;
end;

procedure TfrmPopup.mniMauDoClick(Sender: TObject);
begin
  frmPopup.Color := clRed;
end;

procedure TfrmPopup.mniMauTimClick(Sender: TObject);
begin
  frmPopup.Color := clPurple;
end;

procedure TfrmPopup.mniMauVangClick(Sender: TObject);
begin
  frmPopup.Color := clYellow;
end;

procedure TfrmPopup.mniMauBandauClick(Sender: TObject);
begin
  frmPopup.Color := clBtnFace;
end;

end.
```

- Lưu dự án vào thư mục **S:\ViduPopup**
- Biên dịch và chạy chương trình, RClick lên Form, ta có kết quả như sau:



Hình 73: Kết quả chương trình PopupMenu

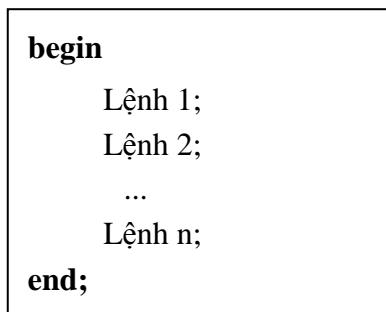
CHƯƠNG 6: CÁC LỆNH CÓ CẤU TRÚC

I. Lệnh ghép (Compound statement)

Một nhóm câu lệnh đơn được đặt giữa từ khoá **begin ... end;** sẽ tạo thành một câu lệnh ghép.

Trong Pascal ta có thể đặt các lệnh ghép con trong các lệnh ghép lớn hơn bao ngoài của nó và có thể hiểu tương tự như cấu trúc ngoặc đơn () trong các biểu thức Toán học.

* **Sơ đồ:**



Hình 1: Lệnh ghép **begin .. end;**

Ở hình minh họa trên ta thấy các lệnh được nhóm lại thành từng khối (block). Một khối lệnh bắt đầu bằng **begin** và chấm dứt ở **end;** Trong một khối lệnh cũng có thể có các khối lệnh con nằm trong nó. Một khối chương trình thường được dùng nhóm từ 2 lệnh đơn trở lên để tạo thành một **Công_viện** của các lệnh có cấu trúc, bạn sẽ gặp khái niệm này trong các ví dụ ở các phần sau.

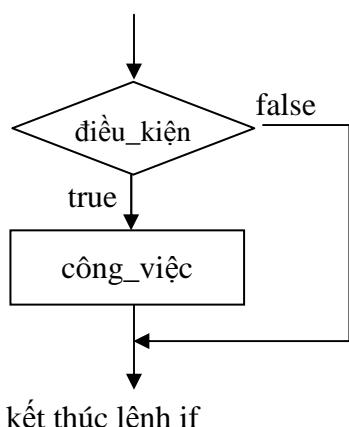
II. Lệnh cấu trúc rẽ nhánh

Lệnh có cấu trúc rẽ nhánh được sử dụng trong trường hợp chương trình chỉ cần chọn một trong các công việc để thực hiện tại một thời điểm.

II.1. Lệnh if ... then ... và lệnh if ... then ... else..

II.1.1. Lệnh if ... then

* Lưu ý: diễm tả các lệnh và ý nghĩa:



* **Cú pháp:**

if <điều_kiện> **then** công_viện;

* **Ý nghĩa:**

- ◆ Nếu điều_kiện có giá trị đúng thì sẽ thực hiện công_viện rồi kết thúc lệnh.
- ◆ Ngược lại (điều_kiện có giá trị sai) thì sẽ bỏ qua công_viện và kết thúc lệnh.

Hình 2: Lệnh if ... then ..;

Ví dụ 1: Đoạn chương trình sau mô tả cách thực hiện lệnh if ... then

begin

{ Lệnh mô tả một câu lệnh if ..then đơn giản với điều kiện luôn có giá trị True và chương trình thực hiện công việc chỉ có 1 lệnh đơn }

if true then

```
ShowMessage( ' Dieu kien dung.' );
```

// Lệnh này giống như lệnh trên nhưng chương trình thực hiện công việc bao gồm 2 lệnh

if (1 = 1) or (true >= false) then // Điều kiện có trị đúng

begin //bắt đầu lệnh ghép

```
ShowMessage( ' Dieu kien dung ' );
```

```
ShowMessage( ' Chao cac ban den voi Delphi - phiên bản 7.0' );
```

end; //hết lệnh ghép

if 'Delphi' = 'Java' then // Điều kiện có giá trị sai

```
ShowMessage( ' Oh, dieu kien sai. Lenh se bi bo qua ' );
```

end;

II.1.2. Lệnh if ... then ... else

*** Cú pháp:**

if <điều kiện> then

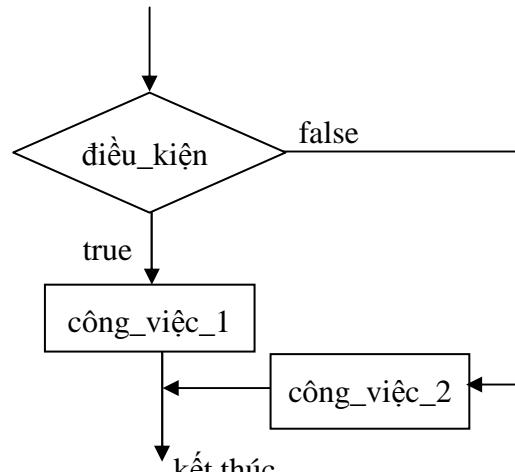
công_việc_1

else

công_việc_2;

***Ý nghĩa:**

- ◆ Nếu điều kiện đúng thì thực hiện công việc 1 (không thực hiện công việc 2).
- ◆ Ngược lại (điều kiện sai) thì sẽ thực hiện công việc 2 (không thực hiện công việc 1).



Hình 3: lệnh if ... then ... else

Chú ý:

- Điều kiện là một biểu thức Boolean (có giá trị đúng: True hoặc sai: False).
- Nếu công việc sau then hoặc sau else có nhiều hơn một lệnh thì ta phải gói lại trong lệnh **begin ... end**;
- Toàn bộ lệnh **if ... then ... else** xem như 1 lệnh đơn.
- Lệnh đứng trước từ khóa **else** thì không có dấu chấm phẩy (;) ở cuối lệnh.

Ví dụ 2: Đoạn chương trình sau mô tả cách thực hiện lệnh if ... then ... else

```

var x: single;
begin
  if 4>=4 then
    begin
      x := 2*sin(PI/2)/sqrt(9);
      ShowMessage( ' x = ' + FormatFloat('0.000', x));
    end //không có dấu ;
  else ShowMessage( 'Dieu kien sai');
  // Kết quả hiển thị giá trị biến x = 0.667

  // Các lệnh IF lồng vào nhau
  if true Then
    if false Then ShowMessage(' I ')
    else ShowMessage(' LOVE ')
  else ShowMessage(' DELPHI ');
end;
// Kết quả hiển thị từ: LOVE

```

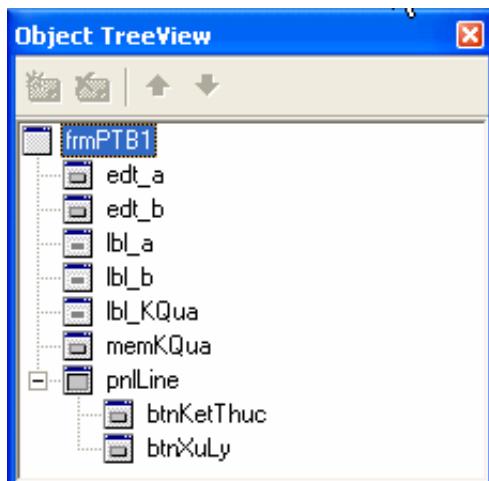
Ví dụ 3: Viết chương trình giải phương trình bậc nhất $ax + b = 0$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**

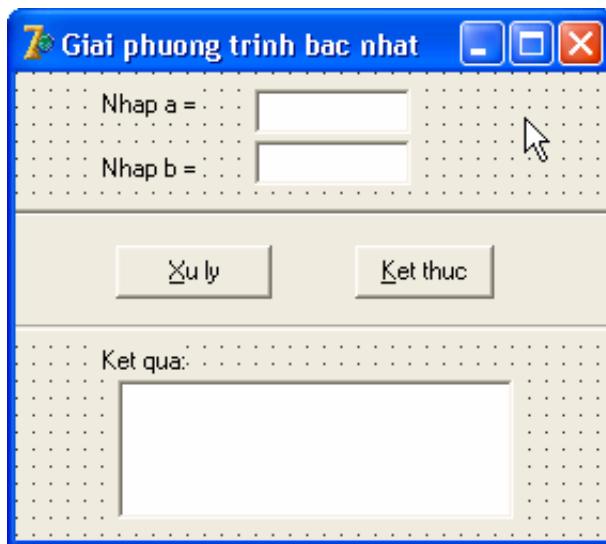
Lưu ý: Do các bạn đã học xong chương 5, nên giáo trình sẽ trình bày cách viết chương trình một cách ngắn gọn hơn. Bạn phải tự xác định và thiết lập các thuộc tính cho các đối tượng trong chương trình sao cho thích hợp nhất theo ý của mình. Giáo trình chỉ đưa ra các tên (name) của đối tượng để viết code cho dự án mà thôi.

- Đặt thuộc tính name cho các đối tượng của chương trình như sau:



Hình 4: Tên (name) của các đối tượng

- Thiết kế giao diện trên form:



Hình 5: Giao diện phương trình bậc nhất

- Đoạn code của chương trình như sau:

Lưu ý: Để viết chương trình cho chính xác, bạn cần phải xác định đúng thủ tục sự kiện mà bạn xác định là của **đối tượng nào ứng với sự kiện gì trên form**. Bạn có thể xem lại trong phần “Cách hình thành thủ tục sự kiện” ở chương 5.

```

unit untPtb1; //tên untPtb1 là do lưu unit file với tên untPtb1.pas
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TfrmPTB1 = class(TForm)
    lbl_a: TLabel;
    lbl_b: TLabel;
    edt_a: TEdit;
    edt_b: TEdit;
    lbl_KQua: TLabel;
    memKQua: TMemo;
    pnLine: TPanel;
    btnXuLy: TButton;
    btnKetThuc: TButton;
    procedure btnXuLyClick(Sender: TObject);
    procedure btnKetThucClick(Sender: TObject);
  end;

```

```
private
  { Private declarations }

public
  { Public declarations }

end;

var
  frmPTB1: TfrmPTB1;

implementation
{$R *.dfm}

procedure TfrmPTB1.btnXuLyClick(Sender: TObject);
var a, b, x: extended;
begin
  memKQua.Lines.Clear;
  if (TryStrToFloat(edt_a.Text, a)=true) and
    (TryStrToFloat(edt_b.Text, b)=true) then
  begin
    if a=0 then
      if b=0 then
        memKQua.Lines.Text:='Phuong trinh vo so nghiem'
      else
        memKQua.Lines.Text:='Phuong trinh vo nghiem'
    else //a <>0
      begin
        memKQua.Lines.Add('Phuong trinh co nghiem:');
        x:=-b/a;
        memKQua.Lines.Add('          x= '+FloatToStr(x));
      end;
  end
  else
  begin
    memKQua.Lines.Text:='Nhap sai! Nhap lai he so a va b.';
    edt_a.Clear;  edt_b.Clear;
    edt_a.SetFocus;
  end;
end;
```

```
procedure TfrmPTB1.btnExitClick(Sender: TObject);
begin
  Close;
end;
end.
```

- Lưu dự án vào thư mục S:\ViduPTB1
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 6: Chạy chương trình

Ví dụ 4: Giải phương trình $ax^2 + bx + c = 0$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 7: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit untPTB2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmPTB2 = class(TForm)
    edt_a: TEdit;
    edt_b: TEdit;
    edt_c: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    btnGiai: TButton;
    memKQ: TMemo;
    Label4: TLabel;
    btnNhaphLai: TButton;
    procedure btnGiaiClick(Sender: TObject);
    procedure btnNhaphLaiClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmPTB2: TfrmPTB2;

implementation
{$R *.dfm}

procedure TfrmPTB2.btnGiaiClick(Sender: TObject);
var a, b, c, delta, x1, x2: single;
  mOK1, mOK2, mOK3: boolean;
begin
  mOK1 := False;  mOK2 := False;  mOK3 := False;
  memKQ.Lines.Clear;
  mOK1 := TryStrToFloat(edt_a.Text,a);
  mOK2 := TryStrToFloat(edt_b.Text,b);
  mOK3 := TryStrToFloat(edt_c.Text,c);
  if (mOK1 = True) and (mOK2 = True) and (mOK3 = True) then
    //nhap du lieu dung

```

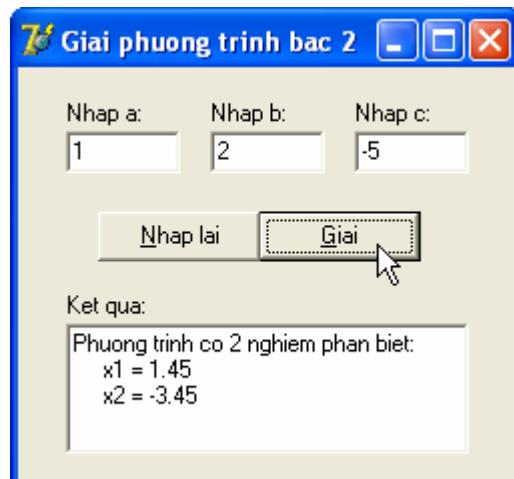
```

begin
    if a = 0 then // giao ptbl
        if b = 0 then
            if c = 0 then memKQ.Lines.Add('Phuong trinh Vo so nghiem')
            else memKQ.Lines.Add('Phuong trinh Vo nighiem')
        else memKQ.Lines.Add('Phuong trinh co nighiem la ' +
                                FloatToStr(c/b))
    else // a <> 0, bat dau giao ptb2
        begin
            delta := sqr(b) - 4*a*c;
            if delta < 0 then
                memKQ.Lines.Add('Phuong trinh Vo nighiem.')
            else
                if delta = 0 then
                    memKQ.Lines.Add('Phuong trinh co nighiem kep x1 = x2 =
                                    '+FloatToStr(-b/(2*a)))
                else
                    begin
                        x1 := (-b + sqrt(delta))/(2*a);
                        x2 := (-b - sqrt(delta))/(2*a);
                        memKQ.Lines.Add('Phuong trinh co 2 nighiem phan biet:');
                        memKQ.Lines.Add('      x1= '+FormatFloat('#,##0.00',x1));
                        memKQ.Lines.Add('      x2= '+FormatFloat('#,##0.00',x2));
                    end;
                end; // end else a<>0
            end // end if mOK1...
        else // else if: Nhập dữ liệu sai.
            ShowMessage('Ban nhap he so a, b, c KHONG hop le!...');
        end;

procedure TfrmPTB2.btnAddLaiClick(Sender: TObject);
begin
    edt_a.Clear;
    edt_b.Clear;
    edt_c.Clear;
    memKQ.Lines.Clear;
    edt_a.SetFocus;
end;
end.

```

- Lưu dự án vào thư mục **S:\ViduPTB2**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 8: Chạy chương trình

II.2. Lệnh Case ... of

* **Cú pháp, ý nghĩa:**

Case <bíểu_thức> of // Xét giá trị của biểu thức:

GT1 : công_viện_1 ; // Nếu biểu_thức = GT1 (giá trị 1) thì thực hiện công_viện_1,

GT2 : công_viện_2 ; // nếu biểu_thức = GT2 thì thực hiện công_viện_2,

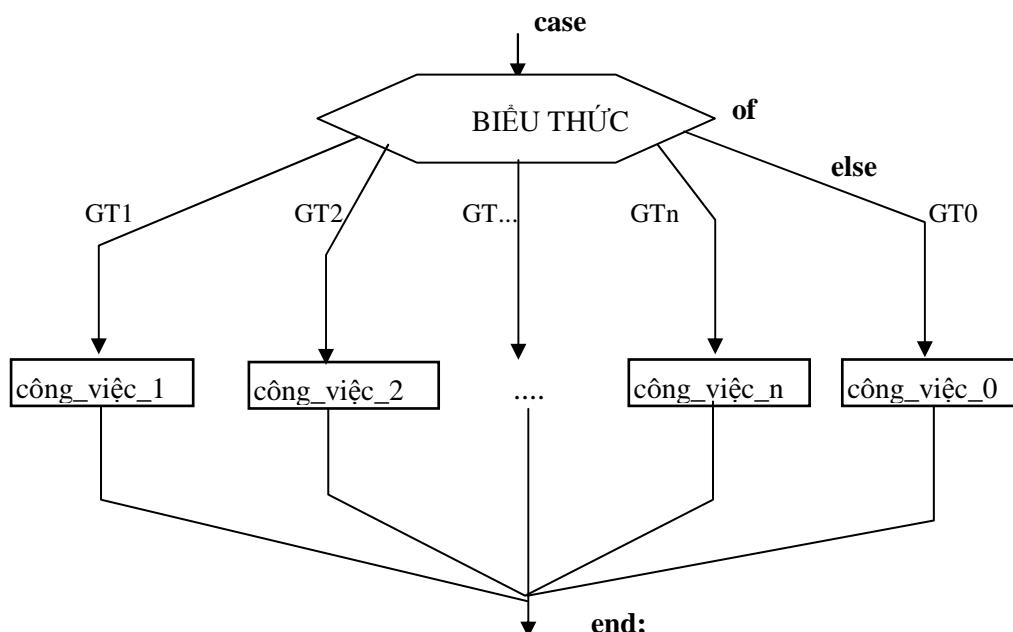
...

GTn : công_viện_n ; // nếu biểu_thức = GTn thì thực hiện công_viện_n

[else công_viện_0 ;] //Ngược lại sẽ thực hiện công_viện_0

End; //end case

* **Lưu đồ:**



Hình 9: Lưu đồ lệnh Case ... of ... end;

Chú ý:

- Lệnh **Case ... of** có thể không có **else**
- Biểu thức trong **Case... of** phải có kiểu dữ liệu rời rạc/dếm được như: Integer, Char,...
- Nếu muốn ứng với nhiều giá trị khác nhau của biểu thức mà máy vẫn thi hành một Công việc thì các giá trị này có thể viết trên cùng một hàng cách nhau bởi dấu phẩy (,):
 k_1, k_2, \dots, k_p : công việc;

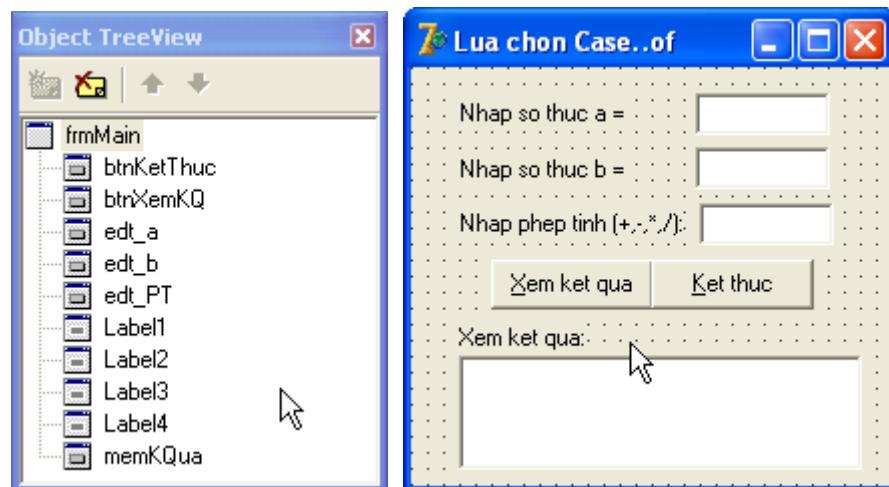
Ví dụ 5: Đoạn chương trình sau mô tả cách sử dụng lệnh **Case ... of**

```
var      color: integer ;
begin
  color := StrToInt(InputBox('Hop nhap lieu', 'Nhập số từ 1 đến 5','1'));
  case color of
    1, 2: ShowMessage('Mau do');
    3, 4: ShowMessage('Mau trang');
    5: ShowMessage('Mau xanh');
  end;
end;
```

Ví dụ 6: Viết chương trình nhập vào hai số thực a, b và một ký tự ch là phép tính cần thực hiện (+, -, *, /). Sau đó thực hiện phép tính và hiển thị kết quả lên màn hình.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 10: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit untCaseOf;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmMain = class(TForm)
    Label1: TLabel;
    edt_a: TEdit;
    Label2: TLabel;
    edt_b: TEdit;
    Label3: TLabel;
    edt_PT: TEdit;
    btnXemKQ: TButton;
    btnKetThuc: TButton;
    Label4: TLabel;
    memKQua: TMemo;
    procedure btnKetThucClick(Sender: TObject);
    procedure btnXemKQClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.btnXemKQClick(Sender: TObject);
var a, b:extended; ch:char;
begin
  memKQua.Lines.Clear;
  a:=StrToFloat(edt_a.Text);
  b:=StrToFloat(edt_b.Text);
  ch:=edt_PT.Text[1]; // lấy ký tự đầu tiên
  case ch of
    '+': memKQua.Lines.Add(' a+b= '+FloatToStr(a+b));
    '-': memKQua.Lines.Add(' a-b= '+FloatToStr(a-b));
    '*': memKQua.Lines.Add(' a*b= '+FloatToStr(a*b));
    '/': if b=0 then
      memKQua.Lines.Add('Loi! Phep chia 0.')
    else

```

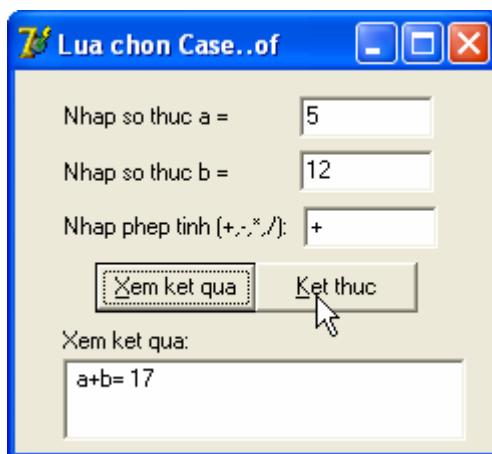
```

memKQua.Lines.Add(' a/b= '+FloatToStr(a/b));
else
    memKQua.Lines.Add('Phep tinh khong thuc hien duoc');
end;
end;

procedure TfrmMain.btnAddClick(Sender: TObject);
begin
    close;
end;
end.

```

- Lưu dự án vào thư mục S:\ViduCaseOf
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 11: Chạy chương trình

III. Cấu trúc lệnh lặp

Lệnh có cấu trúc lặp thường được sử dụng khi chương trình phải thực hiện công việc lặp đi lặp lại nhiều lần.

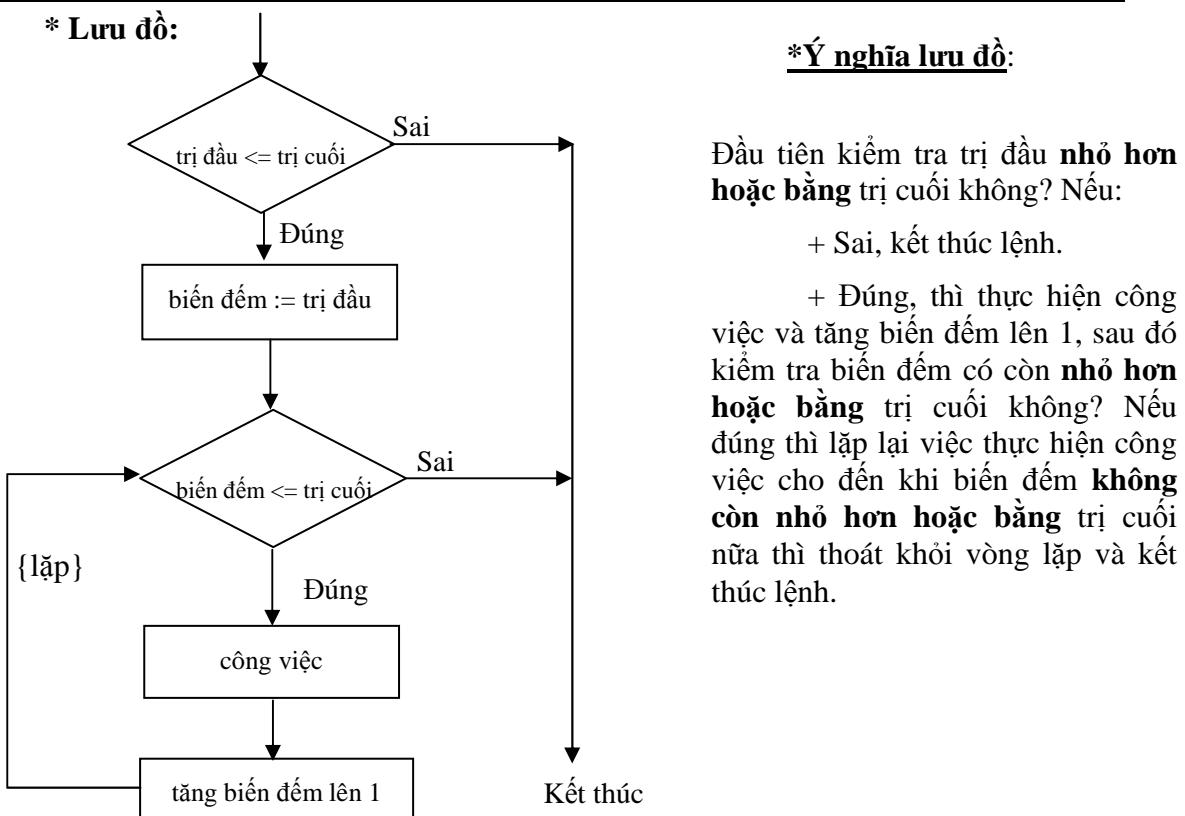
III.1. Lệnh lặp có số lần xác định trước

Lệnh lặp **for** cho phép chương trình thực hiện công việc nhiều lần và với số lần lặp đã biết trước. Lệnh **for** có 2 dạng:

For ... to ... do : đếm lên,
và **For ...ownto ... do** : đếm xuống.

* Cú pháp đếm lên:

for <biến_đếm := trị_đầu> to <trị_cuối> do công_việc;



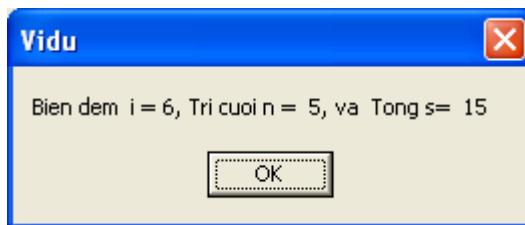
Hình 12: Lưu đồ lệnh lặp for ... to ... do

Chú ý: Biến đếm, trị đầu, trị cuối là kiểu dữ liệu đếm được (rời rạc) như: số nguyên, Char, kiểu liệt kê (Enum).

Ví dụ 7: Tính tổng $s = 1 + 2 + 3 + 4 + 5$.

```

var i, s : integer;
begin
    s := 0; //Khởi tạo biến tổng
    n:= 5;
    for i:=1 to n do s := s + i;
    ShowMessage( 'Bien dem i = '+IntToStr(i) + ', Tri cuoi n = '+IntToStr(n)
        + ', va tong s= '+ IntToStr(s));
end;
  
```



Hình 13: Kết quả chương trình tính tổng

Ví dụ 8: Biến đếm là kiểu liệt kê

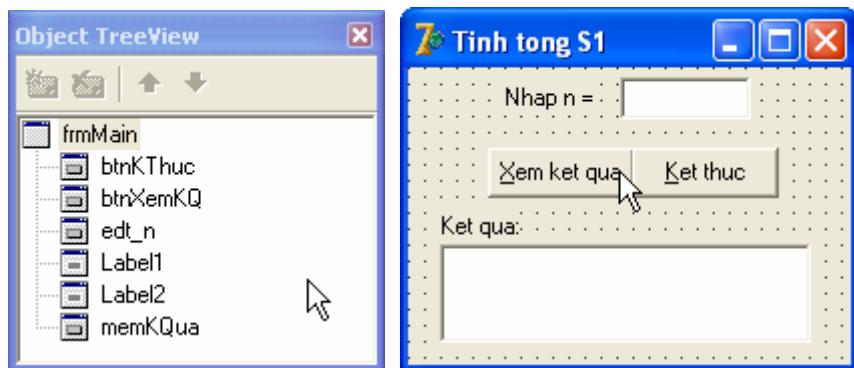
```
var
    suit : (Hearts, Clubs, Diamonds, Spades);
```

```
begin
    // Lệnh For lặp 3 lần
    for suit := Hearts to Diamonds do
        ShowMessage( ' Suit = ' + IntToStr(Ord(suit)) );
    end;
```

Ví dụ 9: Viết chương trình nhập vào một số nguyên dương n, sau đó tính tổng S1
như sau: $S1 = 1 + 2 + 3 + 4 + \dots + n$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 14: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```
unit untTongS1;

interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmMain = class(TForm)
        Label1: TLabel;
        edt_n: TEdit;
        btnXemKQ: TButton;
        btnKThuc: TButton;
    end;
```

```

Label2: TLabel;
memKQua: TMemo;
procedure btnXemKQClick(Sender: TObject);
procedure btnKThucClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.btnXemKQClick(Sender: TObject);
var n,i,s1:integer;
begin
  memKQua.Lines.Clear;
  n:=StrToInt(edt_n.Text);
  s1:=0; // khởi tạo biến tổng
  for i:=1 to n do s1:=s1+i;
  memKQua.Lines.Add( ' Tong S1 = ' + IntToStr(S1));
end;

procedure TfrmMain.btnKThucClick(Sender: TObject);
begin
  close;
end;

end.

```

- Lưu dự án vào thư mục **S:\ViduTongS1**

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 15: Chương trình tính tổng

*** Cú pháp đếm xuống:**

```
for <biến_đếm := trị_đầu> downto <trị_cuối> do công_việc;
```

Ý nghĩa: Tương tự như đếm lên. Lặp đếm xuống có trị_đầu>= trị_cuối, mỗi lần lặp biến đếm tự động giảm xuống 1.

Ví dụ 10: Đếm giảm

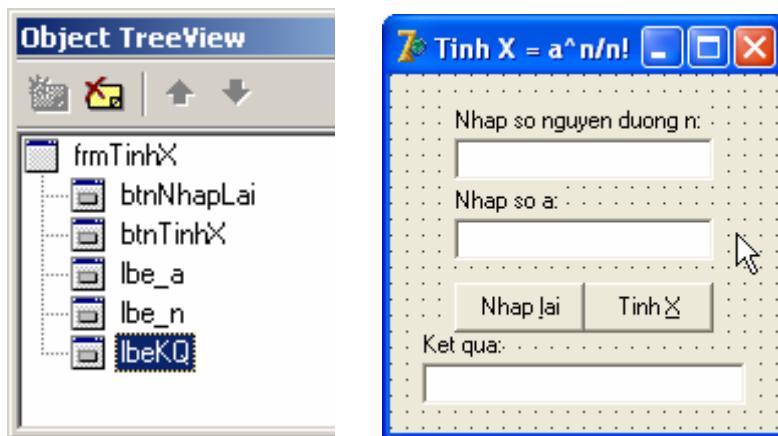
```
var c : char;
begin
  for c := 'E' downto 'A' do ShowMessage('c = '+c);
end;
```

Ví dụ 11: Nhập số nguyên dương n và một số thực a bất kỳ. Hãy tính giá trị của biểu thức X:

$$X = \frac{a^n}{n!} \Leftrightarrow X = \frac{a * a * ... * a}{1 * 2 * ... * n}$$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 16: Tên các đối tượng và giao diện chương trình

Đoạn code của chương trình:

```
unit untTinhX;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
```

Chương 6: Các lệnh có cấu trúc

```
TfrmTinhX = class(TForm)
    lbe_n: TLabledEdit;
    btnTinhX: TButton;
    lbeKQ: TLabledEdit;
    btnNhaphLai: TButton;
    lbe_a: TLabledEdit;
    procedure btnTinhXClick(Sender: TObject);
    procedure btnNhaphLaiClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

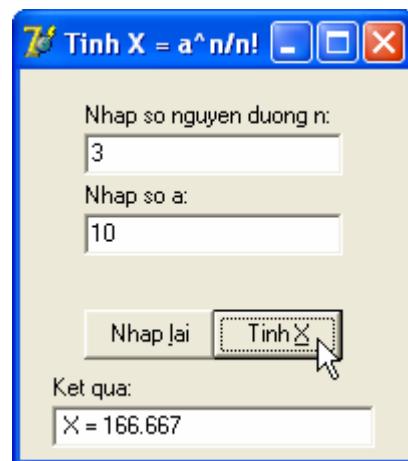
var
    frmTinhX: TfrmTinhX;

implementation

{$R *.dfm}
procedure TfrmTinhX.btnTinhXClick(Sender: TObject);
var mGiaiThua, mLuyThua, a, X : single;
    i, n:Integer;
begin
    if TryStrToInt(lbe_n.Text, n) and (n>=0) and
        TryStrToFloat(lbe_a.Text, a) then
    begin
        if n = 0 then X:=1      //a^0 =1 va 0! =1
        else
        begin
            mGiaiThua := 1;
            mLuyThua :=1;
            for i:=1 to n do
            begin
                mGiaiThua := mGiaiThua * i;
                mLuyThua := mLuyThua * a;
            end;
            X := mLuyThua/mGiaiThua;
        end;
        lbeKQ.Text := ' X = '+FormatFloat('#,#00.000',X);
    end
    else
        ShowMessage('Ban phai nhap n la so nguyen duong.');
end;
```

```
procedure TfrmTinhX.btnAddLaiClick(Sender: TObject);
begin
  lbe_n.Clear; lbe_a.Clear; lbeKQ.Clear;
  lbe_n.SetFocus;
end;
end.
```

- Lưu dự án vào thư mục: S:\ViduTinhX
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 17: Kết quả chương trình

III.2. Lệnh lặp có số lần không xác định trước

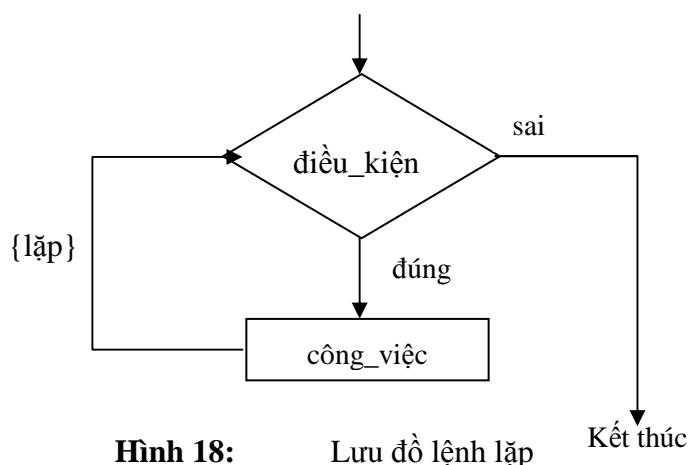
III.2.1. Lệnh lặp while .. do

Lệnh lặp while..do là vòng lặp kiểm tra điều kiện trước và thực hiện công việc sau. Nó cho phép chương trình thực hiện công việc nhiều lần và với số lần lặp không được xác định trước.

* Cú pháp tổng quát:

```
while <điều_kiện> do công_việc;
```

* Lưu đồ:



Hình 18: Lưu đồ lệnh lặp

*** Ý nghĩa:**

Đầu tiên là kiểm tra điều kiện, nếu điều kiện đúng thì thực hiện công việc, rồi quay trở về điều kiện để kiểm tra. Vòng lặp sẽ cứ tiếp tục cho đến khi điều kiện không còn đúng nữa thì sẽ kết thúc lệnh.

Chú ý:

- Điều kiện trong cấu trúc lặp **while ... do** là một biểu thức trả về giá trị kết quả có kiểu **Boolean** (kết quả chỉ có 2 giá trị là Đúng (True) hoặc Sai (False)).

Ví dụ 12: Chương trình sau hiển thị kết quả bình phương giá trị của biến a (ban đầu a = 1) và tăng dần biến a cho đến khi kết quả bình phương giá trị của biến a lớn hơn 100.

```

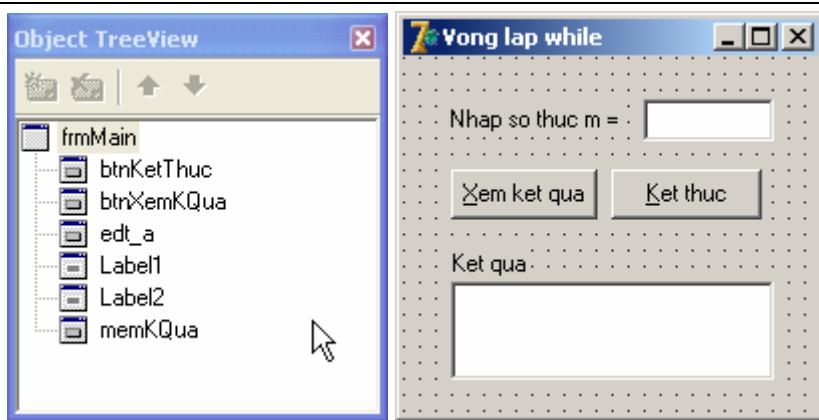
var num, bp: integer;
begin
  num := 1;
  bp := num * num; //bình phương
  while bp <= 100 do
    begin
      // Hiển thị kết quả bình phương giá trị của biến num
      ShowMessage( IntToStr(num) + ' squared = ' + IntToStr(bp) );
      inc(num); // Tăng giá trị của biến num lên 1
      bp := num * num;
    end;
  end.

```

Ví dụ 13: Viết chương trình tìm số nguyên dương k lớn nhất thỏa: $4^k < m$, với $m > 1$ là một số thực được nhập từ bàn phím.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 19: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit untVongLapWhile;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmMain = class(TForm)
    Label1: TLabel;
    edt_a: TEdit;
    btnKetThuc: TButton;
    btnXemKQua: TButton;
    Label2: TLabel;
    memKQua: TMemo;
    procedure btnKetThucClick(Sender: TObject);
    procedure btnXemKQuaClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;

implementation
{$R *.dfm}

procedure TfrmMain.btnKetThucClick(Sender: TObject);
var k:integer;
  s, m:extended;
begin
  memKQua.Lines.Clear;

```

```

m:=StrToFloat(edt_a.Text);
k:=0;
s:=1;
while s < m do
begin
    s:=s*4;
    k:=k+1;
end;
memKQua.Lines.Add('Gia tri k tim duoc la: '+ IntToStr(k-1));
memKQua.Lines.Add('Tong S= '+ FloatToStr(s/4));
end;

procedure TfrmMain.btnAddKQuaClick(Sender: TObject);
begin
    Close;
end;
end.

```

- Lưu dự án vào thư mục **S:\ViduWhile**

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 20: Tìm số dương k lớn nhất

Ví dụ 14: Viết chương trình nhập vào một số nguyên dương n bất kỳ. Hãy tính tổng các số hạng của số n này?

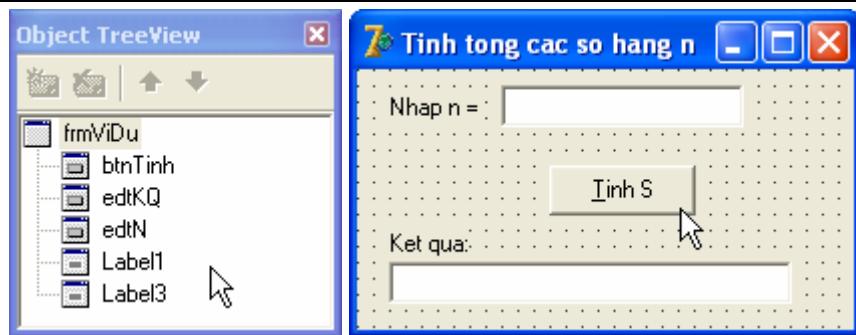
$$\text{Cụ thể với: } n = 356789107 \Rightarrow S = 3 + 5 + 6 + 7 + 8 + 9 + 1 + 0 + 7 \\ S = 46$$

$$n = 1257 \Rightarrow S = 1 + 2 + 5 + 7 \\ S = 15$$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**

- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 21: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit TachSoHang;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmViDu = class(TForm)
    edtN: TEdit;
    Label1: TLabel;
    btnTinh: TButton;
    edtKQ: TEdit;
    Label3: TLabel;
    procedure btnTinhClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmViDu: TfrmViDu;
implementation
{$R *.dfm}
procedure TfrmViDu.btnTinhClick(Sender: TObject);
var n, s: integer;
begin
  if TryStrToInt(edtN.Text, n) and (n>=1) then
  begin
    s:=0;
    while (n>0) do
    begin
      s:=s + n MOD 10;
      n:=n DIV 10;
    end;
  end;
end;

```

```

edtKQ.Text:=IntToStr(s);
end
else ShowMessage( 'Ban nhap du lieu KHONG thoac.' );
end;

end.

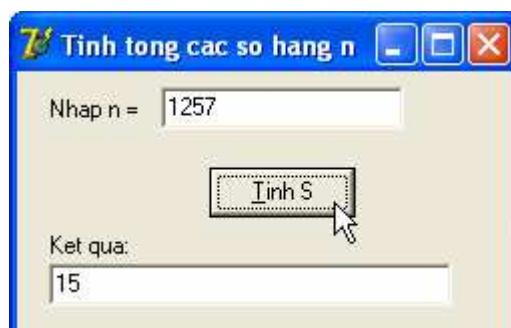
```

- Lưu dự án vào thư mục: S:\ViduTachSoHang

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 22: Với $n = 356789107$



Hình 23: Với $n = 1257$

III.2.2. Lệnh lặp Repeat ... until

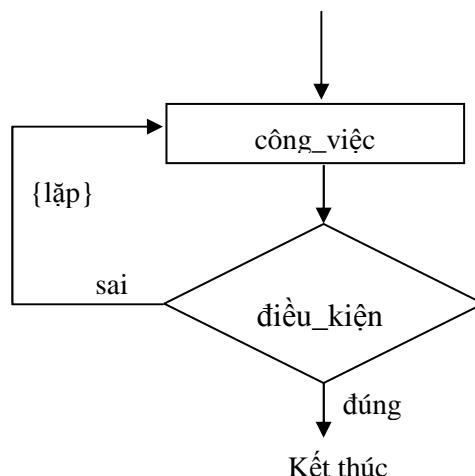
Lệnh lặp **repeat ... until** là vòng lặp thực hiện công việc trước, kiểm tra điều kiện sau. Nó cho phép chương trình thực hiện công việc nhiều lần và với số lần lặp không được xác định trước.

* **Cú pháp:**

```

repeat
    công_việc;
until điều_kiện;

```



Hình 24: Lưu đồ lệnh lặp repeat ... until

*** Ý nghĩa:**

Lệnh lặp **repeat ... until** sẽ tiếp tục thực hiện công việc cho đến khi điều kiện được thỏa, nghĩa là khi điều kiện có giá trị True thì sẽ thoát khỏi vòng lặp.

Chú ý:

- Trong lệnh lặp repeat..until, nếu công việc có từ hai lệnh trở lên thì không nhất thiết phải dùng cặp từ khóa **begin ... end**; để nhóm các lệnh đơn này lại.
- Trong lệnh lặp **while ... do**, do điều kiện được kiểm tra đầu tiên nên công việc có thể không được thực hiện lần nào. Còn trong lệnh lặp **repeat ... until** thì công việc được thực hiện trước và điều kiện sau từ khóa **until** được kiểm tra sau nên công việc được thực hiện ít nhất một lần.

Ví dụ 15: Chương trình sau hiển thị kết quả bình phương giá trị của biến num (ban đầu num = 1) và tăng dần biến num cho đến khi kết quả bình phương giá trị của biến num lớn hơn 100.

var

 num, bp : integer;

begin

 num := 1;

 bp := num * num;

repeat

 ShowMessage(IntToStr(num) + ' squared = ' + IntToStr(bp));

 Inc(num);

 bp := num * num;

until bp >100;

end;

Ví dụ 16: Viết chương trình tìm số nguyên dương k lớn nhất thỏa: $4^k < a$

Trong đó a>1 là một số thực được nhập từ bàn phím.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 25: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit VongLapRepeat;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmMain = class(TForm)
    Label1: TLabel;
    edt_a: TEdit;
    btnXemKQ: TButton;
    btnKetThuc: TButton;
    Label2: TLabel;
    memKQ: TMemo;
    procedure btnXemKQClick(Sender: TObject);
    procedure btnKetThucClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmMain: TfrmMain;
implementation
{$R *.dfm}

procedure TfrmMain.btnXemKQClick(Sender: TObject);
var k:integer;
  s, a:extended;
begin
  memKQ.Lines.Clear;
  a:=StrToFloat(edt_a.Text);

```

```

k:=0;  s:=1;
repeat
    s:=s*4;
    k:=k+1;
until s >= a;
memKQ.Lines.Add('Gia tri k tim duoc la: ' + IntToStr(k-1));
memKQ.Lines.Add('Tong S= ' + FloatToStr(s/4));
end;

Procedure TfrmMain.btnKetThucClick(Sender: TObject);
begin
    Close;
end;
end.
```

- Lưu dự án vào thư mục **S:\ViduRepeat**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 26: Chạy chương trình tìm k lớn nhất

CHƯƠNG 7: CHƯƠNG TRÌNH CON

I. Khái niệm

Khi lập trình, chúng ta thường có những đoạn chương trình hay phép tính lặp đi lặp lại nhiều lần cho một công việc/nhiệm vụ nào đó. Nếu mỗi lần lặp lại, ta phải viết những đoạn lệnh giống nhau này lặp đi lặp lại nhiều lần, thì chương trình của chúng ta trở nên dài dòng không cần thiết. Để khắc phục điều này, ta sử dụng chức năng chức năng sẵn có trong Delphi là chương trình con hay còn gọi là thường trình con (Routines).

Chương trình con thường được viết dưới dạng thủ tục (**procedure**) và hàm (**function**). Bên trong thân mỗi chương trình con đều chứa một dãy các câu lệnh để làm nhiệm vụ cho chương trình con đó. Như vậy, sau khi đã xây dựng xong chương trình con thì khi bạn muốn thi hành công việc mà chương trình con đó đảm nhận, bạn chỉ cần gọi đến nó ở bất kỳ nơi nào, và số lần gọi tùy ý từ bên trong chương trình thông qua tên của chương trình con đó.

Điểm khác biệt nhất giữa hàm và thủ tục là: Hàm luôn luôn trả về duy nhất **một** giá trị khi nó thi hành xong, còn thủ tục thì **không** trả về giá trị nào cả. Như vậy, hàm được xem như là một phần của một biểu thức còn thủ tục thì không.

Ví dụ 1: Trong Delphi, nếu ta sử dụng lệnh:

ShowMessage('Day la mot thu tuc.');

thì bạn thấy câu lệnh này chỉ làm nhiệm vụ in lên màn hình câu thông báo "Day la mot thu tuc." mà không có giá trị nào trả về cả.

Và câu lệnh $x := 10 + \text{Sin}(\text{Pi}/2)$; thì bạn biết $\text{Sin}(\text{Pi}/2)$ chính là một hàm vì nó trả về giá trị là 1 và như vậy, nó có thể tham gia vào biểu thức cộng ở trên.

Bạn cũng cần chú ý phân biệt chương trình con khác với các thủ tục sự kiện mà bạn đã được học ở các chương trước, nhiều nhất là ở chương 5. Chương trình con là chương trình do bạn tự khai báo nó và tự viết các lệnh trong thân của nó. Còn việc khai báo thủ tục sự kiện thì sẽ được Delphi tự động sinh ra tương ứng với một sự kiện cụ thể mà bạn đã chọn cho lớp/thành phần nào đó, còn các câu lệnh trong thân của nó thì bạn mới cần viết mà thôi.

Vị trí để đặt các chương trình con là ở trong phần cài đặt của unit, nghĩa là ngay sau từ khóa **implementation** của unit.

Khi đề cập đến chương trình con, bạn cần phải biết về các loại biến được sử dụng bên trong **chương trình** và bên trong **chương trình con** là khác nhau:

· **Biến toàn cục** (Global variable): Còn được gọi là biến chung, là biến được khai báo ở đầu chương trình của unit form (form file), nó được sử dụng bên trong tập tin dự án (Project file) và cả bên trong chương trình con của unit file đó. Biến toàn cục sẽ tồn tại trong suốt quá trình thực hiện chương trình.

· **Biến cục bộ** (Local variable): Còn được gọi là biến riêng, là biến được khai báo ở đầu chương trình con, và nó chỉ được sử dụng bên trong thân chương trình con hoặc bên trong thân chương trình con khác nằm bên trong nó (các chương trình con lồng nhau). Biến cục bộ chỉ tồn tại khi chương trình con đang hoạt động, nghĩa là biến cục bộ sẽ được cấp phát bộ nhớ khi chương trình con được gọi để thi hành, và nó sẽ được giải phóng ngay sau khi chương trình con kết thúc.

· **Tham số hình thức** (Formal parameter) là biến được khai báo ngay sau tên chương trình con, nó dùng để nhận giá trị của tham số thực truyền đến. Tham số hình thức cũng là một biến cục bộ, ta có thể xem nó như là các đối số của hàm Toán học. Nó được sử dụng **trong quá trình xây dựng** chương trình con. Trong Delphi có nhiều loại tham số như: tham số trị (Value parameter), tham số biến (Variable parameter), tham số hằng (Constant parameter), tham số xuất (Out parameter). Ý nghĩa và cách vận dụng các loại tham số này như thế nào sẽ được trình bày kỹ ở phần “Truyền tham số” ở mục sau.

· **Tham số thực** (Actual parameter) là tham số mà nó có thể là biến toàn cục, biểu thức hoặc giá trị số và cũng có thể là biến cục bộ khi sử dụng chương trình con lồng nhau. Nó được sử dụng khi bạn muốn truyền giá trị của tham số thực cho tham số hình thức tương ứng của chương trình con khi ra lệnh lời gọi chương trình con.

Ví dụ 2: Trong Delphi đã xây dựng sẵn hàm $\sin(x)$. Còn khi sử dụng hàm, bạn phải cụ thể là tính sin của x là bao nhiêu, chẳng hạn $x = \pi/2$. Trong Delphi ta có hàm $\sin(\pi/2)$. Như vậy, x là tham số hình thức, còn $\pi/2$ là tham số thực nghĩa là giá trị thực sự mà bạn cần tính.

* Lời gọi chương trình con (thủ tục và hàm):

Để chương trình con được thi hành, ta phải có lời gọi đến chương trình con, lời gọi chương trình con thông qua tên chương trình con và danh sách các tham số thực tương ứng (nếu có). Các qui tắc của lời gọi chương trình con:

- Trong thân chương trình chính hoặc thân chương trình con, ta chỉ có thể gọi tới các chương trình con trực thuộc nó.
- Trong chương trình con, ta có thể gọi các chương trình con ngang cấp đã được thiết lập trước đó.

II. Hàm

```
function Tên_hàm[(danh_sách_tham_số_hình_thức)]: kiểu_trả_về; [các_chi_dẫn;]
[Khai_báo_cục_bộ]
begin
    // Các câu lệnh trong thân của function.
end;
```

Trong đó:

- **Tên_hàm** được đặt tên theo đúng quy cách của danh hiệu tự đặt trong Delphi.
- **Kiểu_trả_về** là một kiểu vô hướng, biểu diễn kiểu của giá trị trả về của hàm.
- **Các_chi_dẫn** (directives) là các chỉ thị về quy ước gọi hàm như **register**, **pascal**, **cdecl**, **stdcall**, và **safecall**; hoặc chỉ thị khai báo trước **forward**; hoặc khai báo ngoài **external**... Do sự giới hạn của của giáo trình, nên phần này không được trình bày ở đây. Nếu các bạn nào muốn tìm hiểu thêm, các bạn hãy truy cập vào trang web của Delphi là <http://delphi.about.com/> hoặc xem phần help của bộ Delphi.

- Một hàm có thể có 1 hay nhiều tham số hình thức, khi có nhiều tham số hình thức cùng một kiểu giá trị thì ta có thể viết chúng cách nhau bằng dấu phẩy (,). Trường hợp các tham số hình thức khác kiểu thì ta viết chúng cách nhau bằng dấu chấm phẩy (;).

Ví dụ 3: **function** Tinh_tong (x, y: integer; z:extended): extended;

- Để tìm được giá trị kết quả của hàm cần trả về, thường cần phải thông qua nhiều bước trung gian để đạt đến kết quả cuối cùng. Do vậy, chúng ta thường sử dụng 1 biến cục bộ để chứa kết quả trung gian này và được gọi là **biến trung gian**. Kiểu của biến trung gian này phải **tương thích** hoặc chính xác với kiểu trả về của hàm. Sau khi tính được kết quả này xong, bạn cần phải gán kết quả tính được trong biến trung gian này cho **tên hàm** hoặc bạn có thể sử dụng biến được định nghĩa sẵn trong Delphi là **result** để nhận kết quả trả về của hàm. Bạn chú ý rằng, kiểu của biến result luôn luôn tương thích với kết quả trả về của hàm.

Lưu ý: Vị trí để viết các chương trình con (hàm hoặc thủ tục) là sau từ khóa **implementation** và {\$R *.dfm}

Ví dụ 4: Hàm tính x^n được tính bằng cách sử dụng biến định sẵn **result**

```
function So_mu(x: single; n: integer): single;
var
    i: integer;
begin
    result := 1.0;
    i := n;
    while i > 0 do
        begin
            if odd(i) then
                result := result * x;
            i := i div 2;
            x := sqr(x);
        end;
    end;
```

- Khi khai báo kiểu dữ liệu cho các tham số hình thức là kiểu **string** mà có độ dài xác định trước hay kiểu **array** trong thủ tục và hàm, ta phải sử dụng cách khai báo gián tiếp (through qua từ khóa **type**) như sau:

Ví dụ 5: **function** So_sanh(hten1, hten2: string[40]): boolean; //sai cú pháp.
procedure In_mang(a: array[1..10] of integer); //sai cú pháp.

Ta có thể sửa lại như trong ví dụ sau:

Ví dụ 6: Sử dụng cách khai báo gián tiếp để sửa type

```
Str40 = string[40];
```

```
Arr10 = array[1..10] of integer;
```

```
function So_sanh(hten1, hten2: Str40): boolean; //đúng cú pháp.
```

```
procedure In_mang(a: Arr10); //đúng cú pháp.
```

Hoặc sử dụng mảng động để khai báo như sau:

```
procedure In_mang(a: array of integer); //đúng cú pháp.
```

- **Khái niệm thêm cho ví dụ này:**

Câu lệnh bắt lỗi lúc chương trình đang chạy (run-time):

```
try
```

các câu lệnh;

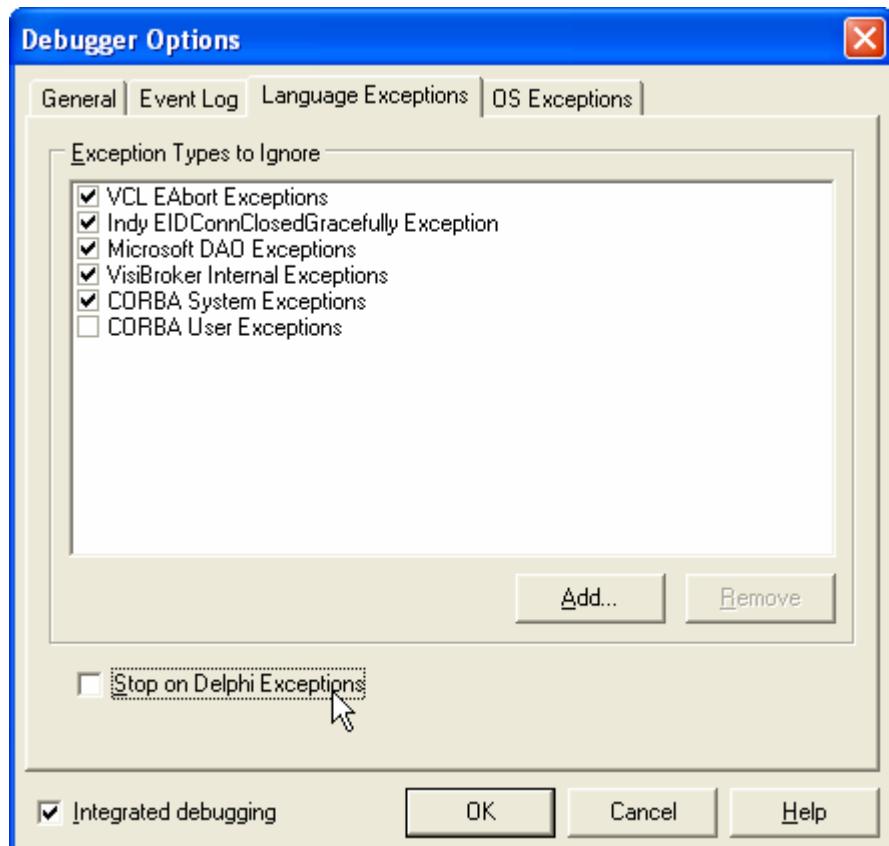
```
except // có phát sinh lỗi
```

// Delphi thông báo lỗi, hoặc bạn có thể đưa ra thông báo lỗi của riêng mình.

```
on E: Exception do ErrorDialog(E.Message, E.HelpContext);
```

```
end;
```

Chú ý: Bạn chọn chức năng **Tools/Debugger Options/Language Exceptions** và bỏ chọn (clear) hộp kiểm **Stop on Delphi Exceptions** để Delphi bắt lỗi.



Hình 74: Hộp thoại Debugger Options

Ví dụ 7: Viết chương trình tính giải thừa bằng chương trình con:

- Tạo một dự án mới **File/New/Application**

- Thiết lập các thuộc tính của form như sau:

Thuộc tính	Giá trị
Name	frmGiaiThua
Caption	Vi du ve ham – Function
Position	poScreenCenter

Bảng 1: Thiết lập thuộc tính cho Form

- Đặt các đối tượng lên form và thiết lập các thuộc tính như sau:

Đối tượng	Thuộc tính	Giá trị
Label	Name	lblKetqua
	Caption	Kết quả n!
Edit	Name	edt_Kq
	Text	Rỗng
Button 1	Name	btnNhaph
	Caption	&Nhập n
Button 2	Name	btnKetThuc
	Caption	&Kết thúc

Bảng 2: Thiết lập thuộc tính cho các đối tượng

- Ở ví dụ này, giáo trình muốn trình bày với bạn cách sử dụng **biến chung**: Để khai báo một biến chung có tên biến là **num_str** và kiểu là **string** thì từ cửa sổ thiết kế form, bạn nhấn phím chức năng F12 để chuyển sang cửa sổ mã lệnh. Bạn chuyển con nháy lên phía đầu **unit** và ngay sau phần khai báo biến biến chung là:

```
var
  frmGiaiThua: TfrmGiaiThua;
```

bạn thêm vào 1 biến chung **num_str** như sau: num_str: string;

Hoặc bạn có thể sử dụng cách đã được trình bày ở phần hộp kiểm (TCheckbox) để thêm biến num_str.

Sau khi hoàn thành, phần khai báo của bạn sẽ như sau:

```
var
  frmGiaiThua: TfrmGiaiThua;
  num_str: string;      //biến chung

implementation

{$R *.dfm}
```

- Ngay sau từ khóa **implementation** và {\$R *.dfm} là **phần để bạn nhập các chương trình con**, bạn nhập vào hàm Giai_thua như sau:

```
function Giai_thua(n:integer):extended; //n là tham số hình thức
var i: integer; kqtg: extended; // biến riêng
begin
    kqtg := 1; //khởi tạo biến trung gian
    if n <= 0 then Giai_thua := 1
    else
        begin
            for i := 1 to n do kqtg := kqtg * i ; //kết quả trung gian
            Giai_thua := kqtg; //gán giá trị của kqtg cho tên hàm
        end ;
    end;
```

- Tiếp theo, bạn lần lượt nhập vào các câu lệnh cho các thủ tục sự kiện.

- Chương trình hoàn chỉnh sẽ có đoạn mã lệnh như sau:

```
unit untGiaithua;

interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmGiaiThua = class(TForm)
        edt_Kq: TEdit;
        lblKetqua: TLabel;
        btnKetThuc: TButton;
        btnNhaph: TButton;
        procedure btnKetThucClick(Sender: TObject);
        procedure btnNhaphClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmGiaiThua: TfrmGiaiThua;
    num_str:string; //biến chung

implementation
{$R *.dfm}
```

```
//-----Phần để bắt đầu viết các chương trình con (nếu có)-----

function Giai_thua(n: integer): extended;
var i:integer; kqtg:extended; //bien rieng
begin
    kqtg := 1;
    if n <= 0 then Giai_thua := 1
    else
    begin
        for i := 1 to n do kqtg := kqtg * i ;
        Giai_thua := kqtg;
    end ;
end ;

//-----các thủ tục sự kiện -----

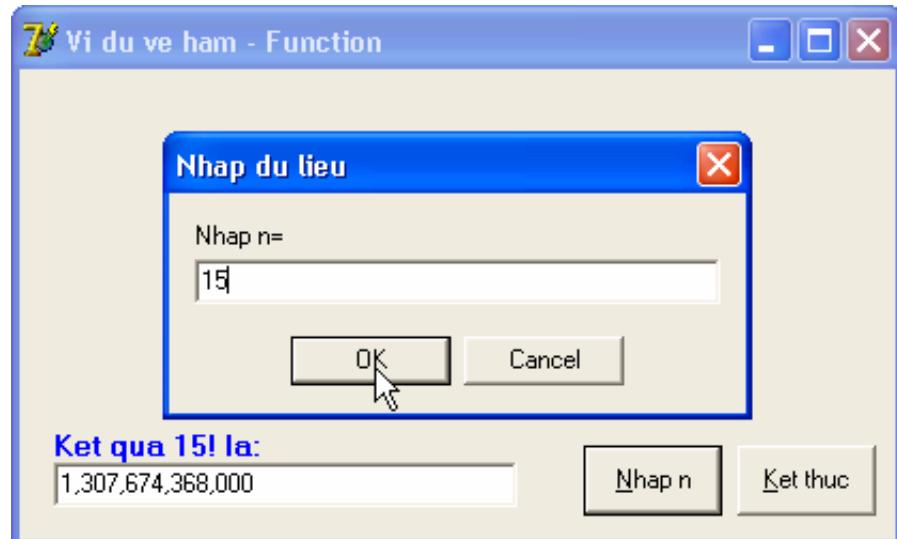
procedure TfrmGiaiThua.btnExitClick(Sender: TObject);
begin
    Close;
end;

procedure TfrmGiaiThua.btnNhapClick(Sender: TObject);
var num:integer;
begin
    edt_Kq.Clear;
    try
        num_str:= InputBox('Nhập dữ liệu', 'Nhập n= ','15');
        num := StrToInt( num_str );
        if num>=0 then
            begin
                lblKetqua.Caption:= 'Kết quả ' + num_str + '! là:';
                edt_Kq.Text:=FormatFloat('#,##0',Giai_thua(num));
            end
        else ShowMessage('Không tính gaii thua của số am.');
    except
        ShowMessage('Giá trị "' + num_str + '" không hợp lệ để tính
                    gaii thua.');
    end; // end try
end;

end.
```

* **Chú ý:** Bạn có thể thay câu lệnh **try ... except ... end;** bằng câu lệnh TryStrToInt như các ví dụ đã trình bày ở phần trên.

- Lưu dự án vào thư mục S:\ViduGiaiThua
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 1: Kết quả tính n!

Nếu bạn nhập làm giá trị là $n=20$, thì chương trình sẽ bắt lỗi và thông báo cho bạn như sau:



Hình 2: Lỗi nhập sai kiểu dữ liệu

Hoặc bạn nhập vào giá trị là $n=-7$, thì chương trình cũng sẽ thông báo cho bạn biết:



Hình 3: Lỗi nhập sai giá trị âm

Từ ví dụ trên, bạn thấy biến num_string đã được khai báo làm biến chung, vậy theo bạn có thể khai báo nó làm biến riêng trong thủ tục sự kiện **procedure TfrmMain.btnNhapClick(Sender: TObject);** được không? Câu trả lời là được. Tùy theo yêu cầu/ý nghĩa của chương trình mà bạn quyết định sử dụng cách khai báo là chung hay riêng cho hợp lý.

III. Thủ tục

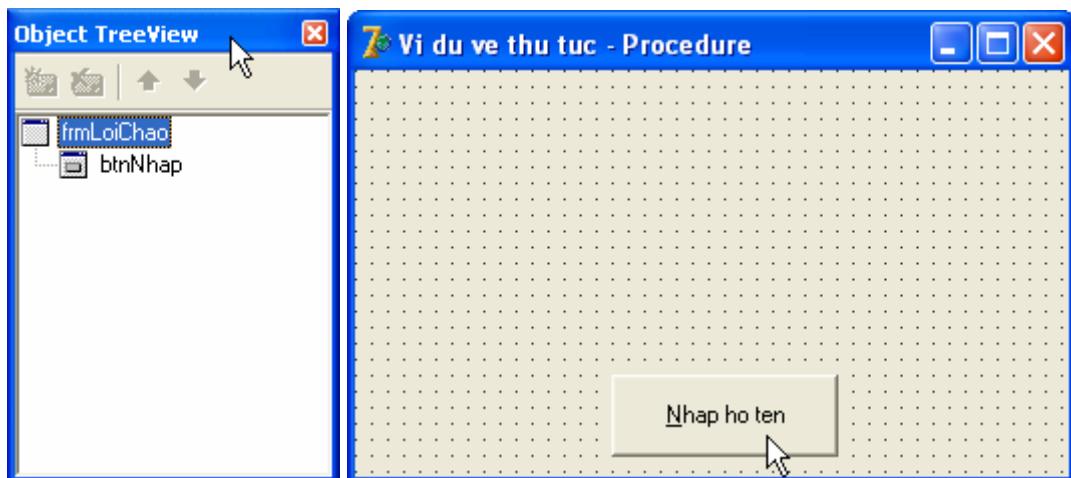
```
Procedure Tên_thủ_tục[(Danh_sách_tham_số_hình_thúc)]; [Các_chỉ_dẫn;]
[Khai_báo_cục_bộ]
begin
    // Các câu lệnh trong thân của procedure.
end;
```

Các khái niệm như tên thủ tục, tham số hình thức, các chỉ dẫn thì giống như phần đã trình bày ở **function**.

Ví dụ 8: Viết chương trình hiển thị câu chào tới anh/chị bất kỳ bằng thủ tục, với họ tên được nhập từ bàn phím.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 4: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```
unit untLoiChao;           //Lời chào

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmLoiChao = class(TForm)
    btnNhap: TButton;
    procedure btnNhapClick(Sender: TObject);
```

```

private
  { Private declarations }

public
  { Public declarations }
end;

var
  frmLoiChao: TfrmLoiChao;

implementation
{$R *.dfm}

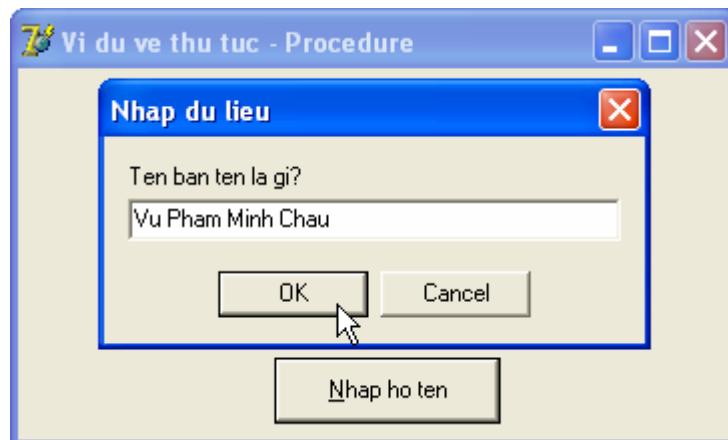
//----- Phần thủ tục -----
procedure Loi_chao(hten:string);
begin
  ShowMessage('Hi! chao ban '+hten+'. Chuc ban mot ngay vui ve!');
end;

//----- Phần thủ tục sự kiện -----
procedure TfrmLoiChao.btnNhapClick(Sender: TObject);
var hoten:string;           //bien rieng
begin
  hoten:=InputBox('Nhập địa điểm','Tên bạn là gì? ','');
  Loi_chao(hoten);          //lời gọi thủ tục
  hoten:=InputBox('Hợp nhập','Hay nhập tên mà bạn yêu thích:','');
  Loi_chao(hoten);          //lời gọi thủ tục
end;

end.

```

- Lưu dự án vào thư mục **S:\ViduLoiChao**
 - Biên dịch và chạy chương trình, ta có kết quả như sau:
- Ban đầu, chương trình sẽ đề nghị bạn nhập họ tên của bạn vào, như “Vũ Phạm Minh Châu” chẳng hạn:



Hình 5: Chương trình ví dụ về thủ tục- nhập tên lần một

tiếp đến, chương trình sẽ gọi thủ tục Loi_chao(hoten) để in ra câu chào như sau:



Hình 6: Lời chào cho lần gọi thủ tục thứ nhất

Khi bạn Click vào nút lệnh OK, thì chương trình sẽ cho bạn nhập thêm họ tên mới, chẳng hạn như “Thúy An”, tiếp đến chương trình sẽ gọi lại thủ tục Loi_chao(hoten) để in ra lời chào tiếp theo:



Hình 7: Nhập tên lần hai



Hình 8: Lời chào cho lần gọi thủ tục thứ hai

Như vậy, thủ tục **Loi_chao(hoten)** đã được gọi 2 lần trong chương trình và như thế bạn thấy chương trình sẽ sáng sủa và rõ ràng hơn.

IV. Truyền tham số

Trong Delphi có nhiều cách truyền tham số như: tham số trị (value), tham số biến (variable), tham số hằng (constant), tham số xuất (out). Ứng với mỗi loại truyền tham số, mà chúng có ý nghĩa khác nhau khi truyền từ tham số thực đến tham số hình thức.

IV.1. Định kiểu và không định kiểu cho tham số hình thức

Ta có 2 khái niệm về định kiểu dữ liệu cho các tham số hình thức đó là: tham số định kiểu (typed parameter) và tham số không định kiểu (untyped parameter). Tham số định kiểu là tham số khi ta khai báo phải chỉ rõ kiểu dữ liệu cho nó, còn tham số không định kiểu thì **không** cần phải chỉ ra trước kiểu của nó và như vậy tham số không định kiểu có thể **tương thích** với bất kỳ kiểu dữ liệu của tham số thực khi truyền đến. Với cách khai báo tham số không định kiểu, nếu ta sử dụng giá trị của các tham số này, thì trước hết ta phải chuyển đổi kiểu (typecast) cho chúng trước.

Ví dụ 9: Viết thủ tục Cong2So với 2 tham số a, b là không định kiểu:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 9: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```
unit untKhongDinhKieu;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmVidu = class(TForm)
    btnNhap: TButton;
    procedure btnNhapClick(Sender: TObject);
  private
```

Chương 7: Chương trình con

```
{ Private declarations }

public
{ Public declarations }
end;

var
frmVidu: TfrmVidu;

implementation
{$R *.dfm}

procedure Cong2So(const a,b; chon:char);
var kq:Extended;
begin
case chon of
'1': begin
kq:= Integer(a) + Extended(b); //chuyen kieu - casting
ShowMessage('Truong hop '+ chon+': Tong 2 so = '+
FloatToStr(kq));
end;
'2': begin
kq:= Extended(a) + Integer(b); //chuyen kieu - casting
ShowMessage('Truong hop '+ chon+': Tong 2 so = '+
FloatToStr(kq));
end;
end;
end;

procedure TfrmVidu.btnNhapClick(Sender: TObject);
var x1:integer; x2:extended;
begin
x1:= -5; x2:=130.5;
Cong2So(x1,x2,'1'); { Truyền x1 kiểu integer cho tham số hình thức a,
truyền x2 kiểu extended cho tham số hình thức b. }

x1:=10;x2:=15.5;
Cong2So(x2,x1,'2'); { Truyền x2 kiểu extended cho tham số hình thức a,
truyền x1 kiểu integer cho tham số hình thức b. }
end;

end.
```

- Lưu dự án vào thư mục **S:\ViduKhongDinhKieu**

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 10: Kết quả chương trình.

Khi chạy chương trình, ta có kết quả bằng 125.5 và 25.5. Như vậy, từ 2 trường hợp trên ta nhận thấy: **tham số a cũng như tham số b đều có thể nhận cả 2 kiểu dữ liệu Integer và extended.**

IV.2. Truyền bằng tham trị (Value parameter)

- **Cú pháp:** Không có từ khóa nào đứng trước tham số hình thức.
- **Ý nghĩa:** Không làm thay đổi giá trị của tham số thực khi giá trị của tham trị thay đổi.
- **Tham số thực:** có thể là một biến, một biểu thức, một hằng, hoặc một giá trị số bất kỳ mà có kiểu tương thích với kiểu của tham trị.
- **Lưu ý:** Bắt buộc sử dụng tham số hình thức định kiểu (typed)

Ví dụ 10: Chương trình tính lập phương bằng **tham trị**

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 11: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình có sử dụng **tham số trị** như sau:

```

unit untThamTri;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmThamTri = class(TForm)
        btnNhap: TButton;
        Label1: TLabel;
        procedure btnNhapClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmThamTri: TfrmThamTri;

implementation
{$R *.dfm}

procedure LapPhuong(a: integer);           //a la tham tri.
begin
    a := a*a*a;      // tinh lap phuong
end;

procedure TfrmThamTri.btnNhapClick(Sender: TObject);
var x:integer;
begin
    x := StrToInt(InputBox('Nhập dữ liệu', 'Nhập số x = ','2'));
    LapPhuong(x);  {tham số thực x được truyền cho tham số a}
    ShowMessage('Lập phương của số vừa nhập là ' +IntToStr(x));
end;
end.

```

- Lưu dự án vào thư mục **S:\ViduThamTri**

- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 12: Kết quả tính lập phương của 2

- Khi chạy chương trình, chẳng hạn bạn nhập $x = 2$, bạn có đoán được kết quả của chương trình sau khi tính lập phương của 2 là bao nhiêu không?... $x = 2$ hay $x = 8$? Kết quả thật bất ngờ: lập phương của 2 vẫn bằng 2. Suy ra chương trình đã cho kết quả **sai**. Vì thủ tục LapPhuong ở trên đã sử dụng cách truyền bằng tham trị, do vậy, mặc dù trong thân thủ tục giá trị của tham số a được gán giá trị 8, nhưng x vẫn giữ nguyên giá trị của nó là 2 mà không thay đổi theo giá trị của a .

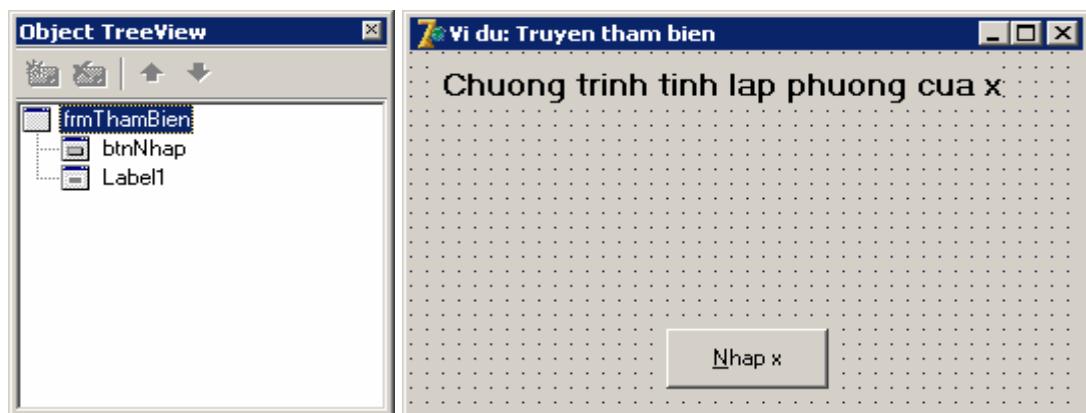
IV.3. Truyền bằng tham biến (Variable parameter)

- **Cú pháp:** Có từ khóa **Var** đứng trước tham số hình thức.
- **Ý nghĩa:** Làm thay đổi giá trị của tham số thực khi giá trị của tham số hình thức thay đổi.
- **Tham số thực:** chỉ có thể là **biến** và có kiểu tương thích với kiểu của tham số hình thức.
- **Lưu ý:** có thể sử dụng cách tham số định kiểu hoặc không định kiểu.

Ví dụ 11: Chương trình tính lập phương bằng **tham biến**:

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 13: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình có sử dụng tham biến **var** như sau:

```

unit untThamBien;

interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmThamBien = class(TForm)
        btnNhap: TButton;
        Label1: TLabel;
        procedure btnNhapClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmThamBien: TfrmThamBien;

implementation
{$R *.dfm}

procedure LapPhuong(var a: integer); {a là tham biến}
begin
    a := a*a*a; // tính lập phương
end;

procedure TfrmThamBien.btnNhapClick(Sender: TObject);
var x:integer; // biến riêng
begin
    x := StrToInt(InputBox('Nhập dữ liệu', 'Nhập số x = ','2'));
    LapPhuong(x); {tham số thực x được truyền cho tham biến a}
    ShowMessage('Lập phương của số vừa nhập là ' +IntToStr(x));
end;

end.

```

- Lưu dự án vào thư mục **S:\ViduThamBien**

- Biên dịch và chạy chương trình, ta có kết quả như sau:

**Hình 14:** Kết quả chương trình tính lập phương x

- Khi chạy chương trình, chẳng hạn bạn nhập $x = 2$, thì kết quả chương trình trả về là $x = 8$. Suy ra chương trình cho kết quả **đúng**. Do thủ tục `LapPhuong` ở trong chương trình phía trên đã sử dụng cách truyền bằng tham biến, do vậy bên trong thân thủ tục `LapPhuong`, của tham số `a` được gán giá trị 8, thì giá trị của `x` cũng sẽ thay đổi theo `a` và như vậy $x = 8$.

IV.4. Truyền bằng tham số hằng (Constant parameter)

- **Cú pháp:** Có từ khóa `Const` đứng trước tham số hình thức.
- **Ý nghĩa:** Giống như tính chất của tham trị, nhưng bạn **không thể** thay đổi giá trị của của tham số hằng trong thân chương trình con được.
- **Tham số thực:** có thể là một biến, một biểu thức, một hằng, hoặc một giá trị số bất kỳ mà có kiểu tương thích với kiểu của tham số hình thức.
- **Lưu ý:** có thể sử dụng cách tham số định kiểu hoặc không định kiểu.

Ví dụ 12: Trường hợp sai khi thay đổi giá trị của tham số hằng `a` trong thủ tục `LapPhuong`:

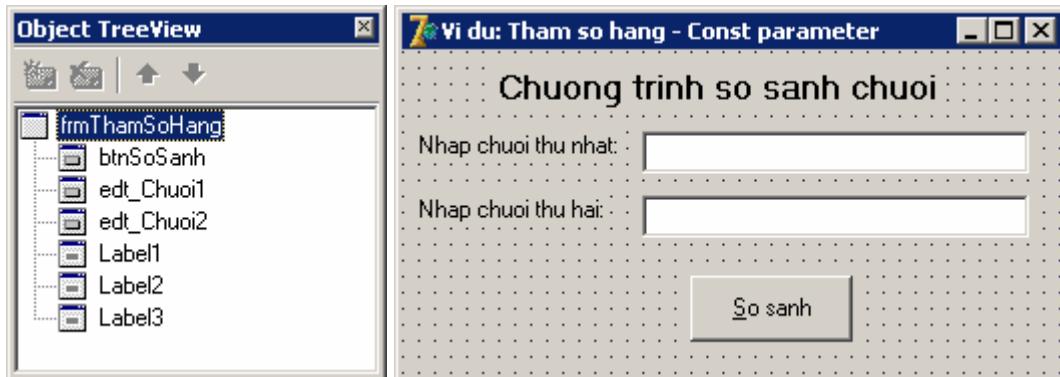
```
procedure LapPhuong(Const a:integer);      {a là một tham số hằng}
begin
  a := a*a*a;    // lỗi cú pháp
end;
```

Ví dụ 13: Xây dựng hàm so sánh chuỗi, nếu 2 chuỗi bằng nhau thì hàm trả về giá trị 0, chuỗi thứ nhất lớn hơn chuỗi thứ 2 thì hàm trả về 1, còn lại hàm trả về giá trị -1

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**

- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 15: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit untThamSoHang;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmThamSoHang = class(TForm)
    btnSoSanh: TButton;
    Label1: TLabel;
    edt_Chuoil: TEdit;
    edt_Chuoiz: TEdit;
    Label2: TLabel;
    Label3: TLabel;
    procedure btnSoSanhClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmThamSoHang: TfrmThamSoHang;

implementation
{$R *.dfm}

function SoSanhChuoi(const chuoil, chuoiz: string): shortint;
  //chuoil va chuoiz la 2 tham so hang.
begin
  if chuoil = chuoiz then result := 0 //bang nhau

```

```

else
  if chuoil > chuoit then result := 1 //chuoil lon hon chuoit
  else result := -1; //chuoil nho hon chuoit
end;

procedure TfrmThamSoHang.btnAddSoSanhClick(Sender: TObject);
var kq: shortint;
begin
  kq := SoSanhChuoi(edt_Chuoil.Text, edt_Chuoit.Text);
  case kq of
    -1: ShowMessage(edt_Chuoil.Text + ' < ' + edt_Chuoit.Text);
    0: ShowMessage(edt_Chuoil.Text + ' = ' + edt_Chuoit.Text);
    1: ShowMessage(edt_Chuoil.Text + ' > ' + edt_Chuoit.Text);
  end;
end;

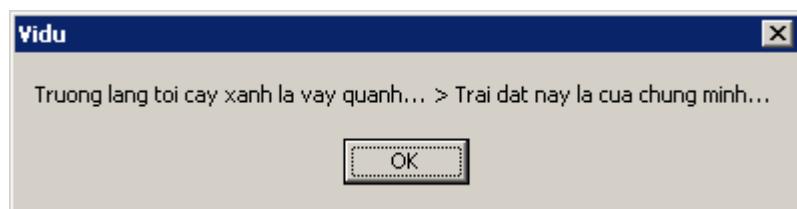
end.

```

- Lưu dự án vào thư mục **S:\Vidu\ThamSoHang**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 16: Chạy chương trình



Hình 17: Kết quả so sánh.

Chú ý: chuoil và chuoit không được thay đổi giá trị của nó trong thân chương trình con So_sanh_chuoi vì chúng là những tham số hằng.

IV.5. Truyền bằng tham số xuất (Out parameter)

- **Cú pháp:** Có từ khóa **Out** đứng trước tham số hình thức.

- **Ý nghĩa:** Làm thay đổi giá trị của tham số thực khi giá trị của tham số xuất thay đổi. Tham số **xuất** này chỉ dùng trong trường hợp bạn không cần quan tâm đến giá trị **đầu vào** của tham số thực trước khi truyền đến nó mà chỉ quan tâm đến kết quả **đầu ra**.

- **Tham số thực:** chỉ có thể là **biến** và có kiểu tương thích với kiểu của tham số hình thức.

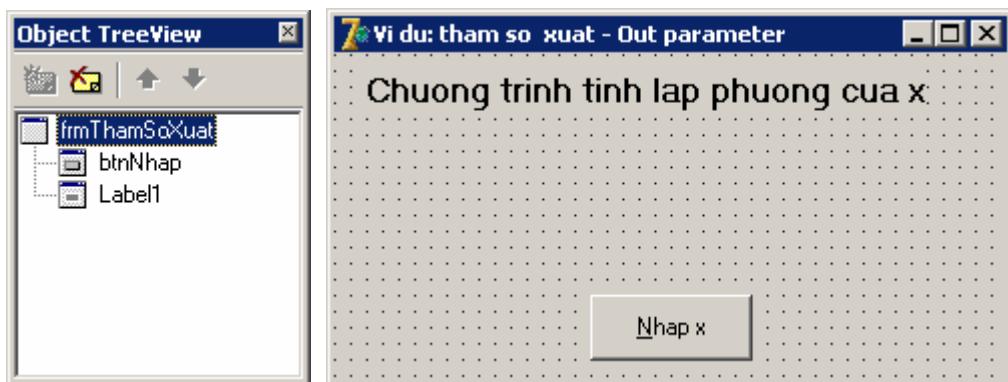
- **Lưu ý:** có thể sử dụng cách tham số định kiểu hoặc không định kiểu.

Ví dụ 14: Chương trình tính lập phương bằng **tham số xuất**:

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**

- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 18: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình có sử dụng tham số **out** như sau:

```
unit untThamSoXuat;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmThamSoXuat = class(TForm)
    btnNhap: TButton;
    Label1: TLabel;
    procedure btnNhapClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;
```

```

var
    frmThamSoXuat: TfrmThamSoXuat;

implementation
{$R *.dfm}

Procedure LapPhuong(a: integer; out kq:integer);
            // a la tham tri, kq la tham so xuat
begin
    kq := a*a*a;      // tinh lap phuong
end;

procedure TfrmThamSoXuat.btnNhapClick(Sender: TObject);
var x, kqlp:integer;
begin
    x := StrToInt(InputBox('Nhập dữ liệu', 'Nhập số x = ','2'));
    LapPhuong(x,kqlp); {tham số thực x được truyền cho tham số a
                        và tham số thực kqlp được truyền cho tham số xuất kq}
    ShowMessage('Lập phương của '+IntToStr(x)+' là: ' +
                IntToStr(kqlp));
end;

end.

```

- Lưu dự án vào thư mục **S:\Vidu\ThamSoXuat**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 19: Kết quả chương trình

- Khi chạy chương trình, thì biến kqlp sẽ được truyền đến cho tham số xuất kq. Vì khi kq thay đổi giá trị thì kqlp sẽ thay đổi theo. Do vậy, nếu bạn nhập x = 2, thì khi thực hiện thủ tục LapPhuong, tham số xuất kq = 8 và suy ra biến kqlp cũng bằng 8.

Ví dụ 15: Viết chương trình nhập vào số nguyên dương n.

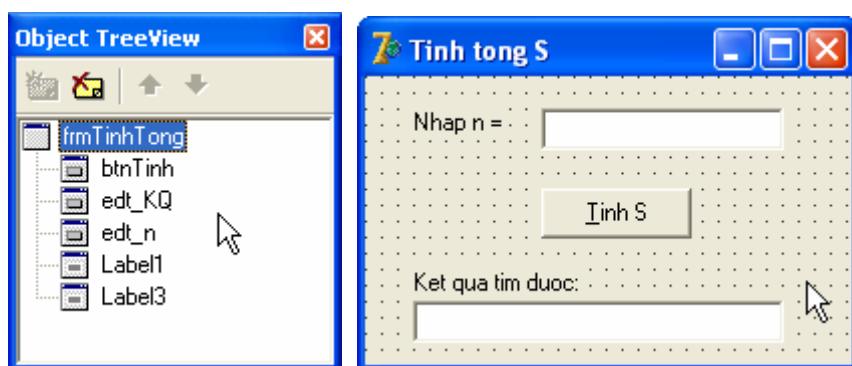
$$\text{Tính tổng } S = [f(1)]^2/g(1) + [f(2)]^2/g(2) + \dots + [f(n)]^2/g(n)$$

Trong đó: $f(k) = 2^0 + 2^1 + 2^2 + \dots + 2^k$

$$g(k) = 0! + 1! + 2! + \dots + k!$$

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 20: Tên các đối tượng và giao diện chương trình

Đoạn code của chương trình như sau:

```

unit untTongS;           //tên unit file

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmTinhTong = class(TForm)
    edt_n: TEdit;
    Label1: TLabel;
    btnTinh: TButton;
    edt_KQ: TEdit;
    Label3: TLabel;
    procedure btnTinhClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmTinhTong: TfrmTinhTong;

implementation

```

```

{$R *.dfm}

function GT(n: integer): extended; //hàm giải thừa của n
var i:integer;
begin
    result:=1;
    for i:=1 to n do result:=result*i;
end;

function LT(a:extended; n: integer): extended; //hàm a lũy thừa n
var i:integer;
begin
    result:=1;
    for i:=1 to n do result:=result*a;
end;

function g(k: integer): extended;
var i:integer;
begin
    result:=0;
    for i:=0 to k do result:=result+ GT(i);
end;

function f(k: integer): extended;
var i:integer;
begin
    result:=0;
    for i:=0 to k do result:=result+ LT(2,i);
end;

procedure TfrmTinhTong.btnExitClick(Sender: TObject);
var i,n: integer;
    S: extended;
begin
    S := 0;
    n := StrToInt(edt_n.Text);
    for i := 1 to n do
        S := S + LT(f(i),i)/g(i);
    edt_Kq.Text := 'S= ' + FloatToStr(S);
end;

end.

```

- Lưu dự án vào thư mục: S:\TongS
- Biên dịch và chạy chương trình, ta có kết quả như sau:



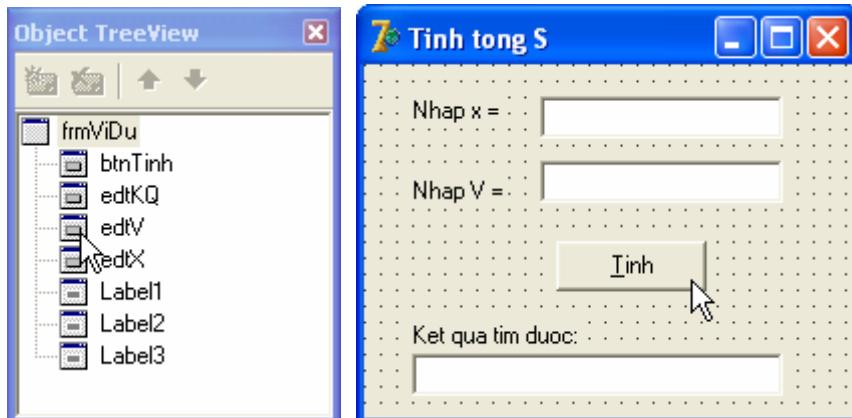
Hình 21: Kết quả chương trình

Ví dụ 16: Viết chương trình tìm số nguyên dương n lớn nhất sao cho tổng S thỏa biểu thức: $S = 1! + x^2 + 3! + x^4 + \dots + (n-1)! + x^n \leq V$

Với $x > 0$, $V > 0$ bất kỳ được nhập từ bàn phím. In kết quả S và n tìm được.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 22: Tên các đối tượng và giao diện chương trình

```

unit untTongCoDK;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmViDu = class(TForm)
    edtX: TEdit;
    Label1: TLabel;
    edtV: TEdit;
    Label2: TLabel;
    btnTinh: TButton;
    edtKQ: TEdit;
  end;

```

```

Label3: TLabel;
procedure btnTinhClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmViDu: TfrmViDu;

implementation
{$R *.dfm}

function GT(n: integer): extended;           // Hàm n!
var i:integer;
begin
  result:=1;
  for i:=1 to n do  result:=result*i;
end;

function LT(a:extended;  n: integer): extended; // Hàm a^n
var i:integer;
begin
  result:=1;
  for i:=1 to n do  result:=result*a;
end;

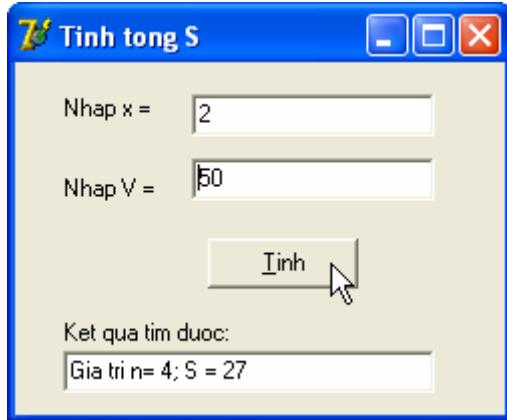
procedure TfrmViDu.btnTinhClick(Sender: TObject);
var n:integer;
  S, x, V: extended;
begin
  if TryStrToFloat(edtX.Text,x) and (x>0) and
    TryStrToFloat(edtV.Text,V) and (V>=1) then
  begin
    n:=0;  S:=0;
    while S <= V do
    begin
      inc(n);
      if odd(n) then S := S + GT(n)
      else S := S + LT(x,n);
    end;
    if odd(n) then S:= S-GT(n)
    else S := S - LT(x,n);
    n:= n-1;
    edtKQ.Text := 'Gia tri n= ' + IntToStr(n) + ';' S = ' +
  end;
end;

```

Chương 7: Chương trình con

```
        FloatToStr(S);  
    end;  
    else ShowMessage( 'Du lieu nhap khong thoa' );  
end;  
  
end.
```

- Lưu dự án vào thư mục: S:\TongCoDK
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 23: Kết quả chương trình

V. Chương trình con đệ quy

Một chương trình con mà trong quá trình thiết lập, nó sẽ gọi chính bản thân nó thì chương trình con có tính chất đệ quy (recursion).

Đệ quy sử dụng cấu trúc dữ liệu ngăn xếp (Stack) có tính chất là **vào trước - ra sau** trong quá trình đệ quy.

Ví dụ 17: Viết chương trình tính giai thừa theo cách đệ quy.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 24: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình tính giai thừa bằng phương pháp **đệ quy** như sau:

```

unit untGT_DeQuy;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics,
    Controls, Forms, Dialogs, StdCtrls;

type
    TfrmGT_DeQuy = class(TForm)
        edt_n: TEdit;
        lbl_n: TLabel;
        edt_KQ: TEdit;
        lblKQ: TLabel;
        btnTinh: TButton;
        btnKetThuc: TButton;
        procedure btnTinhClick(Sender: TObject);
        procedure btnKetThucClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    frmGT_DeQuy: TfrmGT_DeQuy;

implementation
{$R *.dfm}

function Giaithua(n: integer): extended;
begin
    if n=0 then
        Giaithua:=1           // truong hop neo
    else
        Giaithua:=n*Giaithua(n-1);   // de quy
    end;

procedure TfrmGT_DeQuy.btnTinhClick(Sender: TObject);
var num:integer;
begin
    try
        num := StrToInt(edt_n.Text);
        if num>=0 then
            begin

```

Chương 7: Chương trình con

```
lblKq.Caption:=edt_n.Text+'!=';
edt_KQ.Text:= FormatFloat('#,##0',Giaithua(num));
end;
else
begin
  ShowMessage('Khong tinh giai thua cua so am.');
  edt_n.Clear;
  edt_KQ.Clear;
  edt_n.SetFocus;
end;
except
begin
  ShowMessage('Gia tri ' + edt_n.Text + ' khong hop le.');
  edt_n.Clear;
  edt_Kq.Clear;
  edt_n.SetFocus;
end;
end;
end;

procedure TfrmGT_DeQuy.btnExitClick(Sender: TObject);
begin
  Close;
end;

end.
```

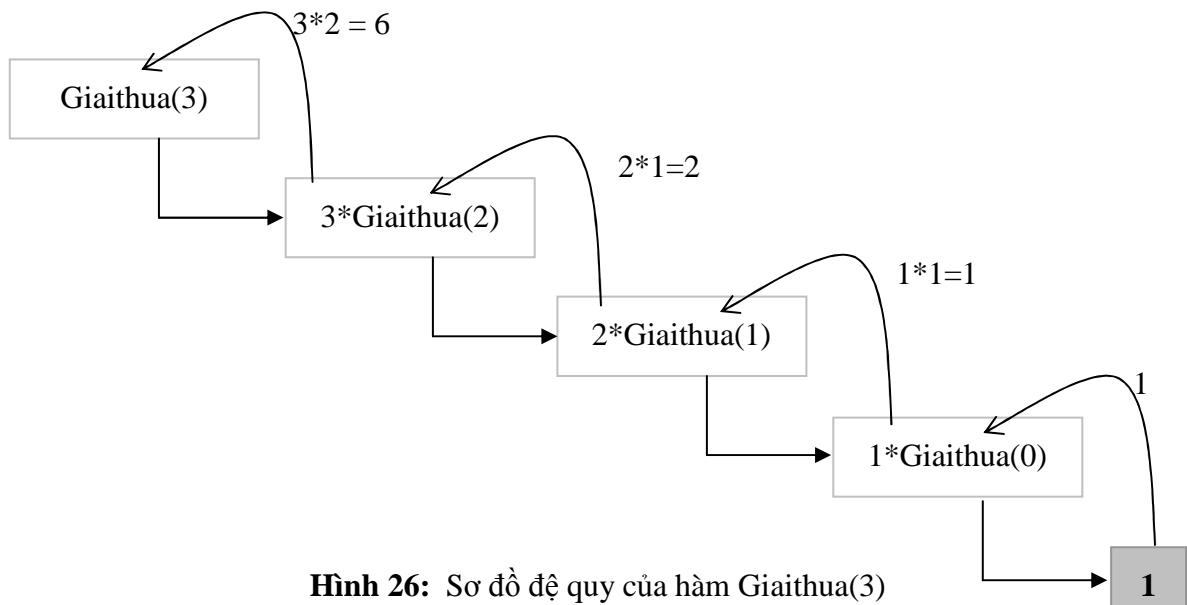
- Lưu dự án vào thư mục S:\ViduDeQuy
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 25: Chương trình đệ quy n!

- Sơ đồ minh họa tính đệ quy và tính kết quả của hàm đệ quy Giaithua như sau:

Giả sử ta nhập $n = 3$, thì chương trình sẽ gọi hàm đệ quy `Giaithua(3)` và ta có sơ đồ như sau:



Chú ý:

- Ưu điểm của thuật toán đệ quy là ngắn gọn. Nó có khả năng định nghĩa một tập hợp rất lớn các đối tượng bằng một số các câu lệnh hữu hạn. Thuật toán đệ quy có vẻ thích hợp cho các bài toán mà tự thân cấu trúc dữ liệu của nó đã được định nghĩa theo lối đệ quy.

- Có một số thuật toán đệ quy sử dụng cho các bài toán đơn giản có thể được thay thế bằng một thuật toán khác không tự gọi chúng, sự thay thế đó được gọi là *khử đệ quy*.

- Trong một số bài toán ta có thể giải theo 2 cách: thuật toán không đệ quy và thuật toán đệ quy. Thông thường, cách giải theo thuật toán không đệ quy thì tốt hơn so với thuật toán đệ quy vì đệ quy đòi hỏi thêm bộ nhớ và thời gian. Khi đó các thanh ghi được sử dụng cho lưu trữ và khi quay trở về phải khôi phục lại trạng thái cũ trong mỗi lần gọi đến chương trình con. Mức độ phức tạp có thể gia tăng khi trong chương trình con theo thuật toán đệ quy có chứa những chương trình con khác. Vì vậy, khi dùng đệ quy ta cần thận trọng, nhất là thuật toán này thường không cho ta thấy rõ trực tiếp toàn bộ quá trình giải các bước. Nói chung, chỉ khi nào không thể dùng thuật toán lặp ta mới nên sử dụng thuật toán đệ quy.

CHƯƠNG 8: KIỂU LIỆT KÊ, MIỀN CON, TẬP HỢP

I. Kiểu vô hướng liệt kê (Enumerated scalar type)

I.1. Khái niệm

Kiểu liệt kê là kiểu định nghĩa một tập hợp hữu hạn có thứ tự của các phần tử đơn giản bằng cách liệt kê danh sách các giá trị nằm trong cặp dấu ngoặc đơn.

I.2. Cách khai báo: Có hai cách khai báo là gián tiếp và trực tiếp

I.2.1. Khai báo gián tiếp

Sử dụng từ khóa type để định nghĩa ra kiểu dữ liệu mới, rồi khai báo biến thông qua kiểu dữ liệu mới vừa định nghĩa ra.

```
type
    Tên_kiểu_liệt_kê = (danh_sách_giá_trị_liệt_kê);
var
    danh_sách_biến: tên_kiểu_liệt_kê;
```

Ví dụ 1: Khai báo gián tiếp

```
type
    Days = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
    Colors = (Red, Yellow, Green, White, Blue, Black);
var
    Ngay : Days;
    Mauve: Colors;
```

Với các khai báo trên chúng ta có định nghĩa hai kiểu liệt kê có tên là Days và Colors. Kiểu liệt kê Days chứa 7 phần tử có giá trị được liệt kê như trên. Sau đó chúng ta khai báo biến Ngay có kiểu là kiểu liệt kê Days.

I.2.2. Khai báo trực tiếp

Kiểu dữ liệu của biến được khai báo trực tiếp sau tên biến.

```
var
    danh_sách_biến: (danh_sách_giá_trị_liệt_kê);
```

Ví dụ 2: Khai báo trực tiếp

```
var
    Ngay : (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
    Mauve : (Red, Yellow, Green, White, Blue, Black);
```

Ta có thể gán cho biến các giá trị của kiểu tương ứng:

Ngay := Mon;

MauVe := Red;

Biến theo định nghĩa của kiểu nào chỉ nhận giá trị của kiểu đó mà thôi.

Theo khai báo như ví dụ ở trên, ta không thể có $MauVe := Mon;$

Kiểu vô hướng liệt kê là một kiểu đếm được.

Theo định nghĩa kiểu vô hướng liệt kê, thứ tự danh sách giá trị liệt kê được ngầm đánh số tăng tuyến tính bắt đầu từ số 0 trở đi theo thứ tự từ trái sang phải. Như vậy, ở ví dụ trên: Sun < Mon < Tue < Wed và Red < Yellow < Green ...

I.3. Một số hàm chuẩn áp dụng cho kiểu vô hướng

* Hàm thứ tự Ord (x)

Hàm này trả về thứ tự của giá trị x trong kiểu vô hướng liệt kê.

Ví dụ 3: Theo ví dụ trên, ta có:

Ord (Sun) = 0	là đúng vì Sun có thứ tự là 0
Ord (Mon) = 1	là đúng vì Mon có thứ tự là 1
Ord (Green) = 3	là sai vì Green có thứ tự là 2
Ord (n) = n	trong đó n là một giá trị kiểu Longint

* Hàm Pred (x)

Hàm này trả về giá trị đúng trước x của kiểu vô hướng liệt kê.

Ví dụ 4: Theo ví dụ trên, ta có:

Pred (Mon) = Sun
Pred (Green) = Yellow
Pred (n) = n - 1

* Hàm Succ (x)

Hàm này cho giá trị đúng sau x trong định nghĩa kiểu của x.

Ví dụ 5: Theo ví dụ trên, ta có:

Succ (Mon) = Tue
 Succ (Green) = White
 Succ (n) \equiv n +

* Hàm chuyển một số nguyên thành một giá trị vô hướng

Tên hàm này chính là tên kiểu vô hướng mà ta đã khai báo trước.

Ví dụ 6: Theo ví dụ trên, ta có:

Days(2)	= Tue
Colors(3)	= White
Longint (n)	= n

II. Kiểu miền con (Subrange types)

II.1. Khái niệm

Khi khai báo một số trường hợp, ví dụ tuổi thọ của người nếu ta khai báo:

var

TuoiTho: Integer ; //Integer có miền xác định -2147483648..2147483647

thì sẽ tồn ô nhớ vì Integer có kích thước 4. Làm như vậy sẽ không cần thiết vì tuổi con người chỉ trong khoảng từ 0 đến 150.

Trong Delphi cho phép ta xác định một biến lấy giá trị trong một khoảng nào đó được giới hạn ở một hằng cận dưới (Low) và một hằng cận trên (Hight). Hai giá trị này phải cùng một kiểu vô hướng đếm được và hằng cận trên có giá trị lớn hơn hằng cận dưới. Khai báo như vậy gọi là khai báo kiểu miền con (Subrange type). Hay nói một cách khác kiểu miền con chính là một định nghĩa thu hẹp của các kiểu có thứ tự khác. Trong lúc chạy chương trình, ta có thể kiểm tra giá trị của biến không được vượt ra khỏi miền giá trị của miền con.

II.2. Cách khai báo

II.2.1. Khai báo gián tiếp

```

type
    tên_kiểu_miền_con = hằng_cận_dưới .. hằng_cận_trên;
var
    danh_sách_biến: Tên_kiểu_miền_con;
  
```

Ví dụ 7: Khai báo gián tiếp

type

TuoiTho = 0 .. 150;

GioiTinh = 0..1; // nữ = 0, nam = 1

var

tuoit: TuoiTho;

phai: GioiTinh;

II.2.2. Khai báo trực tiếp

```
var
```

```
danh_sách_biến: hàng_cận_dưới .. hàng_cận_trên;
```

Ví dụ 8: Khai báo trực tiếp

```
var
```

```
tuoi : 0 .. 150;
```

```
phai: 0..1; // nő = 0, nam =1
```

III. Kiểu tập hợp (Set)

III.1. Khái niệm

Kiểu tập hợp là một tập các giá trị cùng kiểu đếm được và được gọi là các phần tử của tập hợp. Số phần tử tối đa trong một tập hợp là **256**, bao gồm cả tập hợp rỗng. Khái niệm tập hợp trong Delphi tương tự như khái niệm tập hợp trong Toán học.

III.2. Cách khai báo

III.2.1. Khai báo gián tiếp

```
type
```

```
tên_kiểu_tập_hợp = Set of kiểu_cơ_bản;
```

```
var
```

```
danh_sách_biến : Tên_kiểu_tập_hợp;
```

Ví dụ 9: Khai báo gián tiếp

```
type
```

Day100 = 1..100; //Định nghĩa một dãy con để dùng làm kiểu cơ bản

TapSoNguyen = **Set of** Day100;

```
var
```

TapHop1, TapHop2: TapSoNguyen;

III.2.2. Khai báo trực tiếp

```
var
```

```
danh_sách_biến : Set of kiểu_cơ_bản;
```

Ví dụ 10: Khai báo trực tiếp

```
var
```

TapHop1, TapHop2 : **Set of** 0..250; // khai báo đúng

Chu	: Set of char;	// khai báo đúng
So	: Set of 0 .. 9;	// khai báo đúng
X	: Set of integer; // sai, vì vượt quá giới hạn tối đa 256 phần tử	
Z	: Set of real; // sai, do kiểu dữ liệu không rời rạc/dense được	

III.3. Mô tả tập hợp

Một tập hợp được mô tả bằng cách liệt kê các phần tử của tập hợp, chúng cách nhau bằng một dấu phẩy (,) và được đặt giữa hai dấu mốc vuông [], các phần tử có thể là hằng, biến hoặc biểu thức.

Ví dụ 11: Các ví dụ tập hợp trong Delphi

[]	// tập hợp rỗng, không có các phần tử
[5 .. 15]	// tập hợp các chữ số nguyên từ 5 đến 15
[1, 3, 5]	// tập hợp 3 số 1, 3 và 5
[i, i + j*2, 4, 5]	{tập hợp các biến nguyên gồm số 4, 5 và các số nhận từ i, i + j*2 với i, j là 2 biến byte}

III.4. Một số phép toán trên kiểu tập hợp

III.4.1. Phép gán

Ta có thể gán giá trị các tập đã được mô tả vào các biến tập cùng kiểu. Riêng tập hợp rỗng có thể gán cho mọi biến kiểu tập hợp khác nhau.

Ví dụ 12: Các phép gán tập hợp

```
So:=[];
// tập rỗng

TapHop1 := [0,1,3,5,7,9];
TapHop2 := [0,2,4,6,8,9];
TapNguyenAm:=['A', 'a', 'E', 'e', 'T', 'i', 'O', 'o', 'U', 'u']; // tập các nguyên âm
```

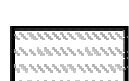
III.4.2. Phép hợp

Hợp của 2 tập hợp A và B là một tập hợp chứa tất cả các phần tử của tập A hoặc B hoặc cả A và B.

Ký hiệu của phép hợp là dấu cộng (+). Phép hợp có tính giao hoán:

$$A + B = B + A$$

Ta có thể mô tả phép hợp qua hình ảnh sau :



A



B



A + B

Hình 1: Hợp 2 tập

Ví dụ 13: Hợp của 2 tập

```
A := [0,1,3,5,7,9];
B := [0,2,4,6,8,9];
C := A + B;           // C = [0,1,2,3,4,5,6,7,8,9]
```

III.4.3. Phép giao

Giao của 2 tập hợp A và B là một tập chứa các phần tử của cả A và cả B.

Ký hiệu $A * B$. Phép giao cũng có tính giao hoán, nghĩa là $A * B = B * A$

Minh họa như sau:



Hình 2: Giao 2 tập

Ví dụ 14: Giao của 2 tập.

Với tập hợp $A = [0,1,3,5,7,9]$ và tập $B = [0,2,4,6,8,9]$:

```
D := A * B ; // tập D = [0,9]
```

III.4.4. Phép hiệu

Hiệu của 2 tập hợp A và B, ký hiệu là $A - B$, là một tập hợp chứa các phần tử chỉ thuộc A mà không thuộc B.

Lưu ý : $A - B$ thì khác $B - A$.

Ví dụ 15: Hiệu của 2 tập:

```
A := [3 .. 7] ;
B := [1.. 6, 10, 15] ;
Thì: A - B = [7], còn B - A = [1, 2, 10, 15]
```

III.4.5. Phép thuộc IN

Phép thuộc **in** cho phép thử (kiểm tra) xem một giá trị nào đó thuộc về một tập hay không? Phép thuộc **in** cho kết quả có kiểu **boolean**. Nếu đúng thì nó sẽ cho kết quả là **true**, ngược lại là **false**.

Ví dụ 16: Phép toán thuộc

```
var
    Kytu : char;
    NguyenAm: Set of char;
```

Sau đó ta gán:

Kytu := 'a';

NguyenAm:= ['A', 'E', 'I', 'U', 'O', 'a', 'e', 'i', 'u', 'o'];

Thì phép toán: Kytu **in** NguyenAm sẽ cho kết quả là true

III.4.6. Các phép so sánh =, <>, <=, và >=

Muốn so sánh hai tập hợp với nhau thì chúng phải có cùng kiểu cơ bản. Kết quả của các phép so sánh là giá trị kiểu Boolean, tức là True hoặc False.

Hai tập hợp A và B gọi là **bằng nhau** ($A = B$) chỉ khi chúng có các phần tử giống với nhau từng đôi một (không kể thứ tự sắp xếp các phần tử trong hai tập). Ngược lại của phép so sánh bằng nhau ($=$) là phép so sánh **khác nhau** (\neq). Nghĩa là, nếu $A = B$ là True thì $A \neq B$ sẽ là False và ngược lại.

Phép so sánh nhỏ hơn hoặc bằng (\leq) của $A \leq B$ sẽ cho kết quả là True nếu mọi phần tử có trong A đều có trong B. Định nghĩa này cũng tương tự như lớn hơn hoặc bằng. Với $A \geq B$ thì mọi phần tử của B đều có trong A, kết quả này True, ngược lại là False.

Chú ý: Trong Delphi không có phép so sánh nhỏ hơn ($<$) và lớn hơn ($>$). Để kiểm tra xem tập A có thực sự nằm trong tập B hay không (A nhỏ hơn B), ta phải sử dụng thêm các phép logic như sau:

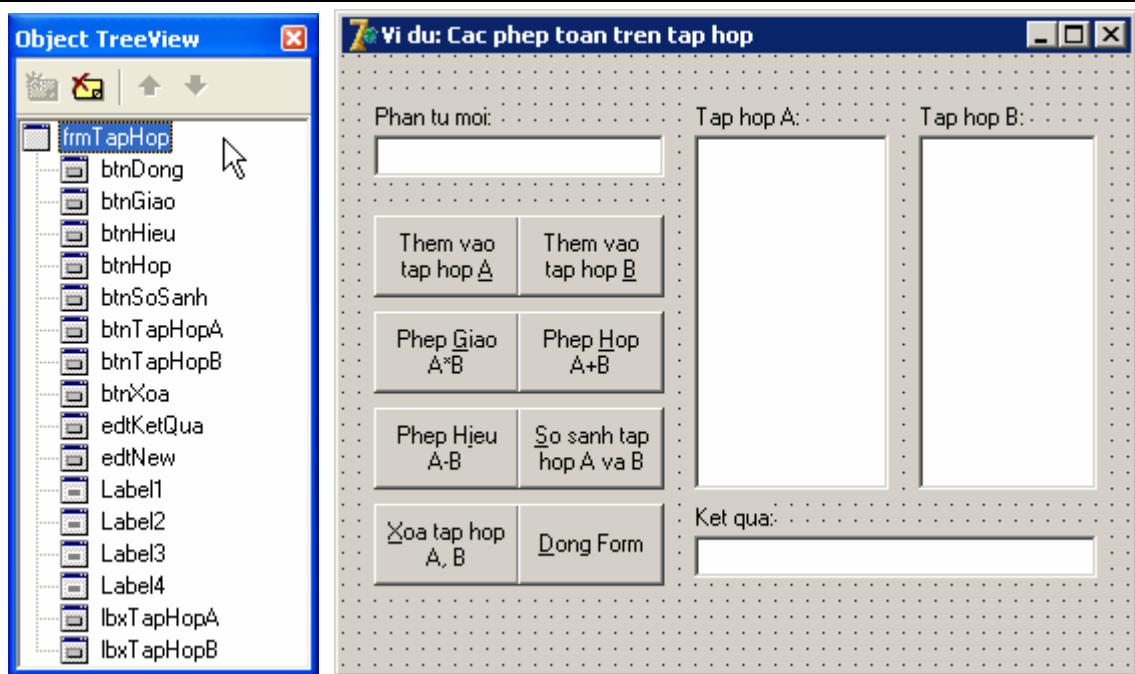
if (a \neq b) and (a \leq b) then ShowMessage('a < b');

$$\left. \begin{array}{l} tap1 := ['a']; \\ tap2 := ['a'..'c']; \end{array} \right\} \Rightarrow \left. \begin{array}{l} tap1 \neq tap2 \\ tap1 \leq tap2 \end{array} \right\} \Rightarrow tap1 < tap2$$

Ví dụ 17: Nhập các phần tử kiểu nguyên thuộc miền giá trị từ 0 đến 100 cho tập A và tập B. Nếu nhập phần tử trùng (đã tồn tại) thì không thêm vào tập hợp và thông báo cho biết đã có. Hãy cho biết kết quả của các phép toán trên tập hợp.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 3: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình như sau:

```

unit untTapHop;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmTapHop = class(TForm)
    edtNew: TEdit;
    Label1: TLabel;
    btnTapHopA: TButton;
    btnTapHopB: TButton;
    Label4: TLabel;
    lbxTapHopA: TListBox;
    lbxTapHopB: TListBox;
    btnXoa: TButton;
    btnGiao: TButton;
    edtKetQua: TEdit;
    Label2: TLabel;
    btnHop: TButton;
    btnHieu: TButton;
    btnSoSanh: TButton;
    Label3: TLabel;
    btnDong: TButton;
  procedure btnTapHopAClick(Sender: TObject);
end;

```

```

procedure FormCreate(Sender: TObject);
procedure btnTapHopBClick(Sender: TObject);
procedure btnXoaClick(Sender: TObject);
procedure btnGiaoClick(Sender: TObject);
procedure btnHopClick(Sender: TObject);
procedure btnHieuClick(Sender: TObject);
procedure btnSoSanhClick(Sender: TObject);
procedure btnDongClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmTapHop: TfrmTapHop;
  taphopA, taphopB, taphopC: Set of byte;

implementation

{$R *.dfm}

procedure TfrmTapHop.FormCreate(Sender: TObject);
begin
  taphopA:=[ ] ; //Khởi tạo, tập rỗng.
  taphopB:=[ ];
  taphopC:=[ ];
end;

procedure TfrmTapHop.btnTapHopAClick(Sender: TObject);
var pt:integer;
begin
  if TryStrToInt(edtNew.Text,pt) and (pt>=0) and (pt<=100) then
    if (pt in taphopA) then
      ShowMessage('Phan tu '+edtNew.Text+' da co trong tap hop
A.')
    else
      begin
        taphopA:=taphopA+[pt];
        lbxTapHopA.Items.Add(IntToStr(pt));
      end
  else ShowMessage('Chuong trinh chi xet cac phan tu co gia tu 0
den 100.');
  edtNew.SetFocus;
end;

```

Chương 8: Kiểu Liet kê, Miền con, Tập hợp

```
procedure TfrmTapHop.btnAddClick(Sender: TObject);
var pt:integer;
begin
  if TryStrToInt(edtNew.Text,pt) and (pt>=0) and (pt<=100) then
    if (pt in taphopB) then
      ShowMessage('Phan tu '+edtNew.Text+' da co trong tap hop
B.');
    else
      begin
        taphopB:=taphopB+[pt];
        lbxTapHopB.Items.Add(IntToStr(pt));
      end
    else ShowMessage('Chuong trinh chi xet cac phan tu co gia tu 0
den 100.');
    edtNew.SetFocus;
  end;

procedure TfrmTapHop.btnDeleteClick(Sender: TObject);
begin
  taphopA:=[ ]; taphopB:=[ ]; taphopC:=[ ];
  edtNew.Clear;
  edtNew.SetFocus;
  lbxTapHopA.Clear;
  lbxTapHopB.Clear;
  edtKetQua.Clear;
end;

procedure TfrmTapHop.btnAddClick(Sender: TObject);
var i:integer;
begin
  edtKetQua.Clear;
  taphopC :=taphopA*taphopB;
  for i:=0 to 100 do //chi xet cac pt tap hop tu 0..100
    if (i in taphopC) then
      edtKetQua.text := edtKetQua.text+' '+ IntToStr(i);
end;

procedure TfrmTapHop.btnAddClick(Sender: TObject);
var i:integer;
begin
  edtKetQua.Clear;
  taphopC :=taphopA + taphopB;
  for i:=0 to 100 do
    if (i in taphopC) then
      edtKetQua.text := edtKetQua.text+' '+ IntToStr(i);
end;
```

```

procedure TfrmTapHop.btnHieuClick(Sender: TObject);
var i:integer;
begin
  edtKetQua.Clear;
  taphopC := taphopA - taphopB;
  if taphopC = [] then
    edtKetQua.text := 'Tap hop rong.'
  else
    for i:=0 to 100 do
      if (i in taphopC) then
        edtKetQua.text := edtKetQua.text+' '+ IntToStr(i);
end;

procedure TfrmTapHop.btnSoSanhClick(Sender: TObject);
begin
  if (taphopA <> taphopB) then
    if (taphopA <= taphopB) then
      edtKetQua.text := 'Tap hop A < Tap hop B'
    else
      if (taphopA >= taphopB) then
        edtKetQua.text := 'Tap hop A > Tap hop B'
      else
        edtKetQua.text := 'Tap hop A <> Tap hop B'
  else
    edtKetQua.text := 'Tap hop A = Tap hop B';
end;

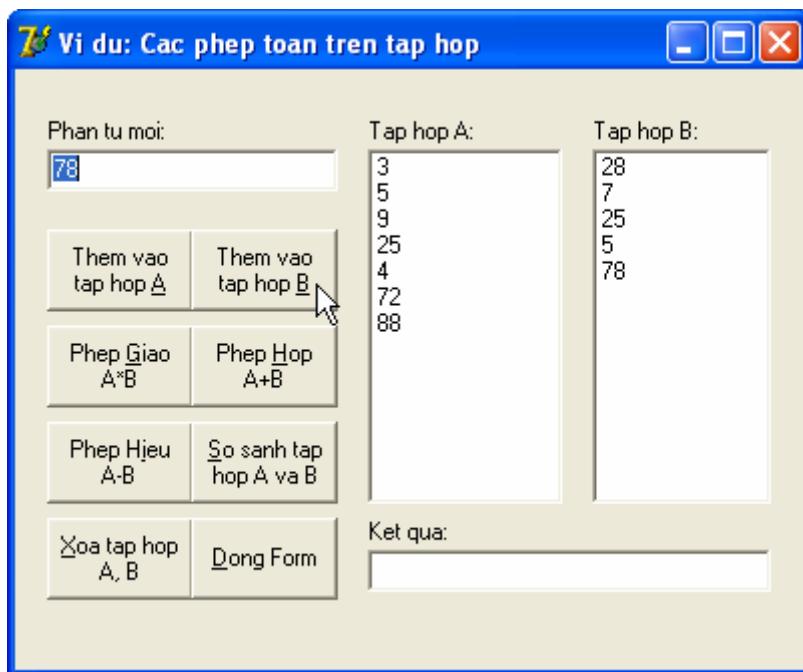
procedure TfrmTapHop.btnDongClick(Sender: TObject);
begin
  Close;
end;

end.

```

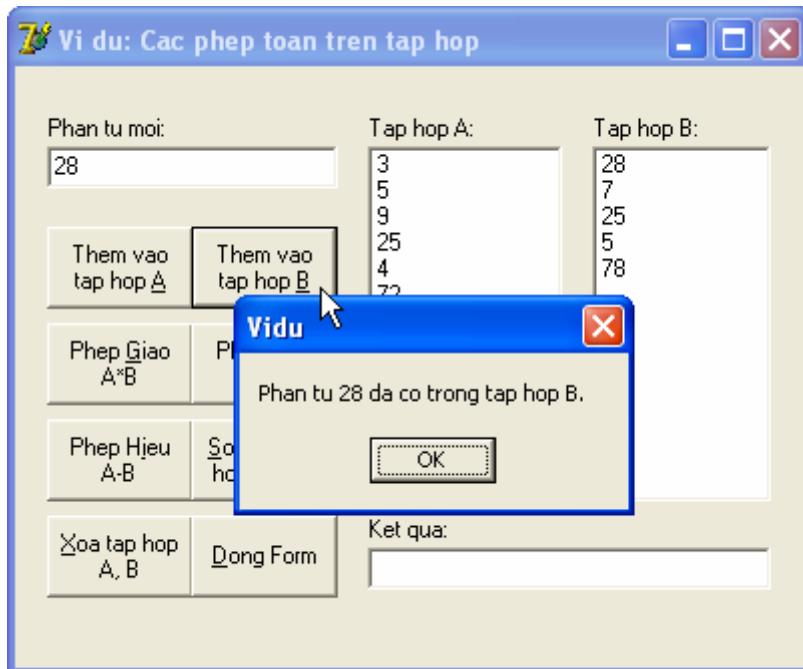
- Lưu dự án vào thư mục **S:\ViduTapHop**
- Biên dịch và chạy chương trình, ta có kết quả như sau:

+ Nhập các phần tử cho tập A và B:



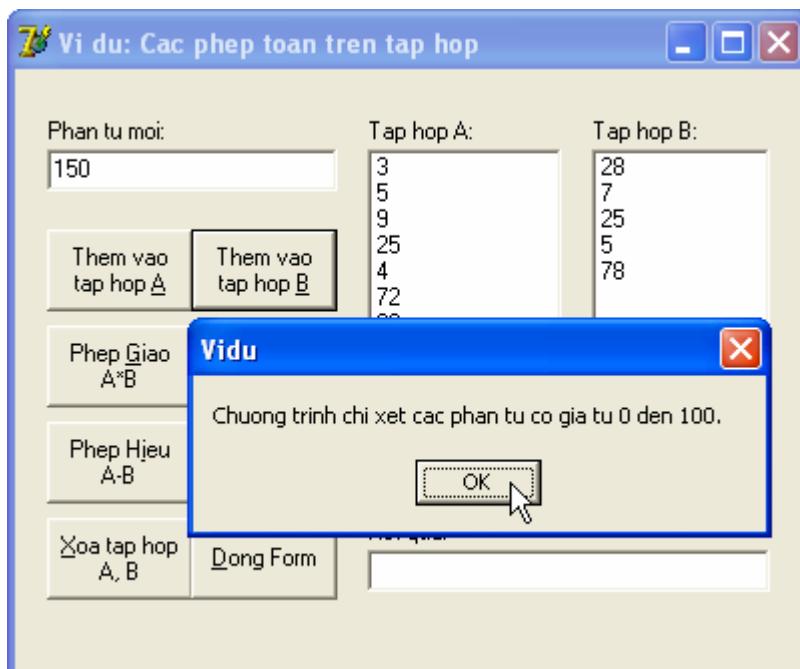
Hình 4: Nhập các phần tử cho tập A và B

+ Nếu nhập phần tử trùng sẽ thông báo:



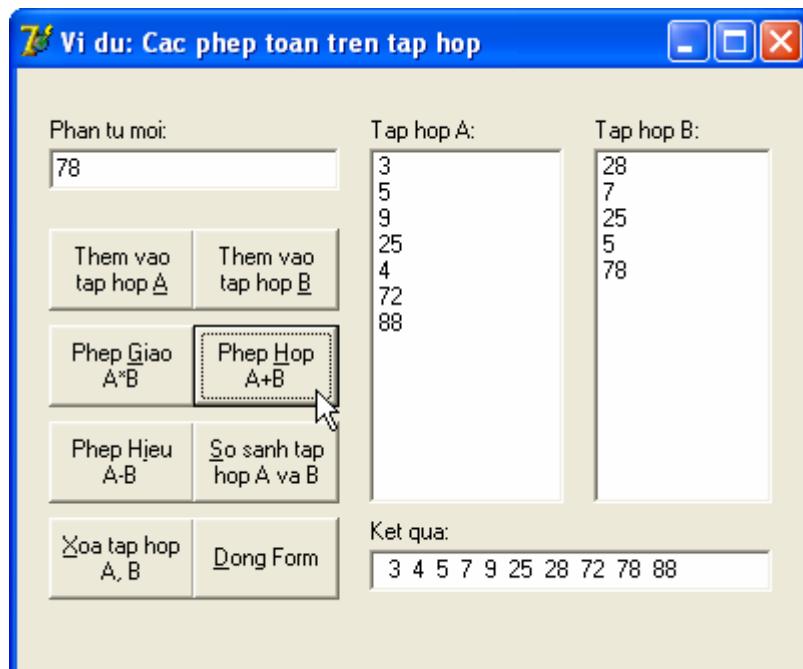
Hình 5: Thông báo nhập phần tử đã tồn tại

+ Nếu nhập vượt khỏi miền giá trị đang xét, thì thông báo:



Hình 6: Nhập vượt khỏi miền giá trị

+ Hợp của 2 tập:



Hình 7: Kết quả hợp của 2 tập.

+ Tính giao của 2 tập:

Hình 8: Kết quả giao của 2 tập.

+ Kết quả so sánh giữa 2 tập A và B.

Hình 9: Kết quả so sánh

CHƯƠNG 9: KIỂU MẢNG

I. Khái niệm về mảng (Array-type data)

Mảng là một kiểu dữ liệu gồm một dãy hữu hạn các phần tử có cùng kiểu, gọi là kiểu cơ bản. Mảng được tổ chức theo một trật tự xác định. Số phần tử của mảng được khai báo ngay từ khi định nghĩa ra mảng (mảng tĩnh) hoặc có thể khai báo và cấp phát một cách linh hoạt trong lúc chương trình đang chạy (run-time) theo nhu cầu sử dụng (mảng động).

II. Mảng tĩnh (Static array)

II.1. Mảng một chiều (One-Dimensional array)

Mảng một chiều có thể được hiểu như một danh sách các phần tử có cùng kiểu. Mỗi phần tử của mảng được xác định và được truy nhập trực tiếp thông qua tên mảng cùng với chỉ dẫn truy nhập được để giữa hai ngoặc vuông [].

Ví dụ chúng ta có một danh sách có tên là DaySo có n phần tử (Với n là một hằng số đã xác định trước). Vậy List chính là một mảng 1 chiều. Các phần tử của List mang các tên List[0], List[1], List[2], ..., List[n-1], và có thể minh họa như hình sau:



Hình 1: Hình ảnh minh họa mảng một chiều.

II.1.1. Khai báo gián tiếp:

```

type
    Kiểu_mảng = array [tập_chỉ_số] of kiểu_phần_tử;
var
    Danh_sách_biéñ: Kiểu_mảng;
  
```

Chú ý: tập chỉ số phải là kiểu có thứ tự (rồi rạc, đếm được), chẳng hạn như kiểu: Ký tự, kiểu miền con, số nguyên...

Ví dụ 1: Khai báo gián tiếp

```

type
    DaySoNguyen = array [0 .. 99] of integer;
    DayKyTu = array [1 .. 20] of char;
var
    a, b: DaySoNguyen;
    c: DaySoNguyen;    d: DayKyTu;
  
```

Ý nghĩa:

- DaySoNguyen là kiểu mảng gồm 100 phần tử số nguyên có chỉ số từ 0 đến 99.
- Biến a, b và c là **cùng kiểu** DaySoNguyen.
- DayKyTu là kiểu mảng gồm 20 phần tử đánh số từ 1 .. 20 có kiểu là các ký tự. Biến d có kiểu là DayKyTu.

II.1.2. Khai báo trực tiếp**var**Danh_sách_biến : **array** [tập_chỉ_số] **of** kiểu_phần_tử;**Ví dụ 2:** Khai báo trực tiếp**var**

```
a, b      : array [1 .. 100] of integer;
c         : array [1 .. 100] of integer;
KyTu    : array [1 .. 20 ] of char;
Chon    : array [0 .. 9 ] of boolean;
```

- Biến a và b được gọi là **cùng kiểu**. Biến a và b là **khác kiểu** với biến c.

Lưu ý: nếu như chúng ta khai báo tạo ra một mảng tĩnh nhưng không gán trị cho các phần tử của mảng thì các phần tử mảng vẫn được cấp phát trong bộ nhớ, tuy nhiên chúng sẽ chứa các giá trị ngẫu nhiên, giống như các biến chưa được khởi tạo giá trị.

II.1.3. Truy xuất các phần tử của mảng

Mỗi phần tử của mảng được truy xuất thông qua **Tên biến mảng** cùng với chỉ số thứ i của mảng trong dấu ngoặc vuông []. Ví dụ tên biến mảng là A, khi viết A[7], ta hiểu nó là phần tử có chỉ số thứ 7 của mảng A.

Ví dụ, chúng ta có thể thực hiện phép gán giá trị 10 cho phần tử thứ bảy của A như sau: A[7]:=10;

Ta có thể gán hai biến mảng cùng kiểu cho nhau.

Ví dụ 3: Khai báo hai mảng x, x như sau:**var**

```
x,y    : array[0..10] of integer; // x, y cùng kiểu
```

Ta có phép gán sau là hoàn toàn hợp lệ: y[0]:=50; x:=y;

Lúc này các giá trị của x sẽ hoàn toàn giống nhau. Chúng ta cần chú ý là: nếu ta thay đổi các giá trị của biến y thì điều đó không làm ảnh hưởng đến x và ngược lại.

Tuy nhiên, nếu ta có khai báo:

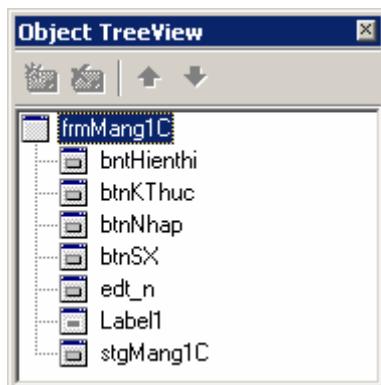
```
var      x, y    : array[0..10] of integer;
          z        : array[0..10] of integer;
```

Thì lệnh gán: x:=z; là không hợp lệ vì Delphi xem biến x và z trong trường hợp này là **khác kiểu**.

Ví dụ 4: Viết chương trình nhập vào một dãy số nguyên có n phần tử. Sắp xếp dãy số vừa nhập theo thứ tự tăng dần.

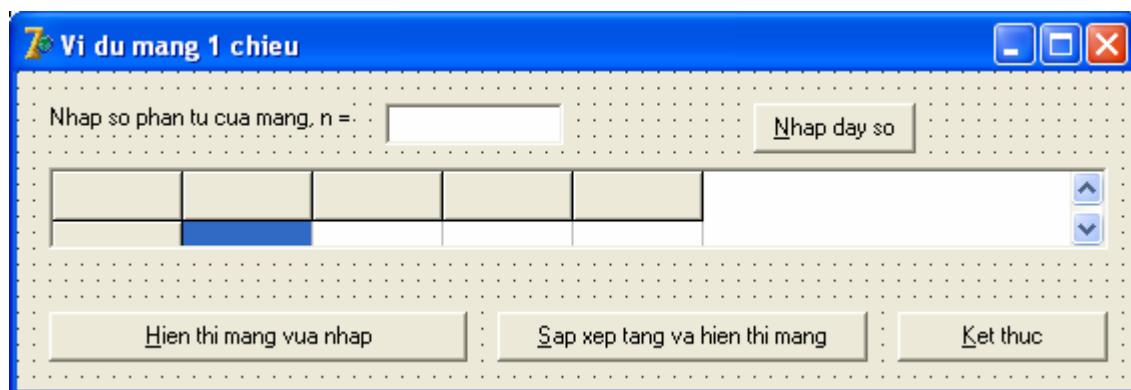
Hướng dẫn:

- Tạo một dự án mới **File/New/Application**
- Tên (name) của các đối tượng trên form như hình sau:



Hình 2: Tên các đối tượng

- Thiết kế giao diện form:



Hình 3: Thiết kế giao diện chương trình

- Đoạn code của chương trình như sau:

```
unit untMang1C;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, Grids, StdCtrls;
type
  TfrmMang1C = class(TForm)
    stgMang1C: TStringGrid;
    edt_n: TEdit;
    Label1: TLabel;
    btnNhap: TButton;
    btnSX: TButton;
    btnKThuc: TButton;
  end;
```

```

bntHienthi: TButton;
procedure btnNhaphClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure bntHienthiClick(Sender: TObject);
procedure btnKThucClick(Sender: TObject);
procedure btnSXClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMang1C: TfrmMang1C;
  n:integer;           // so phan tu mang
  a: array [0..100] of integer; // mang 1 chieu

implementation
{$R *.dfm}

procedure TfrmMang1C.btnNhaphClick(Sender: TObject);
var i: byte;
begin
  if (tryStrToInt(edt_n.Text,n)) and (n>0) then
    begin
      for i:=0 to n-1 do
        a[i]:=StrToInt(InputBox('Hop nhap lieu','Phan tu
                                a['+IntToStr(i)+']= ',''));
      stgMang1C.ColCount :=n;
    end
  else ShowMessage('Ban nhap gia tri n KHONG hop le.');
end;

procedure TfrmMang1C.FormCreate(Sender: TObject);
begin
  stgMang1C.FixedRows:=0;
  stgMang1C.FixedCols:=0;
  stgMang1C.RowCount:=1;
end;

procedure TfrmMang1C.bntHienthiClick(Sender: TObject);
var i: byte;
begin
  for i:= 0 to n-1 do stgMang1C.Cells[i,0]:=IntToStr(a[i]);
end;

```

Chương 9: Kiểu Mảng

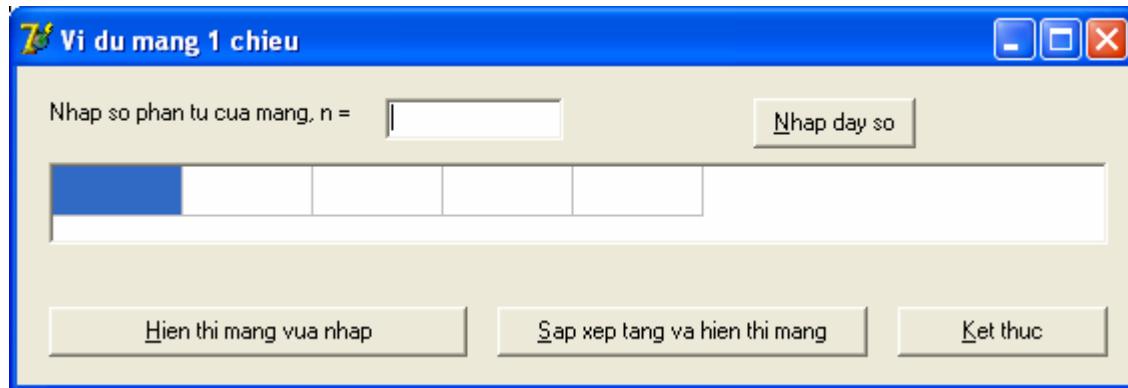
```
procedure TfrmMang1C.btnKThucClick(Sender: TObject);
begin
  Close;
end;

procedure TfrmMang1C.btnSXClick(Sender: TObject);
var i, j: byte;
  temp: integer;
begin
  for i:=0 to n-2 do
    for j:=i+1 to n-1 do
      if a[i]>a[j] then      // hoán vị
      begin
        temp:=a[i];
        a[i]:=a[j];
        a[j]:=temp;
      end;

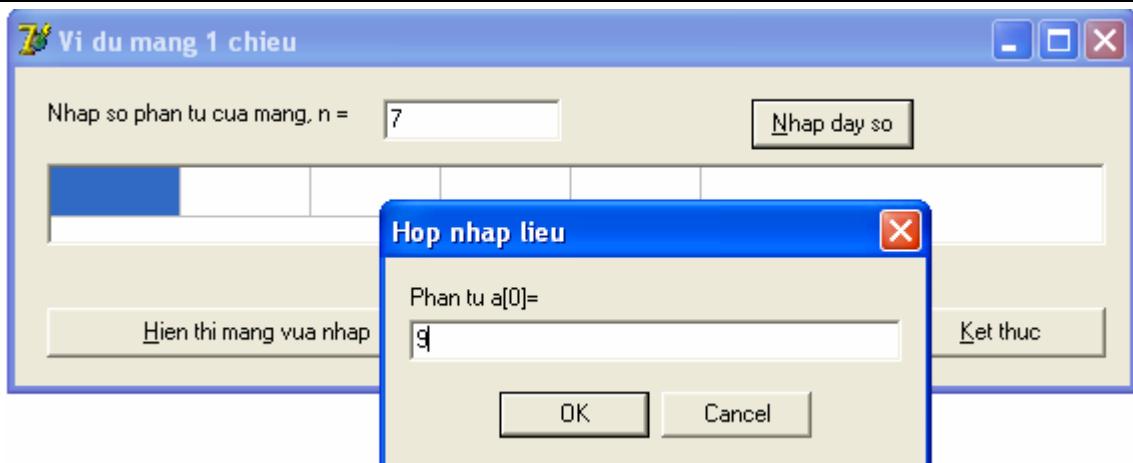
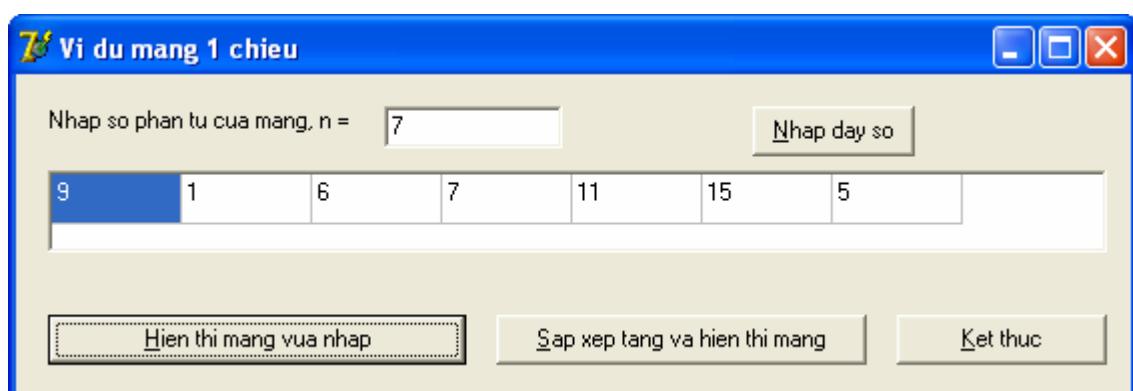
  //goi lai TTSK de hien thi mang da SX
  frmMang1C.bntHienthiClick(Sender);
end;

end.
```

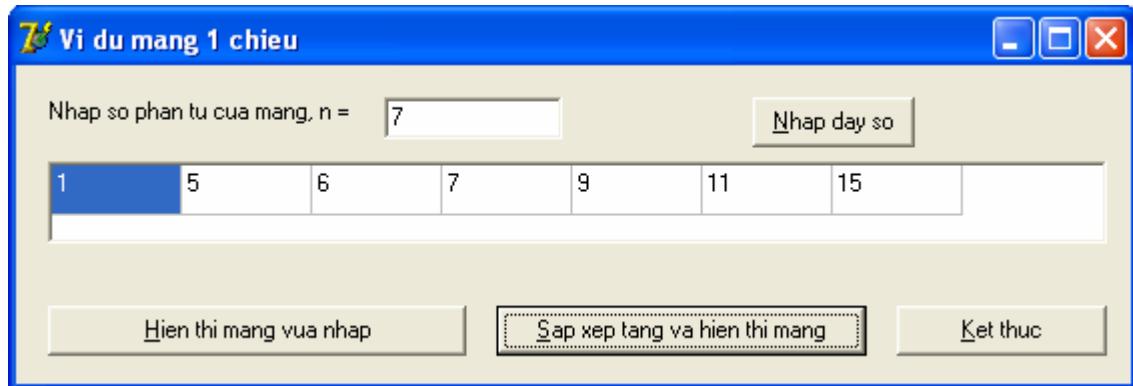
- Lưu dự án vào thư mục **S:\ViduMang1C**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 4: Chương trình bắt đầu chạy

Hình 5: Nhập số phần tử n, và Click nút Nhập dãy số

Hình 6: Hiển thị mảng vừa nhập



Hình 7: Sắp xếp tăng và hiển thị mảng

II.2. Mảng nhiều chiều (Multi-Dimensional array)

Trong một số bài toán thực tế, người ta cần sử dụng các mảng nhiều hơn một chiều, gọi là mảng nhiều chiều. Mảng nhiều chiều còn gọi là mảng của mảng.

Chúng ta hãy xem xét một ví dụ sau:

Phòng Đào tạo quản lý điểm của sinh viên. Trong khoá 30 chặng hạn, người ta tạo ra một mảng 2 chiều: ví dụ một chiều là MÃ SỐ của sinh viên, chiều còn lại là các MÔN

Chương 9: Kiểu Mảng

HỌC (dạng kiểu vô hướng liệt kê), ta có thể hình dung dạng của mảng ghi điểm (tên mảng là ghi_diem) như sau:

MASV \ Môn	Triet	TinHoc	Ly	...	AnhVan
0001	7	8	5	...	10
0002	6	10	6	...	7
0003	8	9	3	...	9
.....

Hình 8: Minh họa mảng hai chiều

Với ví dụ trên, muốn nhập điểm một sinh viên nào đó ta phải khai báo hai tham số là Mã số sinh viên và môn học.

Tương tự, cũng với các khóa kế tiếp theo học những môn như vậy, ta sẽ tạo ra mảng nhiều chiều như hình vẽ minh họa sau:

Khóa					
K30	1	K32			Môn
SVK30\ Môn	Toan		AV	...	
0001	7	8	10	..	
0002	8	10	9	..	
0003	2	9	7		
.....	

Hình 9: Minh họa mảng ba chiều

Trong trường hợp này, muốn biết điểm một sinh viên nào đó ta phải khai báo 3 tham số: Khoa học, sinh viên và môn học, chẳng hạn:

ghi_diem[K31,0001,AV] nhập điểm 10,...

II.2.1. Khai báo gián tiếp

type

```
Kiểu_mảng = array [tập_chỉ_số_1, tập_chỉ_số_2, ...,
                    tập_chỉ_số_n] of kiểu_phần_tử;
```

var

```
Danh_sách_biéń : Kiểu_mảng;
```

II.2.2. Khai báo trực tiếp

var

Danh_sách_biéñ : **array** [tập_chỉ_số_1, ..., tập_chỉ_số_n] **of** kiểu_phàn_tử;

Trong giáo trình này chỉ đề cập chi tiết về mảng hai chiều.

II.3. Mảng hai chiều

Mảng hai chiều là mảng nhiều chiều nhưng chỉ có hai tập chỉ số. Một hình ảnh quen thuộc của mảng hai chiều đó chính là thành phần StringGrid mà giáo trình đã đề cập ở chương 5.

II.3.1. Khai báo gián tiếp

type

Kiểu_mảng = **array** [tập_chỉ_số_1, tập_chỉ_số_2] **of** kiểu_phàn_tử;

var

Danh_sách_biéñ : Kiểu_mảng;

Hoặc:

type

Kiểu_mảng = **array** [tập_chỉ_số_1] **of** **array** [tập_chỉ_số_2] **of** kiểu_phàn_tử;

var

Danh_sách_biéñ : Kiểu_mảng;

Ví dụ 5: Khai báo gián tiếp mảng 2 chiều

type

MaTran = **array** [1 .. 5, 1 .. 10] **of** integer;

var

a : MaTran;

Lệnh trên khai báo một kiểu mảng có tên là Matran, có kiểu các phần tử là kiểu số nguyên. Đây là một mảng 2 chiều, chiều thứ nhất có các chỉ số từ 1 đến 5, chiều thứ hai có các chỉ số từ 1 đến 10, tổng cộng ta có (5 x 10) phần tử. Và ta có một biến a là biến có kiểu Matran.

Ví dụ trên cũng có thể được khai báo tương đương với:

type

Matran = **array** [1 .. 5] **of** **array** [1 .. 10] **of** integer ;

var

a : Matran;

II.3.2. Khai báo trực tiếp

```
var
```

```
Danh_sách_biéñ : array [tập_chỉ_số_1, tập_chỉ_số_2] of kiêu_phàn_tử;
```

Hoặc:

```
var
```

```
Danh_sách_biéñ : array [tập_chỉ_số_1] of array[tập_chỉ_số_2] of kiêu_phàn_tử;
```

Ví dụ 6: Khai báo một biến a có 5 dòng và 10 cột kiểu là integer như sau:

```
var
```

```
a : array [1 .. 5, 1 .. 10] of integer;
```

Hoặc ta cũng có thể khai báo như sau:

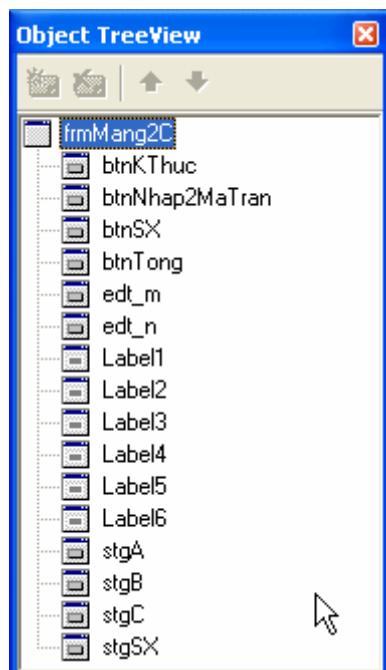
```
var
```

```
a : array [1 .. 5] of array [1 .. 10] of integer; // mảng của mảng.
```

Ví dụ 7: Viết chương trình nhập ngẫu nhiên 2 ma trận A và B có m dòng, n cột với các phần tử là số nguyên trong khoảng [0..500]. Tính cộng hai ma trận C = A+B, và sắp xếp ma trận C theo thứ tự giảm dần theo **từng hàng**.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như trong cửa sổ Object TreeView sau:



Hình 10: Tên các đối tượng

- Thiết kế giao diện form:

Hình 11: Giao diện chương trình

- Đoạn code của chương trình như sau:

```
unit untMang2C;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, Grids, StdCtrls;

type
  TfrmMang2C = class(TForm)
    stgB: TStringGrid;
    edt_m: TEdit;
    Label1: TLabel;
    btnNhap2MaTran: TButton;
    btnSX: TButton;
    btnKThuc: TButton;
    btnTong: TButton;
    Label2: TLabel;
    stgA: TStringGrid;
    edt_n: TEdit;
    Label3: TLabel;
    Label4: TLabel;
  end;
```

```

stgC: TStringGrid;
Label5: TLabel;
stgSX: TStringGrid;
Label6: TLabel;
procedure btnNhaph2MaTranClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure btnTongClick(Sender: TObject);
procedure btnKThucClick(Sender: TObject);
procedure btnSXClick(Sender: TObject);

private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmMang2C: TfrmMang2C;
  m, n:integer;      // so hang m, so cot n
  a, b, c: array [0..10,0..10] of integer; // Mang 2 chieu

implementation
{$R *.dfm}

procedure TfrmMang2C.btnNhaph2MaTranClick(Sender: TObject);
var i, j: byte;
begin
  randomize; // thu tuc tron so ngau nhien
  if tryStrToInt(edt_m.Text,m) and (m>0)
    and tryStrToInt(edt_n.Text,n) and (n>0) then
  begin
    for i:=0 to m-1 do // m hang
      for j:=0 to n-1 do // n cot
        begin
          //----Nhap vao mang a va hien thi vao StringGrid A--
          a[i,j]:=random(500); // ham tra ve 1 so ngau nhien
          stgA.Cells[j,i] := IntToStr(a[i,j]);
        end;
    //----Nhap vao mang b va hien thi vao StringGrid B--
    b[i,j]:=random(500); // ma tran B
    stgB.Cells[j,i] := IntToStr(b[i,j]);
  end;
  stgA.RowCount :=m; stgA.ColCount :=n;
  stgB.RowCount :=m; stgB.ColCount :=n;
  stgC.RowCount :=m; stgC.ColCount :=n;
  stgSX.RowCount :=m; stgSX.ColCount :=n;

```

```

        btnTong.Enabled := true;
    end
    else ShowMessage('Ban nhap gia tri m hoac n KHONG hop le.');
end;

procedure TfrmMang2C.FormCreate(Sender: TObject);
begin
    stgA.FixedRows:=0; stgA.FixedCols:=0; stgA.DefaultColWidth:=32;
    stgB.FixedRows:=0; stgB.FixedCols:=0; stgB.DefaultColWidth:=32;
    stgC.FixedRows:=0; stgC.FixedCols:=0; stgC.DefaultColWidth:=32;
    stgSX.FixedRows:=0;stgSX.FixedCols:=0;stgSX.DefaultColWidth:=32;
    btnTong.Enabled:=False;
    btnSX.Enabled:=False;
end;

procedure TfrmMang2C.btnTongClick(Sender: TObject);
var i, j: byte;
begin
    for i:= 0 to m-1 do
        for j:= 0 to n-1 do
            begin
                c[i,j]:=a[i,j] + b[i,j];
                stgC.Cells[j,i]:=IntToStr(c[i,j]); //hien thi vao StringGridC
            end;
    btnSX.Enabled := True;
end;

procedure TfrmMang2C.btnExitClick(Sender: TObject);
begin
    Close;
end;

procedure TfrmMang2C.btnSXClick(Sender: TObject);
var i, j, k: byte;
    temp: integer;
begin
    for i:=0 to m-1 do      // SX theo tung hang i
        for j:=0 to n-2 do
            for k := j+1 to n-1 do
                if c[i,j]<c[i,k] then //giam dan
                    begin
                        temp:=c[i,j];
                        c[i,j]:=c[i,k];
                        c[i,k]:=temp;
                    end;
end;

```

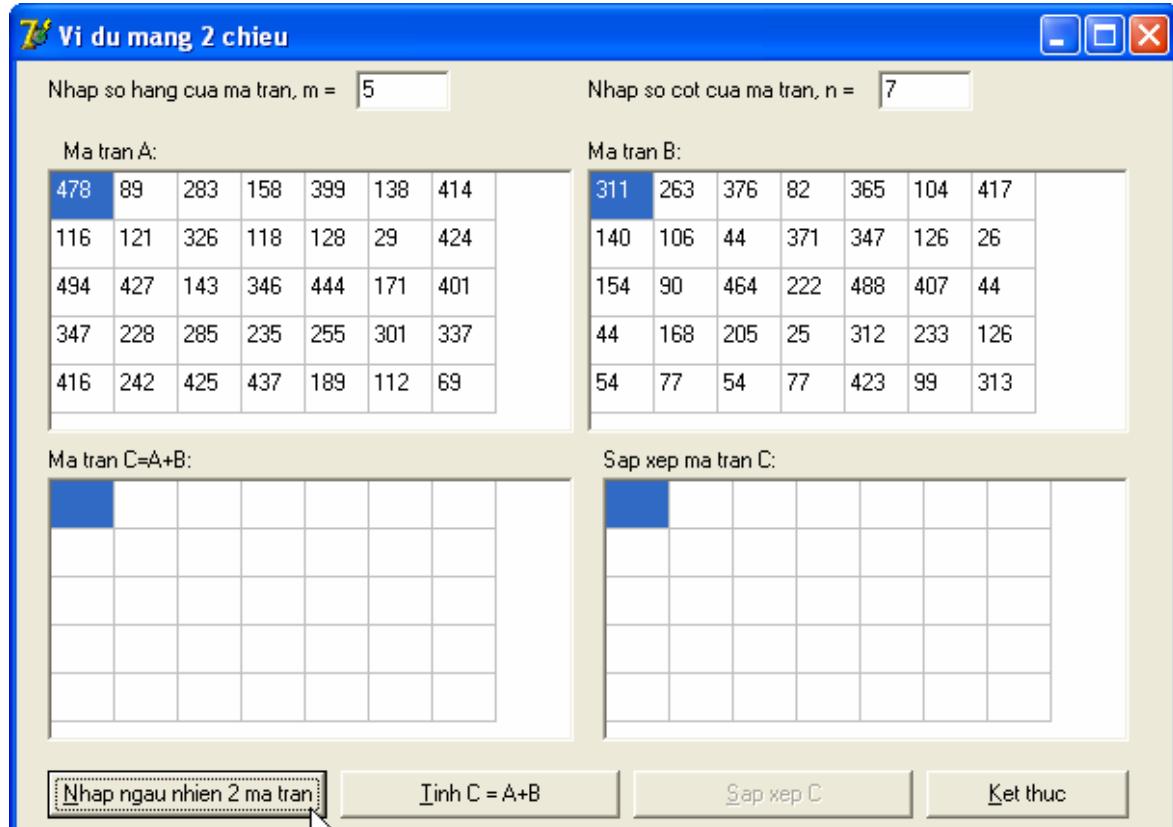
Chương 9: Kiểu Mảng

```
//hien thi mang tran C da SX vao StringGrid SX
  for i:=0 to m-1 do
    for j:=0 to n-1 do
      stgSX.Cells[j,i] := IntToStr(c[i,j]);
end;

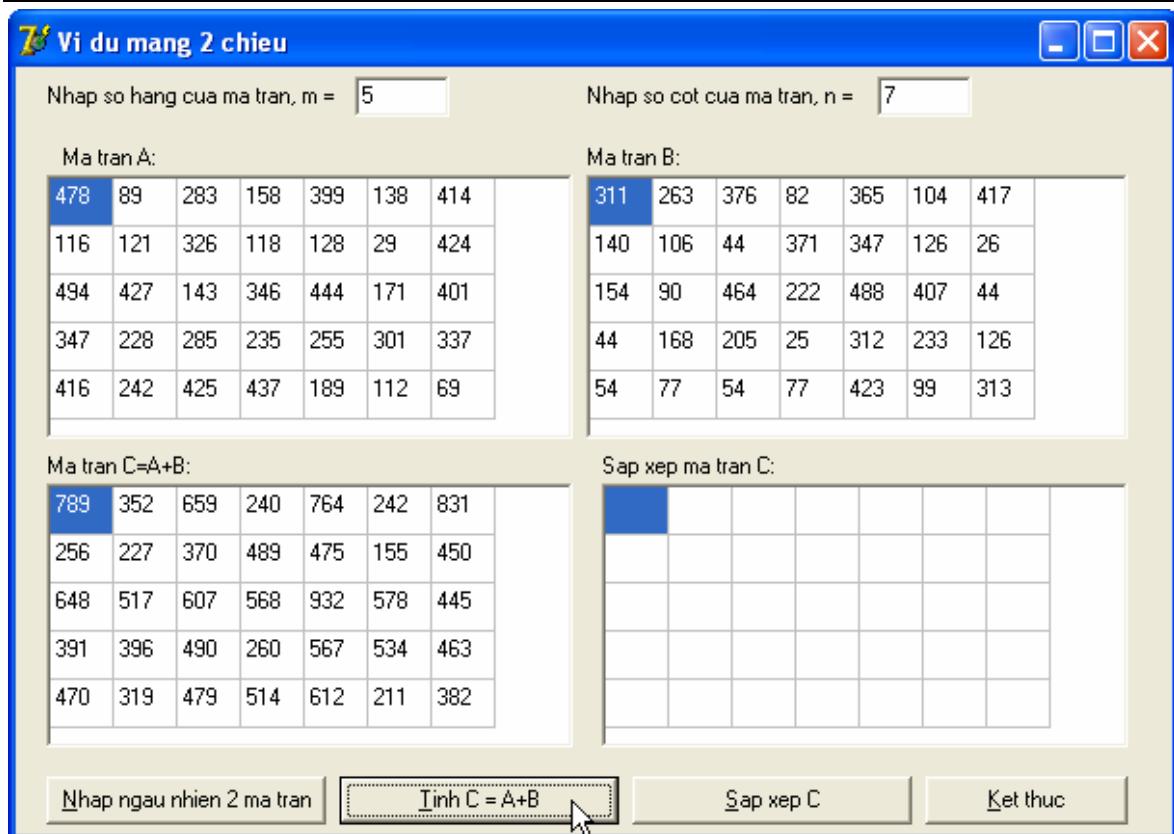
end.
```

- Lưu dự án vào thư mục S:\ViduMang2C

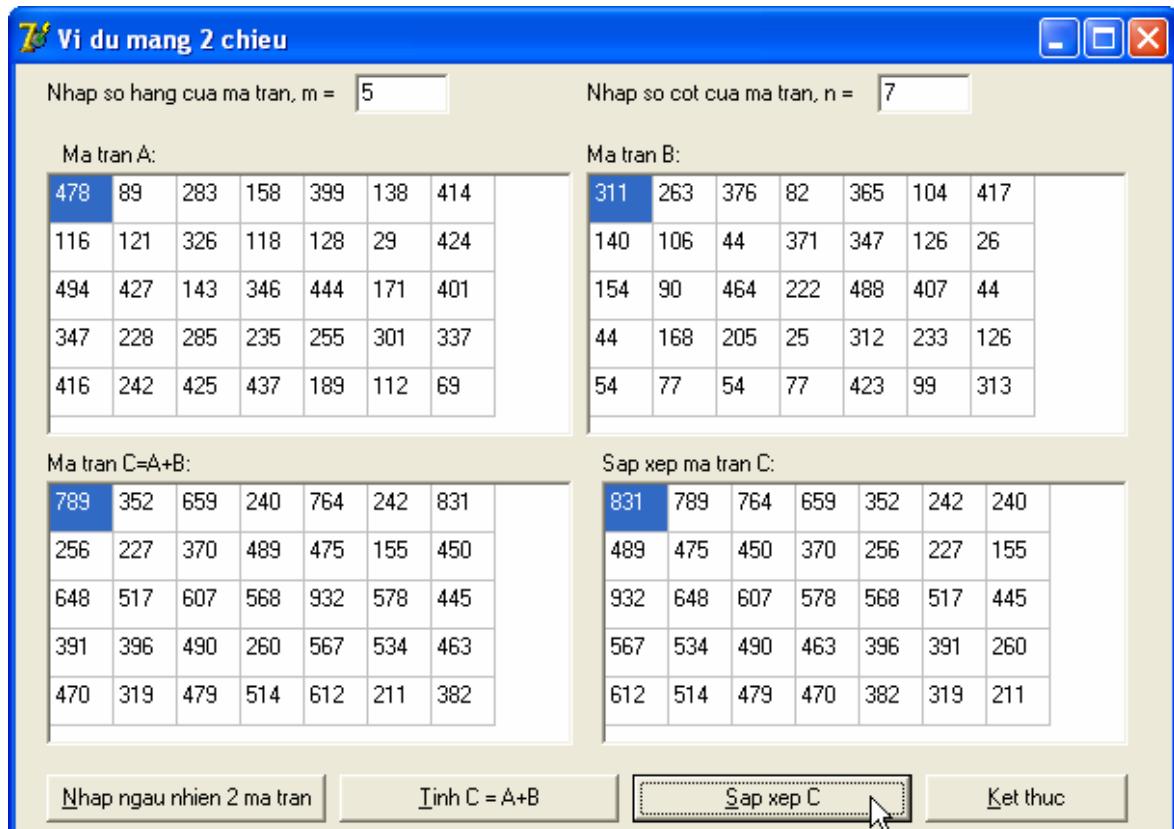
- Biên dịch và chạy chương trình, ta có kết quả như sau:



Hình 12: Nhập ngẫu nhiên 2 ma trận 5 hàng 7 cột



Hình 13: Tổng C = A + B



Hình 14: Sắp xếp mảng C giảm dần theo từng hàng.

III. Mảng động (Dynamic array)

Như đã trình bày ở trên, mảng tĩnh là mảng số phần tử phải được khai báo trước, vì vậy khi ta không sử dụng các phần tử của mảng thì mảng vẫn được cấp phát và chiếm bộ nhớ. Tuy nhiên với mảng động chúng ta có thể khai báo và cấp phát mảng một cách linh động tùy theo nhu cầu sử dụng. Mảng động không chiếm bộ nhớ cố định, ta có thể dùng thủ tục **SetLength** để thay đổi kích thước mảng lúc thực thi.

III.1. Mảng động một chiều

III.1.1. Khai báo gián tiếp

Cú pháp khai báo mảng động một chiều:

```
type
    Kiểu_mảng = array of kiểu_phần_tử;
var
    Danh_sách_biến : Kiểu_mảng;
```

III.1.2. Khai báo trực tiếp

```
var
    Danh_sách_biến : array of kiểu_phần_tử;
```

Để truy xuất đến mỗi phần tử của mảng, chúng ta cũng thực hiện tương tự như truy xuất ở phần mảng tĩnh.

Ví dụ 8: Khai báo một mảng động có tên là DaySo chứa các phần tử số thực.

```
var DaySo : array of extended;
```

III.1.3. Cấp phát và giải phóng vùng nhớ trong mảng động

Trong ví dụ trên, số phần tử của mảng vẫn chưa được xác định. Để sử dụng mảng ta phải khai báo số phần tử mảng bằng thủ tục **SetLength** như sau:

```
SetLength(DaySo,10);
```

Câu lệnh SetLength này sẽ cấp phát 10 phần tử trong bộ nhớ, có chỉ số từ **0..9** (Chú ý: mảng động luôn được cấp phát với chỉ số bắt đầu bằng 0).

Để giải phóng vùng nhớ đã cấp phát cho mảng, chúng ta chỉ cần gán trị **nil** cho biến mảng như trong ví dụ sau:

```
DaySo:=nil; // giải phóng vùng nhớ cho mảng DaySo vừa được cấp phát ở trên.
```

Chú ý: Ta có thể thực hiện phép gán hai biến cùng kiểu mảng cho nhau. Ví dụ X, Y là hai biến cùng kiểu mảng thì lệnh gán X:=Y là hợp lệ. Tuy nhiên lúc này nếu ta thay đổi các giá trị của mảng X thì mảng Y cũng thay đổi theo vì thực chất X chỉ tham chiếu đến Y (điều này khác với mảng tĩnh).

III.1.4. Một số hàm và thủ tục cơ bản được sử dụng để lấy thông tin của mảng động

- Thủ tục SetLength(var S; NewLength: integer);

SetLength cấp phát cho mảng S số phần tử là NewLength.

- Hàm Copy(S; Index, Count: integer): array;

Hàm copy trả về một mảng con có Count phần tử, với phần tử đầu tiên có chỉ số là Index. Ví dụ: khai báo một mảng 100 phần tử.

```
var a: array[0..100] of integer;
```

```
a:= copy(a,0,20); // mảng A còn 20 phần tử
```

- Hàm Low(x); trả về chỉ số thấp nhất của mảng x.

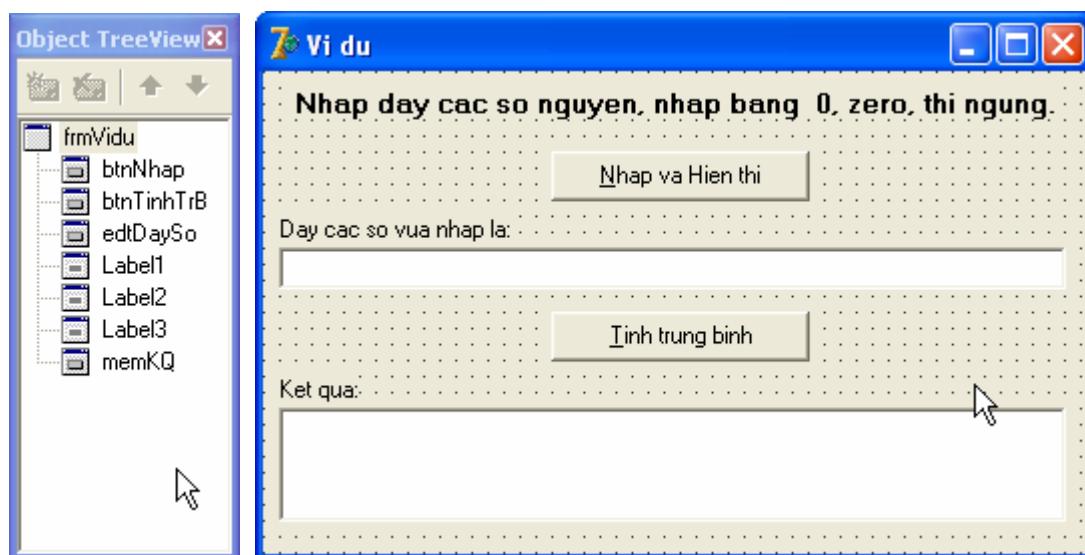
- Hàm High(x); trả về chỉ số cao nhất của mảng x. Nếu mảng rỗng thì hàm sẽ trả về -1.

- Hàm Length(x): integer; trả về số phần tử của mảng x.

Ví dụ 9: Viết chương trình nhập vào một dãy các số thông qua hàm InputBox cho đến khi nhập bằng 0 thì ngừng. Trong quá trình nhập CÓ kiểm tra lỗi nhập liệu. Hiển thị các số vừa nhập lên màn hình. Cho biết trung bình của các số dương (lớn hơn 0) và trung bình của các số âm (nhỏ hơn 0) vừa nhập.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 15: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình:

```

unit untMangDong;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmVidu = class(TForm)
    btnNhaph: TButton;
    Label3: TLabel;
    edtDaySo: TEdit;
    Label1: TLabel;
    btnTinhTrB: TButton;
    Label2: TLabel;
    memKQ: TMemo;
    procedure btnNhaphClick(Sender: TObject);
    procedure btnTinhTrBClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmVidu: TfrmVidu;
  a: array of single;
  spt:integer; //So phan tu ma ban da nhap cho mang.

implementation
{$R *.dfm}

procedure TfrmVidu.btnNhaphClick(Sender: TObject);
var i: integer; so:single; ds:string; kt:Boolean;
begin
  i:=0;
  SetLength(a,1);
  ds := '';
  repeat
    kt := TryStrToFloat(InputBox('Hop nhap','Nhap phan tu thu ' +
      IntToStr(i)+ ' :',''),so);
    if kt = false then
      ShowMessage('Gia tri nhap KHONG phai la kieu so. Nhap lai!');
    else
      begin
        SetLength(a,i+1);

```

```

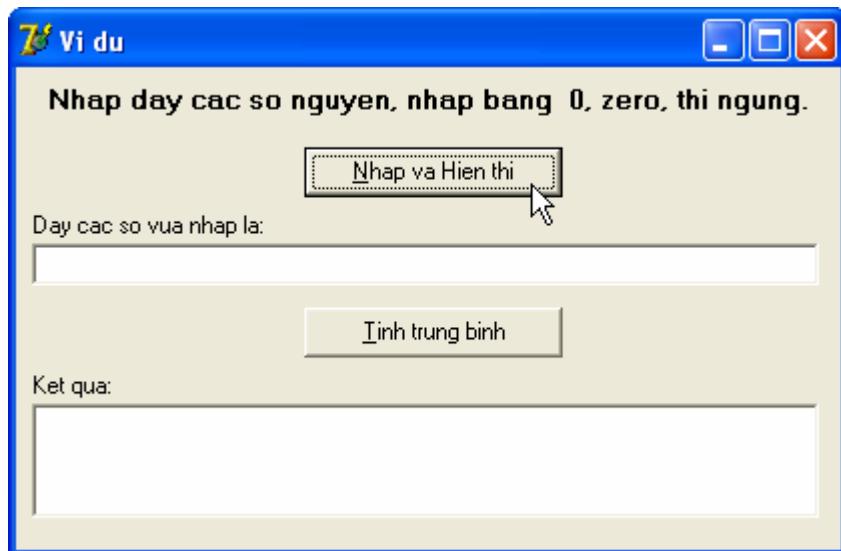
a[i] := so;
ds := ds + #32 + FloatToStr(so); {ghep cac so vua nhap
thanh mot chuoit.}
inc(i);
end;
until (so = 0);
edtDaySo.Text := ds; //Hien thi day so vua nhap ra ListBox
spt := i;
end;

procedure TfrmVidu.btnTinhTrBClick(Sender: TObject);
var mTongAm, mTongDuong : single; i, m, n: integer;
begin
mTongAm := 0; mTongDuong := 0;
m := 0; n := 0;
for i := 0 to spt-1 do // spt: so phan tu cua mang a
begin
if a[i] <0 then
begin
mTongAm := mTongAm + a[i];
inc(m);
end;
if a[i] >0 then
begin
mTongDuong := mTongDuong + a[i];
inc(n);
end;
end;
end;

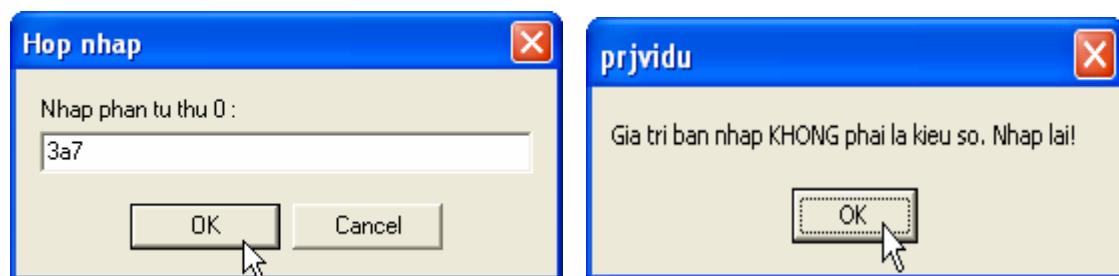
memKq.Lines.Clear;
if m>0 then memKq.Lines.Add('* Trung binh cac so am = '+
FloatToStr(mTongAm/m))
else memKq.Lines.Add('* Trung binh cac so am = 0');
if n>0 then memKq.Lines.Add('* Trung binh cac so duong = '+
FloatToStr(mTongDuong/n))
else memKq.Lines.Add('* Trung binh cac so duong = 0');
end;
end.

```

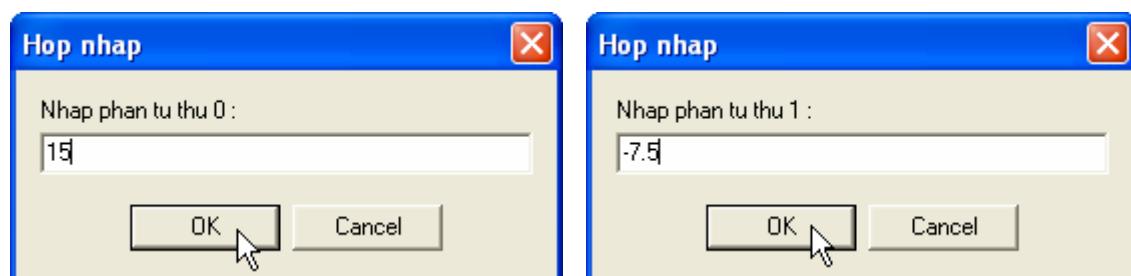
- Lưu dự án vào thư mục: **S:\MangDong1C**
- Biên dịch và chạy chương trình, ta có kết quả như sau:



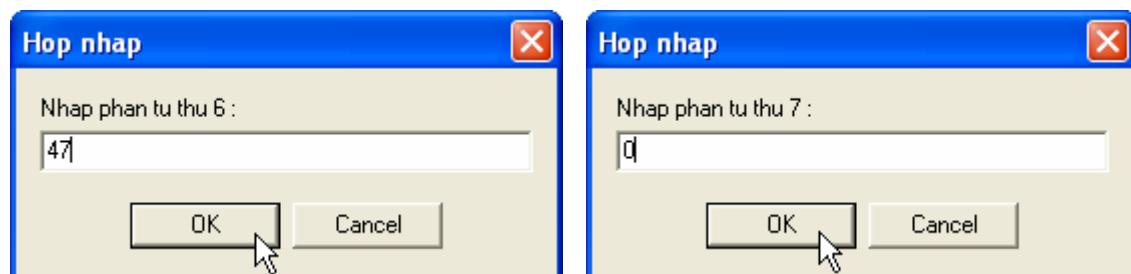
Hình 16: Bắt đầu chạy



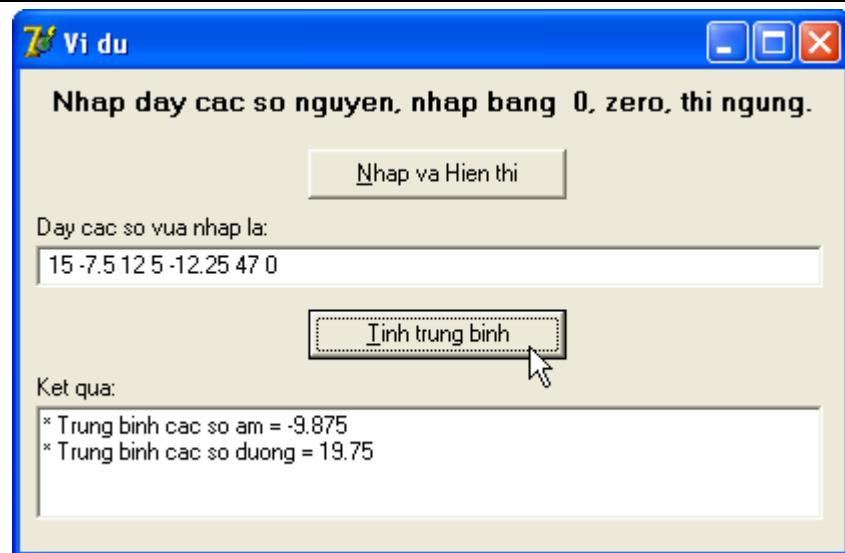
Hình 17: Trường hợp nhập dữ liệu sai, phải nhập lại.



Hình 18: Nhập phần tử thứ 0 là 15 và phần tử thứ 1 là -7.5



Hình 19: Nhập phần tử thứ 6 là 47 và **gõ 0 để kết thúc** nhập liệu.

**Hình 20:** Kết quả của chương trình

III.2. Mảng động nhiều chiều

Tương tự như mảng tĩnh, mảng động nhiều chiều cũng chính là mảng của mảng. Chúng ta có thể dùng từ khóa **array of** để khai báo mảng với nhiều cấp.

III.2.1. Khai báo gián tiếp

```

type
    Kiểu_mảng = array of array of ... array of kiểu_phần_tử;
var
    Danh_sách_biến : Kiểu_mảng;
  
```

Ví dụ 10: Khai báo gián tiếp hai mảng động 2 chiều và 3 chiều:

```

type
    Mang2C = array of array of single;           // mảng động 2 chiều
    Mang3C = array of array of array of integer; // mảng động 3 chiều

var
    a: Mang2C; b: Mang3C;
  
```

III.2.2. Khai trực tiếp

```

var
    Danh_sách_biến: array of array of .. array of kiểu_phần_tử;
  
```

Ví dụ 11: Khai báo trực tiếp hai mảng động 2 chiều và 3 chiều:

```

var
    a: array of array of single;
    b: array of array of array of integer;
  
```

Khi muốn sử dụng biến mảng động nhiều chiều thì ta cũng phải cấp phát vùng nhớ giống như mảng động một chiều.

Ví dụ 12: Cấp phát vùng nhớ cho biến DaySo đã khai báo ở trên

SetLength(DaySo,10,5); // DaySo có kích thước [0..10, 0..5]

Tuy nhiên, chúng ta có thể cấp phát theo kiểu như sau:

SetLength(DaySo,5); // cấp phát 5 phần tử một chiều

SetLength(DaySo[2],5); // mở rộng phần tử thứ ba thành mảng 2 chiều

Kết quả cấp phát của ví dụ bên trên:

0						
1						
2	0	1	2	3	4	5
3						
4						

III.3. Hằng mảng

Để khai báo hằng mảng, chúng ta chỉ cần đặt các giá trị hằng tương ứng với từng phần tử của mảng trong dấu ngoặc đơn.

Ví dụ 13: Khai báo một số hằng mảng

Const So: **array[0..9] of integer** = (0,10,2,35,40,45,20,7,78,9); //hằng mảng số

Const Kys: **array[0..7] of char** = ('0','1','2','3','4','5','6','7'); //hằng mảng ký số

Để định nghĩa hằng mảng nhiều chiều, ta sử dụng dấu ngoặc đơn và dấu phẩy để phân cách các chiều của mảng.

Ví dụ 14: Hằng mảng hai chiều

Type

Mang2C = **array[0..1, 0..1] of Integer;**

Const

MaTran: Mang2C = ((2, 3),(4, 5));

Ý nghĩa: với khai báo trên ta đã tạo ra một mảng hai chiều có tên là MaTran có các giá trị như sau:

MaTran[0,0] = 2

MaTran[0,1] = 3

MaTran[1,0] = 4

MaTran[1,1] = 5

CHƯƠNG 10: KIỂU CHUỖI KÝ TỰ

Chuỗi là kiểu dữ liệu đại diện cho một dãy tuân tự các ký tự. Các ký tự trong chuỗi thì được đánh chỉ số bắt đầu là 1 và tiếp tục cho đến hết độ dài của nó. Một chuỗi không chứa ký tự nào cả được gọi là chuỗi rỗng (null/ empty string) và độ dài của nó sẽ bằng không (zero). Một đặc tính hay của chuỗi là độ dài thực của nó sẽ tự động thay đổi khi chạy chương trình và như vậy sẽ thuận lợi cho người viết chương trình **không** phải tính toán để cập nhật lại sự thay đổi độ dài thực của nó.

I. Các loại chuỗi ký tự trong Object Pascal

Trong Delphi, chuỗi được phân ra làm 3 loại: ShortString, AnsiString và WideString

I.1. Chuỗi ngắn (ShortString)

Là kiểu dữ liệu có khả năng chứa tối đa là 255 ký tự được đánh chỉ số từ 1 đến 255. Riêng phần tử thứ 0 để chứa **độ dài thực** của chuỗi. Với kiểu dữ liệu ngắn này, thì mỗi ký tự chiếm 1 byte bộ nhớ. Trong ngôn ngữ lập trình Delphi, kiểu ShortString được dùng để hỗ trợ kiểu String trong ngôn ngữ lập trình Pascal, hay nói cách khác là giúp người lập trình có thể duy trì mã nguồn của ngôn ngữ Pascal trên phiên bản Delphi.

Để khai báo biến thuộc kiểu dữ liệu chuỗi ngắn này, bạn có thể khai báo thông qua 2 cách như sau:

I.1.1. Sử dụng tên chuẩn ShortString

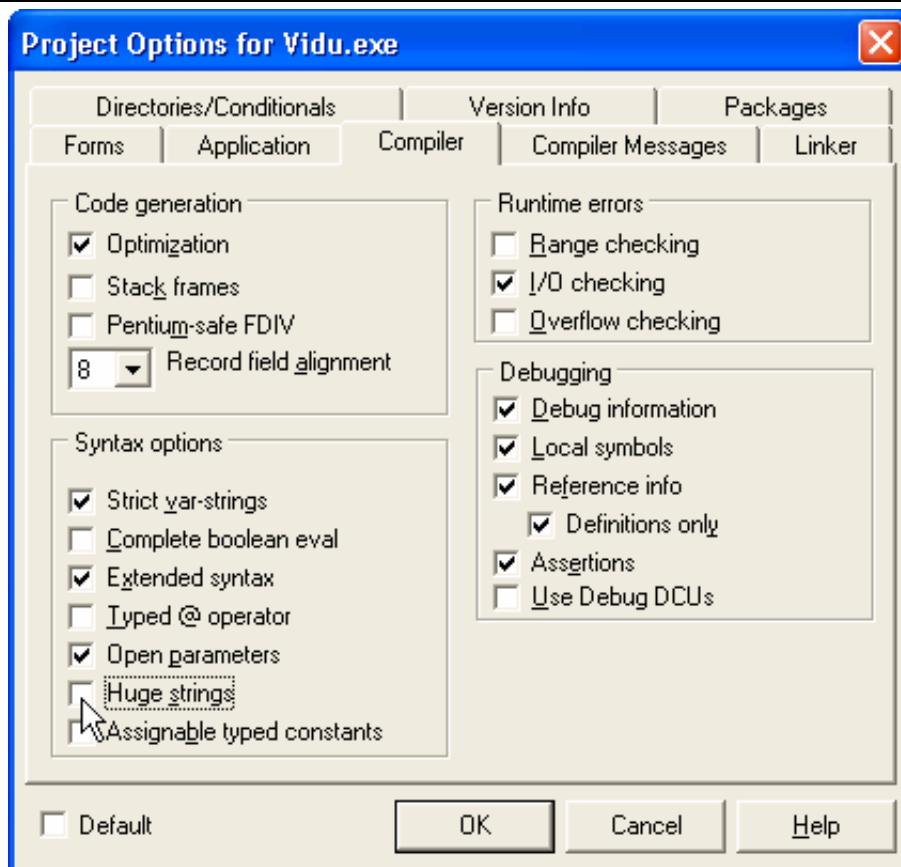
Với cách khai báo dữ liệu theo cách này, bạn muốn khai báo chuỗi có độ dài tối đa là 255 ký tự và bạn không thể giới hạn lại **độ dài khai báo** của nó bằng cách đặt một hàng chỉ độ dài trong cặp dấu mốc vuông.

Ví dụ 1:

```
var diachi: ShortString;           // đúng cú pháp
hoten: ShortString[30];           // lỗi cú pháp
```

I.1.2. Sử dụng từ khóa string

Với cách khai báo này, bạn phải sử dụng thêm chỉ thị **không** sử dụng các loại chuỗi lớn (huge strings – sẽ được đề cập sau) là {\$H-} hoặc bỏ chọn mục Huge strings trong hộp thoại “Project Options” thông qua chức năng Project/Options/Compiler như sau:

**Hình 1:** Tắt chỉ thị Huge strings

Cách sử dụng này có thể khắc phục được khuyết điểm của ShortString là bạn có thể giới hạn lại **độ dài khai báo** của nó.

Ví dụ 2:

```
{$H-} // String chuyển thành ShortString
var    diachi: string;        // đúng cú pháp, độ dài khai báo là 255
       hoten: string[30];      // đúng cú pháp, độ dài khai báo là 30
```

Trong chương trình, bạn có thể đặt lại độ dài thực cho nó bằng thủ tục SetLength mà bạn sẽ được trình bày ở phần dưới.

I.2. Chuỗi dài 1 byte (AnsiString)

Đây là loại dữ liệu chuỗi được sử dụng phổ biến nhất trong 3 loại, và mặc nhiên (default) Delphi đã bật sẵn chỉ thị {\$H+} hay chức năng “Huge strings” hộp thoại Project Options.

AnsiString có khả năng chứa tới khoảng 2,147,483,648 ($\sim 2^{31}$) ký tự được đánh chỉ số bắt đầu từ 1. Mỗi ký tự của AnsiString chiếm 1 byte bộ nhớ. Khác với ShortString, phần tử thứ 0 không được sử dụng để xác định độ dài thực của chuỗi. Độ dài thực này sẽ được xác định bởi hàm Length.

Để khai báo biến thuộc kiểu dữ liệu chuỗi AnsiString này, bạn có thể khai báo thông qua 2 tên chuẩn như sau:

I.2.1. Sử dụng tên chuẩn AnsiString

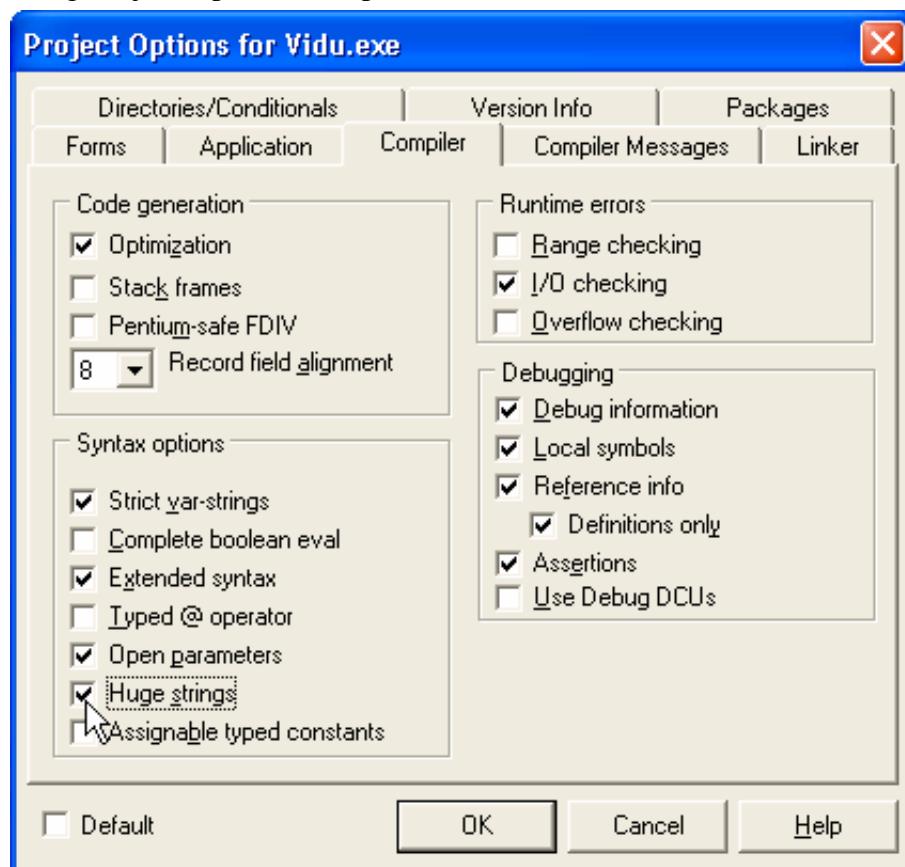
Với cách khai báo dữ liệu theo cách này, bạn muốn khai báo chuỗi có độ dài của nó tối đa và bạn không thể giới hạn lại độ dài khai báo của nó bằng cách đặt một hằng chỉ độ dài trong cặp dấu móc vuông.

Ví dụ 3:

```
var DiaChi: AnsiString;           // đúng cú pháp
                                // lỗi cú pháp
      HoTen: AnsiString[30];
```

I.2.2. Sử dụng tên chuẩn String

Với cách khai báo này, bạn phải sử dụng thêm chỉ thị **có** sử dụng các loại chuỗi lớn (huge strings) là {\$H+} hoặc chọn mục Huge strings trong hộp thoại “Project Options” thông qua chức năng Project/Options/Compiler như sau:



Hình 2: Bật chỉ thị Huge strings

Ví dụ 4:

```
{$H+}                           // bật chỉ thị sử dụng AnsiString
var GhiChu: String;           // đúng cú pháp, được hiểu là AnsiString
```

Cũng như kiểu ShortString, bạn có thể điều chỉnh lại độ dài thực cho nó bằng thủ tục SetLength trong thân chương trình.

Một tính năng hữu dụng của kiểu AnsiString trong Delphi là nó đã được cài đặt sẵn chức năng **dếm-tham chiếu** (Reference-counting) và **sao chép-ghi** (Copy-on-write).

I.3. Chuỗi dài 2 byte (WideString)

Có tính chất tương tự như AnsiString, nhưng điểm khác biệt là mỗi ký tự trong WideString chiếm 2 byte bộ nhớ (ký tự Unicode), ít được sử dụng hơn, và không có tính chất đếm-tham chiếu (Reference-counted) và sao chép-ghi (Copy-on-write) như kiểu AnsiString.

WideString có khả năng chứa tới khoảng $1,073,741,824(\sim 2^{30})$ ký tự được đánh chỉ số bắt đầu từ 1. Cũng tương tự như kiểu AnsiString, độ dài thực của chuỗi kiểu WideString cũng có thể điều chỉnh lại được bởi thủ tục SetLength và xác định độ dài hàm Length.

Để khai báo biến thuộc kiểu dữ liệu chuỗi WideString, bạn có thể khai tên chuẩn là **WideString** như sau:

```
var m_ChuoI2byte: WideString;
```

II. Các thao tác trên chuỗi

Trong các hàm và thủ tục để xử lý chuỗi ký tự, thì hầu hết các hàm được sử dụng cho cả 3 loại ShortString, AnsiString và WideString. Tuy nhiên, giáo trình sẽ tập trung chủ yếu vào các thao tác trên kiểu chuỗi dài 1 byte – AnsiString.

Lưu ý: Dưới đây là các đoạn chương trình nhỏ, mục đích nhằm minh họa rõ hơn các hàm, thủ tục được trình bày. Các bạn có thể đặt các đoạn chương trình này vào trong thủ tục sự kiện của nút lệnh (TButton) để thử nghiệm chương trình.

II.1. Phép toán cộng chuỗi

Ví dụ 5: Sử dụng toán tử cộng

```
var s:AnsiString;
begin
    s := 'Le' + #32 + 'Loi';
    ShowMessage('s = ' + s);
end;
```



Hình 3: Cộng 2 chuỗi

II.2. Phép toán so sánh

Các so sánh gồm có so sánh bằng, lớn hơn, lớn hơn hoặc bằng, khác, nhỏ hơn, nhỏ hơn hoặc bằng tương ứng với các toán tử so sánh $=, >, \geq, <, \leq$

Khi so sánh 2 chuỗi ký tự thì các ký tự được so sánh theo từng cặp một theo chiều từ trái sang phải dựa vào giá trị trong bảng mã ASCII. Có 2 khả năng xảy ra khi so sánh: Nếu chuỗi thứ nhất là một phần của chuỗi thứ hai kể từ đầu chuỗi, thì chuỗi thứ hai sẽ lớn hơn chuỗi thứ nhất nếu chiều dài của chuỗi thứ hai lớn hơn, như ví dụ ở hình 4, hoặc bằng nếu chuỗi thứ hai có độ dài bằng chuỗi thứ nhất, như ví dụ ở hình 5.

Ghi chú: Chuỗi rỗng (null string, viết là "") là chuỗi không có chứa gì cả. Giá trị mã ASCII của nó là zero, nhỏ nhất trong các ký tự trong bảng mã ASCII.

Vì vậy: 'A' > " và chr(32) > "

Ví dụ 6: Chương trình so sánh 2 chuỗi bất kỳ nhập từ bàn phím:

```
var s1, s2: AnsiString;
```

```
begin
```

```
  s1:= InputBox('Nhập chuỗi','s1= ','Que huong la chum khe ngot');
```

```
  s2 := InputBox('Nhập chuỗi','s2= ','Que huong neu ai khong nho');
```

```
  if s1=s2 then ShowMessage(s1+ ' = ' + s2)
```

```
  else
```

```
    if s1>s2 then ShowMessage(s1+ ' > ' + s2)
```

```
    else ShowMessage(s1+ ' < ' + s2);
```

```
end;
```

Khi chạy chương trình:

+ Nếu nhập s1 = 'Vietnamese' , s2 = 'Vietnam' ta có kết quả như sau:



Hình 4: Chuỗi s1 > s2

+ Nếu nhập s1 = 'Me hien yeu dau' , s2 = 'Me hien yeu dau' ta có kết quả như sau:



Hình 5: Chuỗi s1 = S2

+ Với giá trị nhập như giá trị mặc định, ta có kết quả như sau:



Hình 6: Chuỗi s1 < s2

II.3. Các thủ tục và hàm chuẩn xử lý chuỗi ký tự

Trong Delphi, các bạn cần chú ý các thủ tục và hàm chuẩn trên từng loại chuỗi.

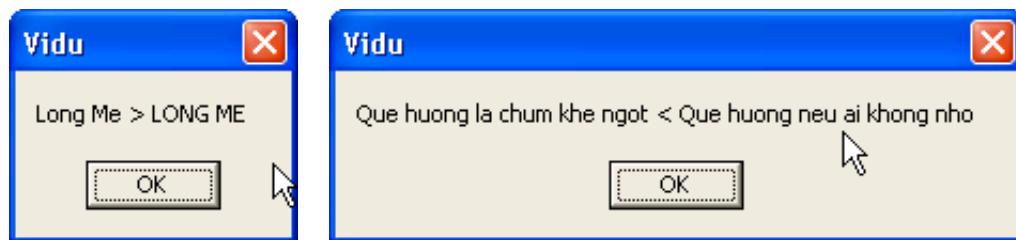
II.3.1. Hàm so sánh chuỗi

Cú pháp: `function CompareStr(const S1, S2: string): Integer;`

Ý nghĩa: So sánh có phân biệt ký tự hoa thường của chuỗi S1 với chuỗi S2. Kết quả trả về của hàm bằng: không nếu S1 bằng S2, một giá trị nhỏ hơn không nếu S1 < S2, và một giá trị lớn hơn không nếu S1>S2.

Ví dụ 7: Chương trình sử dụng hàm CompareStr

```
var s1,s2: String; m_ret: integer;
begin
  s1 := InputBox('Nhập chuỗi','s1= ','Long Me');
  s2 := InputBox('Nhập chuỗi','s2= ','LONG ME');
  m_ret := CompareStr(s1,s2);
  if m_ret = 0 then ShowMessage('Hai chuỗi bằng nhau')
  else
    if m_ret > 0 then ShowMessage(s1+ '>' + s2)
    else ShowMessage(s1+ '<' + s2);
end;
```



Hình 7: Phân biệt hoa thường

Cú pháp: `function CompareText(const S1, S2: string): Integer;`

Ý nghĩa: Như hàm so sánh CompareStr ở trên, nhưng **không** phân biệt ký tự hoa thường.



Hình 8: Không phân biệt hoa thường

II.3.2. Thủ tục xóa chuỗi

Cú pháp: procedure Delete(var S: string; Index, Count:Integer);

Ý nghĩa: Xóa khỏi chuỗi S một số ký tự là Count bắt đầu từ vị trí Index tính từ trái sang.

Ví dụ 8: Minh họa thủ tục xóa chuỗi

```
var s: string;
begin
  s := 'Troj xanh xanh';
  ShowMessage('Chuoi ban dau: ' + s);
  Delete(s,6,4);
  ShowMessage('Chuoi sau khi xoa: ' + s);
end;
```

Khi chạy chương trình, ta sẽ thấy trên màn hình:



Hình 9: Kết quả ví dụ xóa chuỗi

II.3.3. Thủ tục chèn chuỗi

Cú pháp: procedure Insert(Source: string; var S: string; Index: Integer);

Ý nghĩa: Chèn chuỗi Source vào chuỗi S tại chỉ số Index.

Ví dụ 9: Minh họa thủ tục chèn chuỗi

```
var s: string;
begin
  s := 'Mat troi con';
  ShowMessage('Chuoi ban dau: ' + s);
  Insert(' be',s,9);
  ShowMessage('Chuoi sau chen: ' + s);
end;
```



Hình 10: Kết quả ví dụ chèn chuỗi

II.3.4. Thủ tục đổi số thành chuỗi

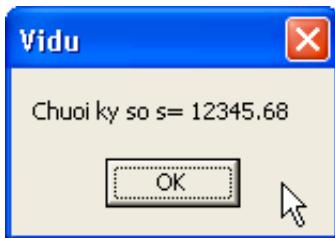
Cú pháp: procedure Str(X [: Width [: Decimals]]; var S);

Ý nghĩa: Đổi giá trị số S thành chuỗi rồi gán cho St, Giá trị Width:Decimals nếu có sẽ là số vị trí và số chữ số thập phân của S.

Ví dụ 10: Minh họa thủ tục đổi số thành chuỗi

```
var s: string; num:extended;
begin
  num:=12345.679;
  Str(num:5:2,s);
  ShowMessage('Chuoi ky so s = ' + s);
end;
```

Kết quả trên màn hình:



Hình 11: Kết quả ví dụ đổi sang chuỗi

II.3.5. Thủ tục đổi chuỗi thành số

Cú pháp: procedure Val(S; var V; var Code: Integer);

Ý nghĩa: Đổi chuỗi ký số S thành giá trị kiểu số và gán giá trị này cho V. Code là số nguyên dùng để phát hiện lỗi: nếu đổi đúng thì Code có giá trị = 0, nếu sai do S không biểu diễn đúng số nguyên hoặc số thực thì Code sẽ nhận giá trị bằng vị trí của ký tự sai trong chuỗi ký số S.

Ví dụ 11: Minh họa thủ tục đổi chuỗi thành số

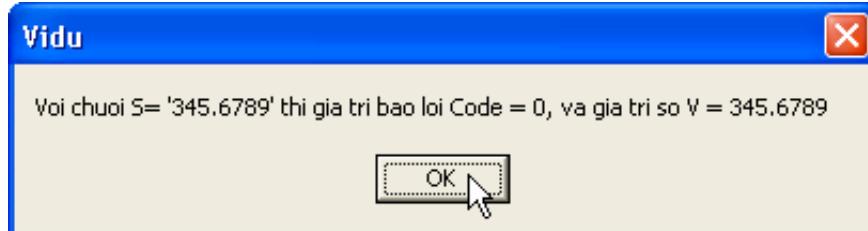
```
var S: string; V: Real; Code: integer;
begin
  S := InputBox('Nhập chuỗi','Chuỗi ký số hợp lệ s: ','345.6789');
  Val(S,V,Code);
  ShowMessage('Với chuỗi S= "' + S + '" thì giá trị bao lỗi Code = ' +
    IntToStr(Code) + ', và giá trị số V = '+FloatToStr(V));
  S := InputBox('Nhập chuỗi','Chuỗi ký số KHÔNG hợp lệ s: ','345A.6789');
```

```

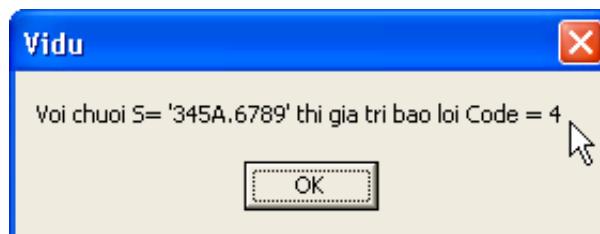
Val(S,V,Code);
ShowMessage('Voi chuoi S= "'+ S +'" thi gia tri bao loi Code = '+ IntToStr(Code));
end;

```

Khi chạy với các giá trị nhập là mặc định, bạn sẽ có những kết quả sau:



Hình 12: Đổi được từ chuỗi sang số



Hình 13: Đổi **không** được từ chuỗi sang số

II.3.6. Hàm xác định độ dài của chuỗi ký tự

Cú pháp: function Length(S): Integer;

Ý nghĩa: Cho kết quả là một số nguyên chỉ độ dài của chuỗi ký tự St.

Ví dụ: Minh họa hàm Length

var S: String;

begin

S := 'Vet chan tron tren cat';

ShowMessage('Do dai cua chuoi "' + s + '" la: ' + IntToStr(Length(S)));

end;



Hình 14: Kết quả hàm Length

II.3.7. Hàm sao chép chuỗi

Cú pháp: function Copy(S; Index, Count: Integer): string;

Ý nghĩa: Kết quả trả về là một chuỗi con của chuỗi S, bắt đầu tại chỉ số Index và chép Count ký tự.

Ví dụ 12: Sao chép chuỗi ‘Hát với chú ve con’ từ vị trí thứ 9 và lấy 6 ký tự

```
var S, Substr: String;
begin
  S := 'Hat voi chu ve con';
  Substr:=Copy(S,9,6);
  ShowMessage('Chuoi con Substr vua duoc copy la: '+Substr);
end;
```



Hình 15: Kết quả hàm Copy

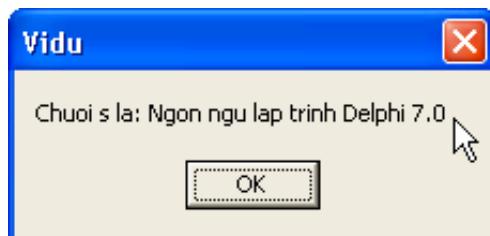
II.3.8. Hàm ghép nối chuỗi (Concatenate)

Cú pháp: function Concat(s1 [, s2,..., sn]: string): string;

Ý nghĩa: Cho kết quả là một chuỗi mới được ghép lần lượt bởi các chuỗi s1, s2,...,sn. Hàm này giống như phép cộng các chuỗi.

Ví dụ 13: Ghép nối chuỗi

```
var s:AnsiString;
begin
  s := Concat('Ngon ngu lap trinh',Chr(32),'Delphi',#32,'7.0');
  ShowMessage('Chuoi s la: '+ s);
end;
```



Hình 16: Kết quả hàm Concat

II.3.9. Hàm xác định chỉ số của chuỗi

Cú pháp: function Pos(Substr: string; S: string): Integer;

Ý nghĩa: Cho kết quả là vị trí đầu tiên của chuỗi Substr trong chuỗi S. Nếu không tìm thấy thì hàm sẽ trả về giá trị 0.

Ví dụ 14: Tìm kiếm chuỗi ‘ve con’ và chuỗi ‘mèo con’ trong chuỗi ‘Hát với chú ve con’:

```

var S, Substr: String;
      m_Index: integer;
begin
  Substr := 've con';
  S := 'Hat voi chu ve con';
  m_Index := Pos(Substr,S);
  if m_Index=0 then
    ShowMessage('Chuoi "'+ SubStr +"' khong tim thay trong chuoi "'+ S +'")"
  else
    ShowMessage('Chuoi "'+ SubStr +"' bat dau tai chi so '+ IntToStr(m_Index)+"
                 ' cua chuoi "'+ S +'");
  Substr := 'meo con';
  m_Index := Pos(Substr,S);
  if m_Index=0 then
    ShowMessage('Chuoi "'+ SubStr +"' khong tim thay trong chuoi "'+ S +'")"
  else
    ShowMessage('Chuoi "'+ SubStr +"' bat dau tai chi so '+ IntToStr(m_Index) +
                 ' cua chuoi "'+ S +'");
end;

```

Khi chạy chương trình ta có 2 kết quả hiển thị lần lượt như sau:



Hình 17: Tìm thấy vị trí, m_Index =13



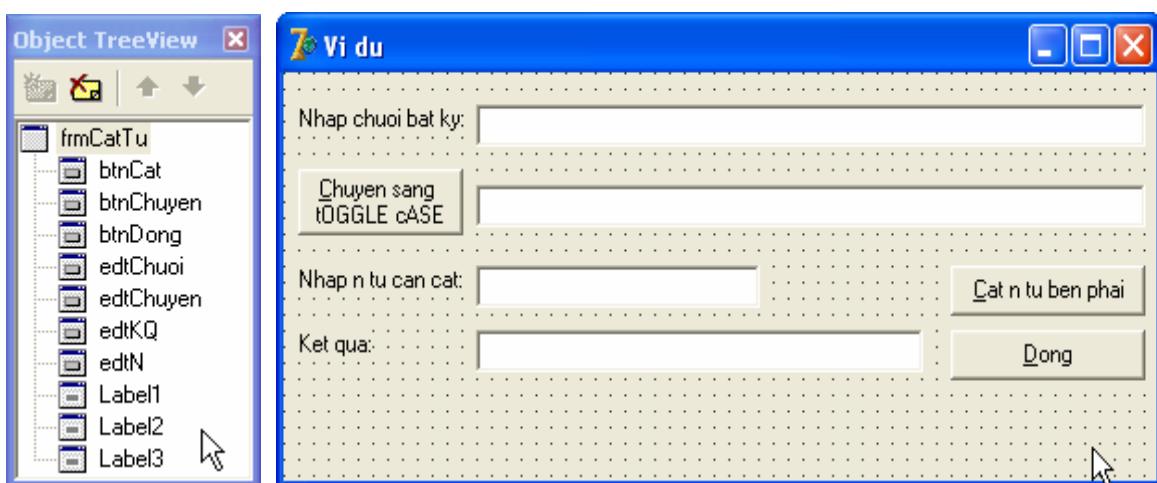
Hình 18: Không tìm thấy, m_Index = 0

* Sau đây là các ví dụ minh họa về chương chuỗi.

Ví dụ 15: Thiết kế form và viết code để giải bài toán: Nhập vào một chuỗi ký tự bất kỳ từ bàn phím. Chuyển đổi chuỗi vừa nhập sang dạng **TOGGLE cASE**. Nhập vào giá trị n, hãy cắt n **từ cuối** của s, nếu n nhập không thỏa mãn so với s, thông báo lỗi: n không thỏa.

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 19: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình:

```
unit untCatPhai;           //Tên unit file
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TfrmCatTu = class(TForm)
    edtChuoi: TEdit;
    Label1: TLabel;
    btnCat: TButton;
    edtN: TEdit;
    Label2: TLabel;
    edtKQ: TEdit;
    Label3: TLabel;
    btnDong: TButton;
    btnChuyen: TButton;
  end;
```

```

edtChuyen: TEdit;
procedure btnCatClick(Sender: TObject);
procedure btnDongClick(Sender: TObject);
procedure btnChuyenClick(Sender: TObject);
private
  { Private declarations }
public
  { Public declarations }
end;

var
  frmCatTu: TfrmCatTu;

implementation
{$R *.dfm}

procedure TfrmCatTu.btnCatClick(Sender: TObject);
var i, j, n, dem, sotu :integer;
  s: string;
begin
  s := edtChuyen.Text;
  sotu := 0;
  for i:=1 to length(s) do
    if s[i]=#32 then inc(sotu);
  sotu := sotu + 1;

  if TryStrToInt(edtN.Text,n) and (n>=1) and (n<=sotu) then
    begin
      dem:=0;
      j:=length(s); //Bat dau tu ben phai...
      while (dem < n) and (j>=1) do
        begin
          if s[j]=#32 then inc(dem);
          dec(j); //...va tro nguoc ben trai
        end;
      edtKq.Text := Copy(s,j+2,length(s)-j);
    end
  else
    edtKq.Text := 'Gia tri n nhap vao khong thoa bai toan.';
end;

procedure TfrmCatTu.btnDongClick(Sender: TObject);
begin
  Close;
end;

```

```

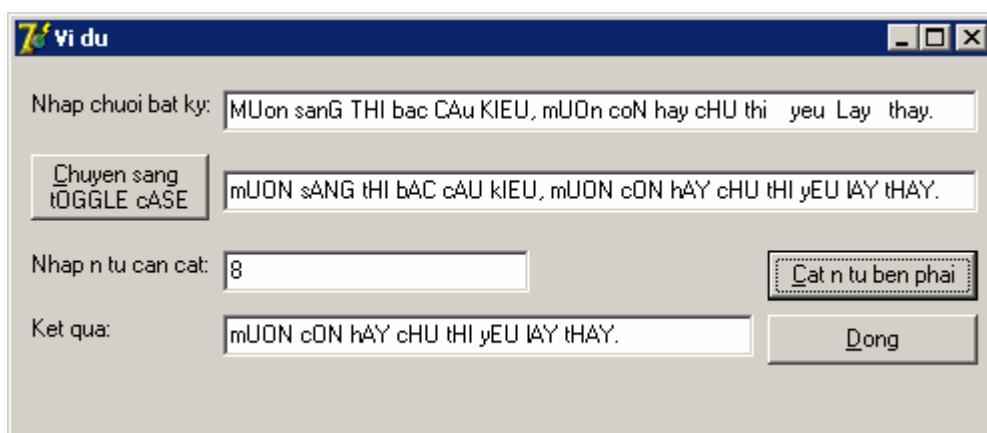
procedure TfrmCatTu.btnChuyenClick(Sender: TObject);
var s: string;
    i: integer;
begin
    s := edtChuoi.Text;
    s := Trim(s);
    while pos(#32#32,s)>0 do delete(s,pos(#32#32,s),1);
    s :=UpperCase(s);

    s[1] := Chr(ord(s[1])+ 32); //đổi ký tự thứ nhất sang thường
    for i:=2 to length(s)-1 do
        if s[i]=#32 then s[i+1] := Chr(ord(s[i+1])+ 32);
    edtChuyen.Text := s;
end;

end.

```

- Lưu dự án vào thư mục: S:\ViduCatPhai
- Biên dịch và chạy chương trình, ta có kết quả như sau:

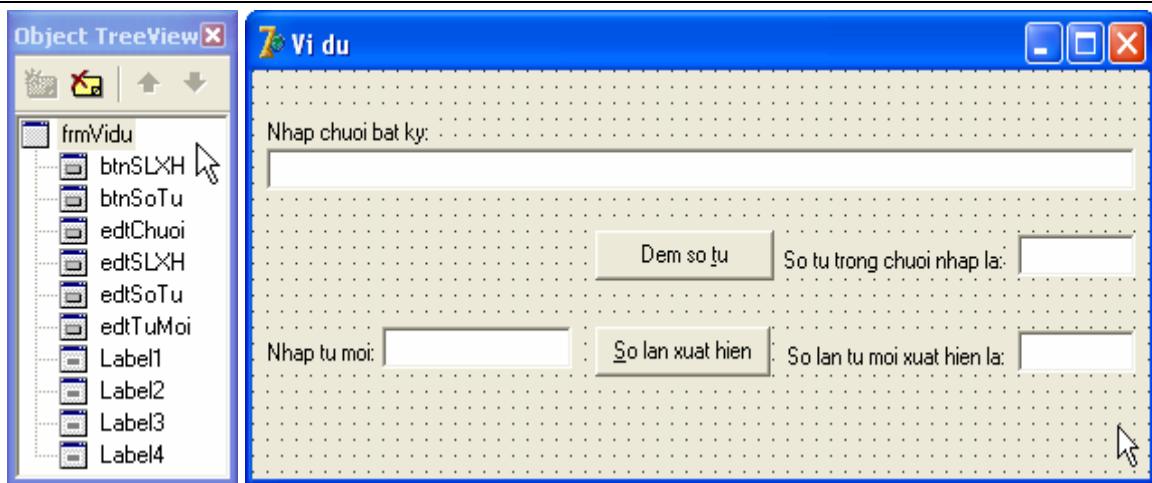


Hình 20: Kết quả chương trình

Ví dụ 16: Thiết kế form và viết code để giải bài toán: Nhập vào một chuỗi s. Cho biết chuỗi có bao nhiêu từ. Nhập vào một từ mới, xét xem từ mới này xuất hiện bao nhiêu lần trong chuỗi s?

Hướng dẫn:

- Tạo dự án mới: **File/New/Application**
- Đặt thuộc tính name cho các đối tượng của chương trình như cửa sổ Object TreeView, và thiết kế form như hình sau:



Hình 21: Tên các đối tượng và giao diện chương trình

- Đoạn code của chương trình:

```

unit untDemTu;

interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TfrmVidu = class(TForm)
    edtChuoi: TEdit;
    Label1: TLabel;
    btnSLXH: TButton;
    edtTuMoi: TEdit;
    Label2: TLabel;
    edtSLXH: TEdit;
    Label3: TLabel;
    btnSoTu: TButton;
    edtSoTu: TEdit;
    Label4: TLabel;
    procedure btnSLXHClick(Sender: TObject);
    procedure btnSoTuClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  frmVidu: TfrmVidu;
  s: string; // biến chung.

implementation

```

```

{$R *.dfm}

procedure TfrmVidu.btnSLXHClick(Sender: TObject);
var slxh :integer;
    tumoi, s2, tu :string;
begin
    s2 := s; //Lay chuoi da cat bo khang trang thua.
    s2 := s2 + #32; //them phan tu cam canh.
    tumoi := trim(edtTuMoi.Text);
    slxh := 0;
    while s2 <>'' do //s <> rỗng
begin
    tu := copy(s2,1, pos(#32,s2)-1);
    if tumoi = tu then inc(slxh);
    delete(s2,1, pos(#32,s2));
end;
    edtSlxh.Text := IntToStr(slxh);
end;

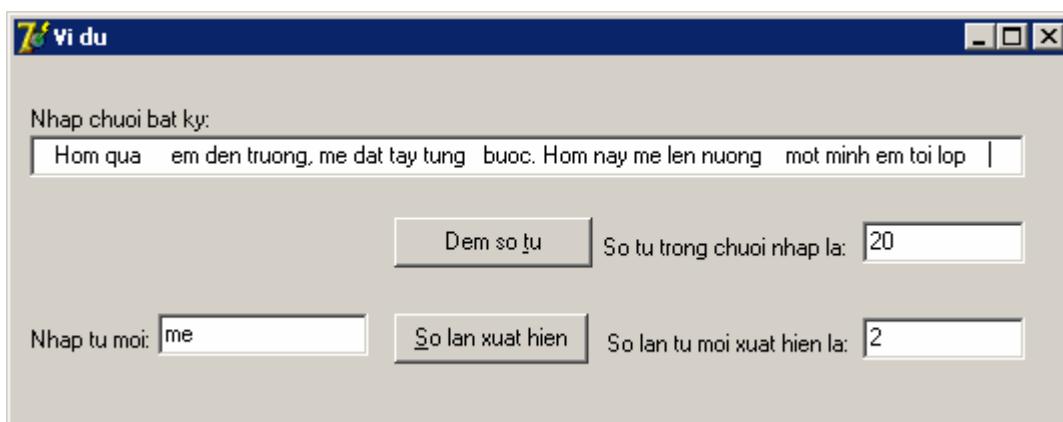
procedure TfrmVidu.btnSoTuClick(Sender: TObject);
var i, sotu: integer;
begin
    if edtChuoi.Text = '' then ShowMessage('Ban da nhap chuoi rong.
                                              Hay nhap lai!')
    else
begin
    s := trim(edtChuoi.Text);
    while pos(#32#32,s)>0 do delete(s,pos(#32#32,s),1);
    sotu := 0;
    for i:=1 to length(s) do
        if s[i]=#32 then inc(sotu);
    edtSoTu.Text := IntToStr(sotu + 1);
end;
    btnSLXH.Enabled := True;
end;

procedure TfrmVidu.FormCreate(Sender: TObject);
begin
    btnSLXH.Enabled := False;
end;

```

- Lưu dự án vào thư mục: S:\ViduDemTu
- Biên dịch và chạy chương trình, ta có kết quả như sau:

- Trường hợp từ mới có xuất hiện 2 lần trong chuỗi:



Hình 22: Từ “mẹ” có xuất hiện trong chuỗi nhập

- Trường hợp từ mới không xuất hiện lần nào trong chuỗi:



Hình 23: Từ “nhà” không xuất hiện trong chuỗi nhập

PHỤ LỤC

Phần này giới thiệu một số thành phần thông dụng và chương trình hiển thị ảnh đơn giản – Simple view picture:

1. Thẻ System



Thành phần TTimer: Đồng hồ bấm giờ. Biểu tượng: 

Thuộc tính	Ý nghĩa
Name	Để đặt tên đối tượng
Enable	Nếu có giá trị true (mặc nhiên) thì tự động bật sự kiện bấm giờ - OnTimer. Còn False thì không.
Interval	Khoảng thời gian (milliseconds) để kích hoạt sự kiện OnTimer

Sự kiện OnTimer: Sự kiện được kích hoạt khi hết khoảng thời gian xác định trong Interval

2. Thẻ Additional



Thành phần TImage: Hiển thị ảnh. Biểu tượng: 

Thuộc tính	Ý nghĩa
Name	Để đặt tên đối tượng
Picture	Xác định hình được nạp (load) vào để hiển thị

Và một số sự kiện quen thuộc như: OnClick, OnContextPopup, OnChange...

3. Thẻ Win 3.1



a. TListBox: Hộp danh sách liệt kê các tập tin. Biểu tượng: 

Thuộc tính	Ý nghĩa
Name	Để đặt tên đối tượng
Mask	Mặt nạ lọc tập tin được hiển thị trong nó như: *.jpg;*.gif;*.bmp
Cursor	Các kiểu con trỏ mouse khi bạn trỏ đến đối tượng. Mặc nhiên là crDefault.

Và một số sự kiện quen thuộc như: OnClick, OnContextPopup, OnChange...

b. TFilterComboBox: Hộp liệt kê thả lọc kiểu tập tin. Biểu tượng: 

Thuộc tính	Ý nghĩa
FileList	Để lọc các kiểu tập tin được hiển thị trong Hộp danh sách liệt kê các tập tin. Bạn Click chọn tên đối tượng FileListBox (flbPics) để gắn kết vào.
Cursor	Các kiểu con trỏ mouse khi bạn trỏ đến đối tượng. Mặc nhiên là crDefault.
Filter	Các kiểu tập tin để lọc. Bạn sử dụng các ký tự đại diện để nhóm lại một số loại tập tin như: *.doc, *.dpr, *.jpg,... Bạn Click vào biểu tượng  để soạn thảo các kiểu lọc thông qua cửa sổ "Filter Editor".

Và một số sự kiện quen thuộc như: OnClick, OnContextPopup, OnChange...

c. TDIRECTORYLISTBOX: Hộp danh sách liệt kê các thư mục. Biểu tượng: 

Thuộc tính	Ý nghĩa
FileList	Để hiển thị các tập tin trong Hộp danh sách liệt kê các tập tin. Bạn Click chọn tên đối tượng FileListBox (flbPics) để gắn kết vào.
TextCase	Hiển thị tên ổ đĩa là chữ in hoa (tcUpperCase) hay thường (tcLowerCase)
Cursor	Các kiểu con trỏ mouse khi bạn trỏ đến đối tượng. Mặc nhiên là crDefault.
DirLabel	Để gắn tên đối tượng nhãn dùng cho việc hiển thi đường dẫn (path) khi bạn chọn mở (DClick) thư mục làm việc .

Và một số sự kiện quen thuộc như: OnClick, OnContextPopup, OnChange...

d. TDriveComboBox: Hộp liệt kê thả các ổ đĩa. Biểu tượng: 

Thuộc tính	Ý nghĩa
DirList	Để hiển thị các thư mục vào trong hộp danh sách liệt kê các thư mục. Bạn Click chọn tên đối tượng DirectoryListBox (dlbPics) để gắn kết vào.
TextCase	Hiển thị tên ổ đĩa là chữ in hoa (tcUpperCase) hay thường (tcLowerCase)
Cursor	Các kiểu con trỏ mouse khi bạn trỏ đến đối tượng. Mặc nhiên là crDefault.

Và một số sự kiện quen thuộc như: OnClick, OnContextPopup, OnChange...

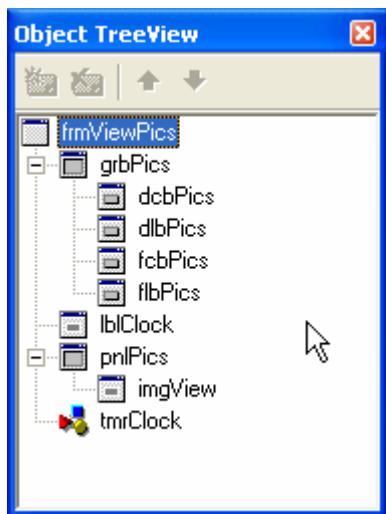
Phần này chỉ nhằm giới thiệu các thành phần cơ bản trong Windows. Trong Delphi còn có các thành phần khác ưu việt hơn đó là các thành phần trên thẻ Dialogs, thẻ Simples,... Bạn có thể dễ dàng viết lại chương trình này thông qua các thành phần trong 2 thẻ này.



Trong tương lai, Bộ môn Tin học Ứng Dụng - Khoa Khoa Học Tự Nhiên có thể sẽ viết thêm phần Delphi nâng cao để giúp các bạn khai thác sức mạnh của Delphi như lập trình về Cơ sở dữ liệu, lập trình Mạng, lập trình Web, Trò chơi,...

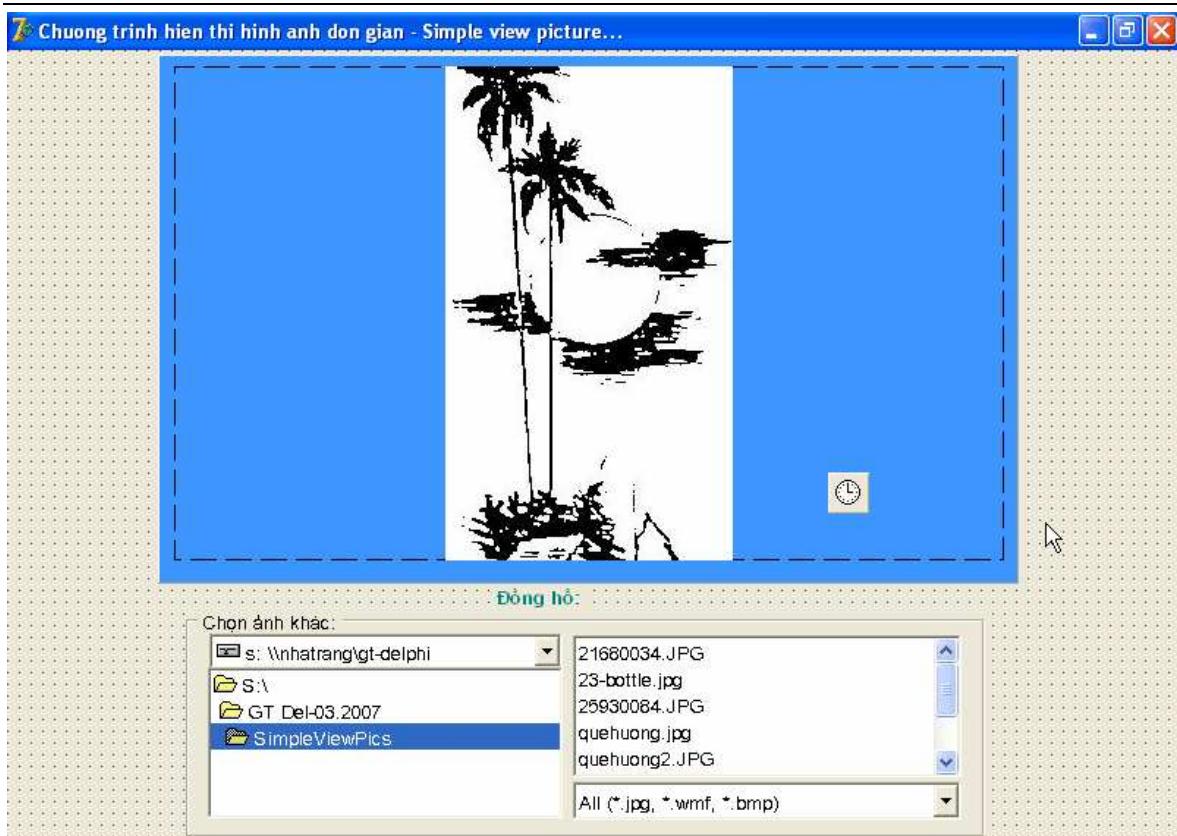
Chương trình hiển thị hình ảnh đơn giản - Simple view picture, để hiển thị các tập tin hình ảnh dạng jpg, .wmf, .bmp. Ngoài ra chương trình cũng tạo dòng chữ chạy (Marquee) trên thanh tiêu đề của form theo hướng từ phải sang trái theo hướng nhìn (trái sang phải theo hướng màn hình), và có hiển thị đồng hồ chạy.

Chương trình hiển thị hình ảnh đơn giản - Simple view picture:



frmViewPics	:	TForm
grbPics	:	TGroupBox
dcbPics	:	TDriveComboBox
dlbPics	:	TDirectoryListBox
flbPics	:	TFileListBox
fcbPics	:	TFilterComboBox
lblClock	:	TLabel
pnlPics	:	TPanel
imgView	:	TImage
tmrClock	:	TTimer

Tên của các đối tượng



Đoạn lệnh của chương trình như sau:

```
unit untViewPics;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, ExtCtrls, StdCtrls, jpeg, ExtDlgs,
  FileCtrl;
type
  TfrmViewPics = class(TForm)
    tmrClock: TTimer;
    pnlPics: TPanel;
    imgView: TImage;
    grbPics: TGroupBox;
    dcBpics: TDriveComboBox;
    fcBpics: TFilterComboBox;
    dlbPics: TDirectoryListBox;
    flbPics: TFileListBox;
    lblClock: TLabel;
    procedure tmrClockTimer(Sender: TObject);
    procedure flbPicsClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
```

```
{ Public declarations }
end;
var
  frmViewPics: TfrmViewPics;
implementation
{$R *.dfm}

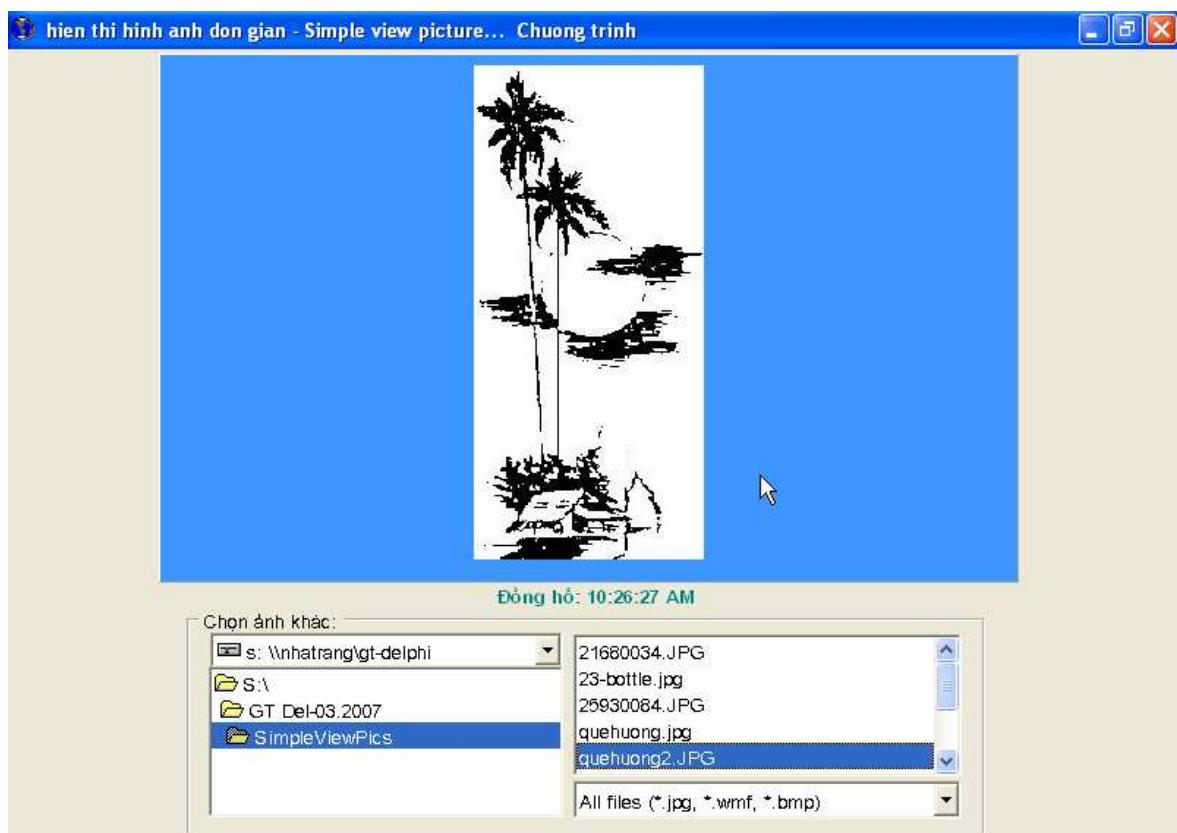
procedure TfrmViewPics.tmrClockTimer(Sender: TObject);
begin
  lblClock.Caption := 'Đồng hồ: ' + TimeToStr(Time);
  Caption:= Copy(Caption,2,length(Caption)-1)+Copy(Caption,1,1);
end;

procedure TfrmViewPics.flbPicsClick(Sender: TObject);
begin
try
  imgView.Picture.LoadFromFile(flbPics.FileName);
except
  on E: Exception do ShowMessage('Sorry! Tap tin anh nay bi SAI
kieu hoac KHONG duoc ho tro!... ')
end;
end;

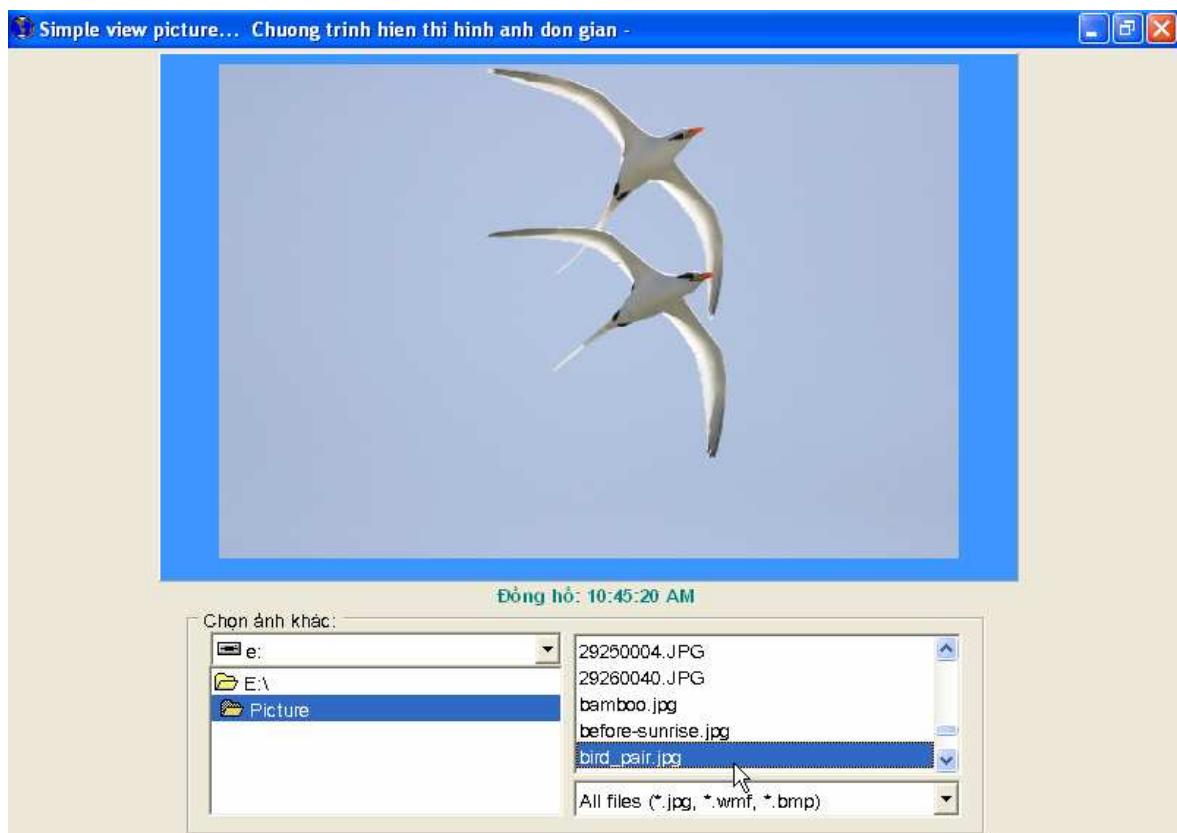
procedure TfrmViewPics.FormCreate(Sender: TObject);
begin
  dcBpics.DirList := dblPics;
  dblPics.FileList := flbPics;
  flbPics.Mask := '*.jpg;*.gif;*.bmp';
  fcbPics.FileList := flbPics;
  fcbPics.Filter := 'All (*.jpg, *.wmf, *.bmp)|
*.jpg;*.wmf;*.bmp|All files (*.*)|*.*';
  lblClock.Font.Color := clTeal;
  lblClock.Font.Name := 'VNarial';           {Delphi7 khong ho tro
                                              font Unicode. Con Delphi8 thi ok...}
  lblClock.Font.Size := 10;
  lblClock.Font.Style := [fsBold];
  imgView.Center := True;
  imgView.Proportional := True;
  imgView.Stretch := False;
  frmViewPics.Caption:='Chuong trinh hien thi hinh anh don gian -
Simple view picture... ';
  frmViewPics.WindowState := wsMaximized;
  pnlPics.Color := clGradientActiveCaption;
  tmrClock.Enabled := True;
  tmrClock.Interval := 150;
end;
end.
```

- Lưu vào trong thư mục S:\Simple view picture

- Biên dịch và chạy ta được hình ảnh như sau:



Chương trình Simple view picture.



TÀI LIỆU THAM KHẢO

1. Borland IDD - JM, MM, HM, CC, JG (2005), Object Pascal Language Guide.
2. <http://delphi.about.com>
3. John Barrow, Linda Miller, Katherine Malan, Helene Gelderblom (2005), Introducing Delphi Programming: Theory through Practice (Paperback), Oxford University Press, USA.
4. John Ayres (2002), The Tomes of Delphi – Win32 Core API Windows 2000 Edition, Wordware Publishing, Inc.
5. Julian Bucknall (2001), The Tomes of Delphi – Algorithms and Data Structures, Wordware Publishing, Inc.
6. Lê Phương Lan, Hoàng Đức Hải (2000), Giáo trình lý thuyết và bài tập Borland Delphi, NXB Giáo Dục, Hà Nội.
7. Marco Cantu (2003), Mastering Delphi 7, SyBex.

BẢN QUYỀN TÁC GIẢ

Tài liệu này được viết nhằm phục vụ cho việc học tập của sinh viên ở giai đoạn đầu làm quen và học cách lập trình sự kiện như thế nào thông qua ngôn ngữ lập trình Delphi, môi trường xây dựng ứng dụng tức thời RAD. Tác giả sẽ upload tài liệu này lên internet trong thời gian gần thông qua file: “Giao trình lập trình cơ bản B – Delphi 7.pdf”. Mọi thông tin cần trao đổi, xin liên hệ tác giả thông qua địa chỉ email: ydlinh@ctu.edu.vn.

Cần thơ, 19 November 2010, © Vũ Duy Linh.

