

MỤC LỤC

BÀI 1- GIỚI THIỆU C++ BUILDER	2
1. Ưu thế của C++ Builder so với Visual C++	2
2. Giới thiệu C++ Builder	2
3. Môi trường phát triển tích hợp IDE và các công cụ đặc trưng	6
BÀI 2- TỔNG QUAN VỀ VCL	15
1. Giới thiệu VCL.	15
2. Mô hình phân cấp VCL.	15
3. Các thuộc tính, biến cố chung với TControl	16
4. Các thuộc tính, biến cố chung với TWinControl	20
5. Các đối tượng, thành phần chuẩn của VCL	22
BÀI 3- LÀM VIỆC VỚI CÁC ĐIỀU KHIỂN	53
1. Thực hiện các hành động rê và thả	53
2. Thực hiện các hành động rê và kết dính	63
3. Làm việc với văn bản trong các điều khiển	64
4. Thêm đồ hoạ vào điều khiển	65
BÀI 4- TƯƠNG TÁC GIỮA NGƯỜI DÙNG VÀ ỨNG DỤNG	68
1. Sử dụng các hộp thoại	68
2. Hộp thoại Windows dùng chung	77
3. Tạo và quản lý các hộp thoại	78
4. Sử dụng đa biểu mẫu	87
5. Xây dựng chương trình TextEditor	103
BÀI 5- ĐỒ HOẠ VÀ MULTIMEDIA	112
1. Tổng quan về lập trình đồ hoạ	112
2. Các thành phần đồ hoạ VCL	112
3. Sử dụng các thuộc tính và các phương thức của đối tượng Canvas	112
4. Tạo và quản lý các bức vẽ	115
5. Lập trình multimedia.	141
BÀI 6- LIÊN KẾT CƠ SỞ DỮ LIỆU	152
1. Các kỹ thuật truy cập dữ liệu trong C++ Builder.	152
2. Các thành phần truy cập cơ sở dữ liệu của C++ Builder.	153
3. Các điều khiển nhận biết dữ liệu của C++ Builder	156
4. Các thành phần cơ sở dữ liệu cơ bản.	158
5. Xây dựng chương trình quản lý danh bạ	160
BÀI 7 – LẬP TRÌNH MẠNG	167
1. Các tính năng đặc biệt trong C++ Builder	167
2. Lập trình mạng với Socket	167
BÀI 8- XÂY DỰNG VÀ TRIỂN KHAI CÁC ỨNG DỤNG	171
1. Các kỹ thuật gỡ rối trong C++ Builder	171
2. Sử dụng tài nguyên Windows để xây dựng trợ giúp trực tuyến.	171
3. Sử dụng chương trình InstallShield Express.	174

BÀI 1- GIỚI THIỆU C++ BUILDER

1. Ưu thế của C++ Builder so với Visual C++

C++ Builder có nhiều ưu thế hơn so với Visual C++, sau đây chúng tôi liệt kê một số tính năng ưu việt:

- C++ Builder là công cụ phát triển phần mềm nhanh, cung cấp các thành phần trực quan được gói gọn mà tránh gọi các lệnh WinAPI như Visual C++. Tuy nhiên, C++ Builder vẫn hỗ trợ cho người lập trình khả năng gọi các hàm WinAPI.
- C++ Builder lập trình với giao diện đơn giản và trình sinh mã tự động mang lại hiệu quả cao mà không cần phải viết các đoạn lệnh để tạo cửa sổ từ đầu như Visual C++.
- C++ Builder có thể gói chương trình và các thành phần liên quan vào một tập tin duy nhất trong khi Visual C++ phải phụ thuộc vào MFCxx.DLL và các tập tin .DLL khác.
- C++ Builder cho phép chúng ta lập trình trên nền tảng chéo trong khi Visual C++ chỉ lập trình trên Windows.
- C++ Builder sử dụng các thành phần VCL trong khi Visual C++ sử dụng MFC. Thành phần VCL được bao gồm vào bộ cài đặt C++ Builder giúp người lập trình có thể tùy biến theo ý mình thích.
- C++ Builder cho phép phát triển các phần mềm trên nền tảng chéo (Linux và Windows) trong khi Visual C++ chỉ cho phép phát triển trên nền Windows.
- Visual C++ kết nối với cơ sở dữ liệu chỉ sử dụng công nghệ dựa trên nền tảng ADO trong khi C++ Builder còn có thể kết nối cơ sở dữ liệu có thể sử dụng BDE và SQL Links.

2. Giới thiệu C++ Builder

Borland C++ Builder là một trong những bộ biên dịch đầu tiên cho phép bạn sử dụng các đối tượng được gọi là những thành phần bằng cách sử dụng chuột và di chuyển xung quanh để thay đổi thiết kế trực quan của chương trình hơn là sử dụng các đoạn mã lệnh để thực hiện điều này. Cốt lõi của kỹ thuật này là lập trình thành phần, các thành phần có thể chạy với một tốc độ rất nhanh, thậm chí một đối tượng có thể hiện ra và ẩn đi mà mắt chúng ta chưa kịp nhìn thấy và nhấn chuột lên nó.

Lập trình với Borland C++ Builder dễ đến mức bất ngờ. C++ Builder tạo thêm sự sôi nổi cho ngôn ngữ lập trình C++. Đây là một công cụ tốt để chúng ta phát triển nhanh các chương trình một cách dễ dàng và tạo được các chương trình mạnh. C++ Builder cho phép sử dụng đầy đủ các thành phần cao cấp của C++, bao gồm mẫu (templates), không gian tên (name spaces), phép toán nạp chồng và toàn bộ hàm Windows API, kể cả các hàm dùng cho DirectX, OLE và ActiveX.

Ngôn ngữ C++ vẫn là cốt lõi của C++ Builder. C++ Builder đưa ra một cấp độ cao hơn để hỗ trợ ngôn ngữ C++. Borland C++ Builder tích hợp thư viện các thành phần trực quan (Visual Component Library) vào để dùng cho ngôn ngữ C++. Vấn đề nằm ở chỗ C++ Builder sử dụng ngôn ngữ Pascal, trong khi đó C++ Builder lại sử dụng ngôn ngữ C++. Tuy nhiên, dựa trên sự phê chuẩn của ANSI, Borland đã thêm các thành phần để hỗ trợ cho C++.

Trong ngôn ngữ lập trình hướng đối tượng, các biến thành viên dùng để lưu trữ các giá trị hoặc trạng thái của đối tượng trong suốt thời gian sống của nó. Tuy nhiên, những biến lại được truy cập trực tiếp, do đó, có thể một phần nào đó của chương trình sẽ làm gây chương trình vì đã đưa vào các giá trị không phù hợp.

Để giải quyết vấn đề trên, rất nhiều nhà phát triển ứng dụng hướng đối tượng tạo ra các hàm thành viên gọi là hàm lấy giá trị và cài đặt giá trị nhằm mục đích không cho phép truy cập trực tiếp các biến thành viên đồng thời cũng tạo ra các truy bắt ngoại lệ nhằm áp đặt giá trị luôn luôn phù hợp với biến lưu trữ. Phương pháp này còn cho phép lập trình viên có thể ẩn được các dạng biến dùng để lưu trữ dữ liệu chẳng hạn như tập tin, cơ sở dữ liệu, hoặc một kiểu dữ liệu phức hợp.

Borland tạo ra ngôn ngữ C++ Builder đã cung cấp **thuộc tính** nhằm thống nhất hai hàm thành viên lấy giá trị và cài đặt giá trị thành một thành viên trong lớp. Trong lớp của C++ Builder, một thuộc tính thường gồm một biến khai báo với bộ từ private và một hàm dùng để lấy giá trị và cài

đặt giá trị được khai báo với bộ từ protected, một khai báo thuộc tính với bộ từ public. Sau đây là một minh họa:

```
class aClassWithAProperty //khai báo lớp aClassWithAProperty
{
private:
    int myMemberVariable; //biến thành viên
protected:
    int __fastcall GetMemberVariable(void) //hàm lấy giá trị
    {
        return myMemberVariable;
    };
    void __fastcall SetMemberVariable(int theMemberVariable)
//hàm cài đặt giá trị
    {
        myMemberVariable = theMemberVariable;
    };
public:
    __property int MemberVariable = //thuộc tính
    {
        read=GetMemberVariable,
        write=SetMemberVariable
    };
};
```

Khi đó, chúng ta có thể sử dụng cài đặt của lớp để truy cập thuộc tính MemberVariable như sau:

```
AClassWithAProperty ClassWithAProperty;
ClassWithAProperty.MemberVariable = 2;
int Something = ClassWithAProperty.MemberVariable;
```

Các thuộc tính được sử dụng rất nhiều trong các thành phần C++ Builder. Để trợ giúp cho môi trường trực quan được cung cấp bởi C++ Builder, Borland đã tạo ra một phần đặc biệt cho lớp, được định nghĩa bởi __published tương tự như hai phần public và private ở ví dụ trên. Nếu một thuộc tính được đặt trong phần __published, nó sẽ được nhìn thấy trong cửa sổ điều chỉnh thuộc tính trong quá trình thiết kế.

Sau đây là minh họa về thuộc tính được đặt trong phần published:

```
class aClassWithAProperty: public TComponent
{
private:
    int myMemberVariable;
protected:
    int __fastcall GetMemberVariable(void)
    {
        return myMemberVariable;
    };
    void __fastcall SetMemberVariable(int theMemberVariable)
    {
        myMemberVariable = theMemberVariable;
    };
__published:
    __property int MemberVariable =
    {
        read=GetMemberVariable,
        write=SetMemberVariable
    };
};
```

Khi đó thuộc tính MemberVariable có thể được điều chỉnh trực tiếp trong cửa sổ điều chỉnh thuộc tính của Borland C++ Builder (Object inspector).

Như đã nói ở trên, C++ Builder thừa hưởng rất nhiều đặc tính của C++ Builder. C++ Builder cung cấp một thuộc tính mặc định. Với thuộc tính này, chúng ta có thể truy cập bằng cách dùng phép toán chỉ số. Chúng ta có thể truy cập thuộc tính mặc định này như sau:

```
(*StringListVariable)[Index]
```

hoàn toàn tương thích với cách truy cập sau vì Strings là thuộc tính mặc định:

```
StringListVariable->Strings[Index]
```

Ngôn ngữ C++ Builder giới thiệu một thành phần cho phép truy bắt ngoại lệ được chấp nhận bởi Java. Trong C++ Builder, chúng ta dùng `__finally` và `try` để truy bắt các ngoại lệ.

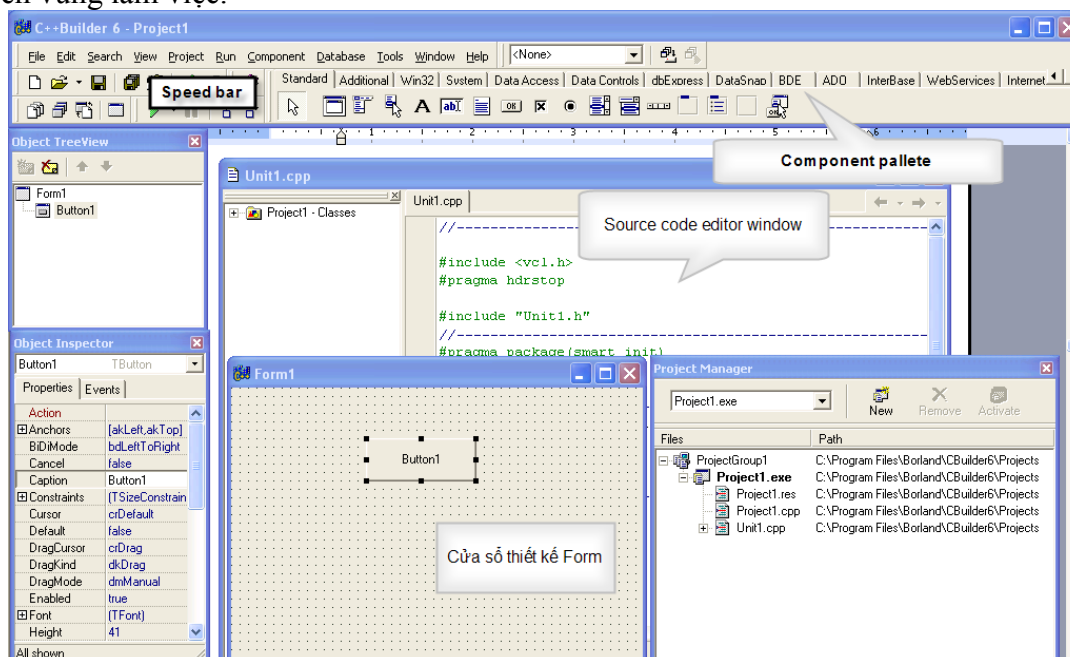
Ví dụ:

```
TStringList *List = new TStringList;
```

```
try
{
    // thực hiện các câu lệnh
}
__finally
{
    delete List;
```

Đoạn lệnh đặt trong `__finally` sẽ thực hiện khi các câu lệnh trong `try` đã được thực hiện xong hay có một ngoại lệ xảy ra.

Khi viết chương trình bằng Borland C++ Builder, chúng ta dùng thư viện thành phần trực quan (VCL – Visual Component Library). VCL là nguồn gốc cho các thành phần được sử dụng để tạo các ứng dụng C++ Builder (các thành phần tương tự VCL dùng để phát triển đa nền tảng gọi là CLX). Một thành phần là một đối tượng, thường trực quan, chẳng hạn một hộp đánh dấu (checkbox), một danh sách chọn ổ đĩa, hoặc một hình ảnh. Các thành phần cũng có thể không trực quan, chẳng hạn như các kết nối cơ sở dữ liệu hoặc các kết nối mạng. Các thành phần có thể được chọn từ bảng thành phần (component palette) của IDE bằng dùng chuột trái để chọn và đặt chúng lên vùng làm việc.



Hình 1 – IDE của Borland C++ Builder

Chúng ta cũng có thể thêm hoặc viết thêm cái thành phần của chúng ta, VCL thực sự đã giảm bớt mức độ khó cho quá trình làm việc của chúng ta.

Tất cả các thành phần đều có các thuộc tính, chúng ta có thể điều chỉnh bằng cửa sổ object inspector (xem hình 1)

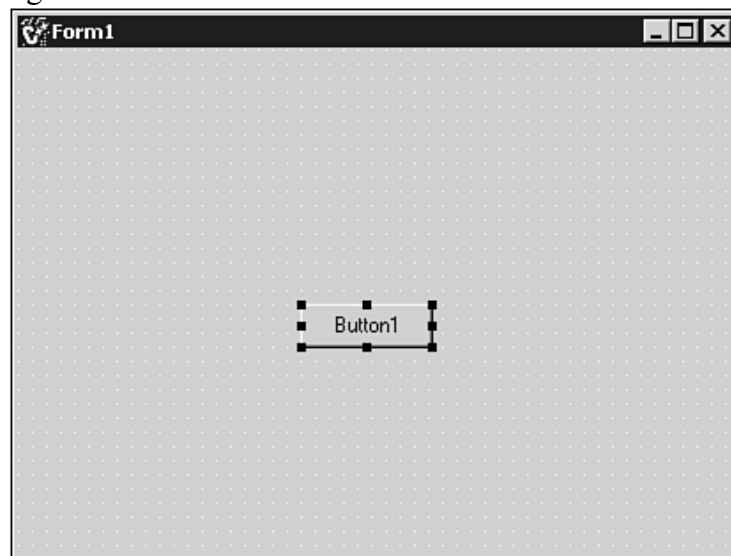
Khi chúng ta tạo một ứng dụng mới trong C++ Builder, một biểu mẫu (form) trông sẽ được tạo một cách tự động. Để xây dựng giao diện ứng dụng của chúng ta, một cách đơn giản là thêm các đối tượng trực quan vào biểu mẫu, sau đó điều chỉnh vị trí và kích thước cho phù hợp. Chúng ta cũng có thể thêm các thành phần không trực quan vào một biểu mẫu, chẳng hạn như các timer; những thành phần này sẽ hiển thị một biểu tượng trong quá trình thiết kế nhưng sẽ không nhìn thấy khi chương trình được thực thi. Chúng ta cũng có thể tạo các biểu mẫu đặc biệt như tool window hoặc dialog. Một chương trình có thể có nhiều biểu mẫu nhưng chỉ có một biểu mẫu làm biểu mẫu chính (main form). Mặc định, khi người sử dụng chạy chương trình của chúng ta, biểu mẫu chính sẽ hiển thị lên màn hình, các biểu mẫu sẽ hiển thị đúng vị trí mà chúng ta đã thiết kế trên màn hình IDE, vị trí này có thể được điều chỉnh lại bởi Object Inspector.

Để đưa các thành phần vào biểu mẫu, chúng ta phải sử dụng các thành phần trong bảng thành phần.

Bảng thành phần (Component Palette) nằm ở dưới menu chính, chứa tất cả các thành phần trong VCL. Các thành phần này được nhóm thành từng nhóm theo tên chủ đề trên các ngăn (tab) trên các thành phần. Để lựa chọn một thành phần, nhấn chuột trái lên nó, sau đó nhấn lại một lần nữa lên biểu mẫu để đặt đối tượng vào vị trí mà chúng ta muốn. Như đã đề cập ở trên, chúng ta có thể điều chỉnh các thuộc tính của thành phần bằng Object Inspector. Chúng ta cũng có thể điều chỉnh lại các thành phần trực quan bằng cách rê cạnh biên để điều chỉnh kích thước của nó, hoặc có thể rê nó đến vị trí mới.

Để đối tượng có thể tương tác được với người sử dụng, chúng ta phải viết mã lệnh để thực thi nhằm đáp ứng các sự kiện (events) do người dùng phát sinh hay do hệ điều hành phát sinh. Các mã lệnh để đáp ứng một sự kiện nào đó gọi là giải quyết sự kiện.

Một ví dụ đầu tiên về việc sử dụng sự kiện trong C++ Builder, chúng ta hãy đưa một nút nhấn (Button) đơn giản vào giữa màn hình như sau:



Hình 2-Nút nhấn Button1 nằm giữa biểu mẫu

C++ Builder đã tạo một nút nhấn như một phần của chương trình. Tại thời điểm này, nút nhấn này vẫn chưa thể thực hiện được gì, nghĩa là nếu chương trình được biên dịch và thực thi thì sẽ không có kết quả gì khi chúng ta nhấn chuột vào nút trên. Chúng ta có thể dùng các đoạn mã lệnh để đáp ứng sự kiện nhằm thực hiện một vài hành động như: lưu thông tin người sử dụng nhập vào, hiển thị một hộp thoại thông báo, hoặc bất cứ việc gì có thể.

Khi nút nhấn được nhấn chuột tại thời điểm thực thi, một sự kiện được phát sinh. Ứng dụng của chúng ta phải có đoạn mã lệnh để xử lý, đoạn mã lệnh này sẽ tự động thực hiện mỗi khi sự kiện tương ứng được phát sinh. Để giải quyết vấn đề này, C++ Builder đã tạo ra sự kiện OnClick để viết các mã lệnh nhằm thực thi khi sự kiện nhấn chuột được phát sinh. Để viết lệnh cho sự kiện này, trong cửa sổ Object Inspector, chúng ta chọn ngăn Events, chọn tên sự kiện OnClick, nhấn đôi chuột trái lên nó hoặc đơn giản hơn, chúng ta có thể nhấn đôi chuột lên đối tượng

Button1 trên biểu mẫu. Khi đó, cửa sổ soạn thảo mã lệnh (Source code editor Window) sẽ xuất hiện với văn bản như sau:

Cấu trúc của sự kiện OnClick như sau:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
}
```

Nếu chúng ta nhấn chuột phải trong cửa sổ soạn thảo mã lệnh và chọn Open Source/Header File, chúng ta sẽ nhìn thấy được đoạn mã lệnh sau:

```
//-----
#ifndef Unit1
#define Unit1
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
__published: // IDE-managed Components
    TButton *Button1;
    void __fastcall Button1Click(TObject *Sender);
private: // User declarations
public: // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
```

Đoạn mã lệnh trên được sinh ra một cách tự động bởi C++ Builder. Tôi đưa vào đây để chúng ta có thể nhìn thấy những gì mà C++ Builder có thể làm cho chúng ta một cách tự động.

Bây giờ, chúng ta sẽ thêm một đoạn mã lệnh để hiển thị một hộp thoại thông báo khi người sử dụng nhấn chuột lên nút nhấn trên. Trong cửa sổ soạn thảo mã lệnh, chúng ta nhấn vào ngăn Unit1.cpp để chuyển về vị trí viết mã lệnh cho sự kiện của Button. Thêm đoạn mã lệnh vào để có được đoạn mã lệnh sau:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage("Hello world! This is a test application! Press OK");
    Close();
}
```

Khi chương trình thực thi, người sử dụng nhấn chuột vào nút nhấn và sự kiện được phát sinh, đoạn mã lệnh trên sẽ được thực thi và hiển thị lên màn hình một thông báo. Khi người sử dụng đóng thông báo này lại, chương trình sẽ được dừng bởi câu lệnh Close().

3. Môi trường phát triển tích hợp IDE và các công cụ đặc trưng

Khi chúng ta khởi động C++ Builder, chúng ta được đặt trong một môi trường phát triển tích hợp gọi là IDE (Integrated Development Environment). IDE cung cấp cho chúng ta tất cả các công cụ cần thiết để thiết kế, phát triển, kiểm tra, gỡ rối (debug) và triển khai các ứng dụng, cho phép phát triển các ứng dụng với thời gian phát triển nhanh hơn.

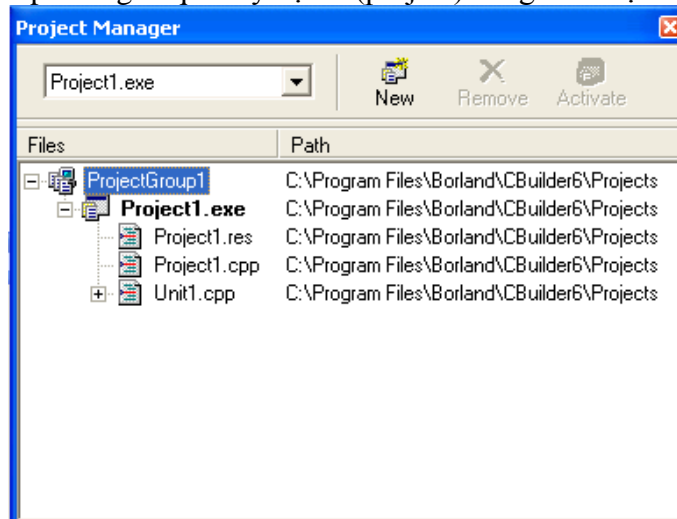
IDE gồm các công cụ sau:

- * Cửa sổ chính và các công cụ

Cửa sổ chính của IDE khít với phần trên của màn hình. Nó chứa các công cụ khác nhau, bao gồm hệ thống thực đơn (menu bar), tất cả chúng có thể được di chuyển. Chúng ta có thể thêm hoặc loại bỏ các nút nhấn bằng cách sử dụng chức năng Customize từ thực đơn đồ xuống (Popup menu) trên bất kỳ thanh công cụ nào. Chúng ta cũng có thể kéo và thả (drag and drop) bất kỳ lệnh nào được liệt kê trong hộp thoại ngăn Commands của hộp thoại Customize vào bất kỳ thanh công cụ nào. Chúng ta chỉ có thể sử dụng những công cụ đã có mà không được tạo ra bất kỳ công cụ nào cũng như xóa chúng đi. Sử dụng cách tương tự như trên để loại bỏ các nút nhấn bằng cách chọn chức năng customize và kéo nút nhấn ra khỏi thanh công cụ đang giữ nó.

* Project Manager

Project Manager không hiển thị một cách mặc định nhưng nó thường rất quan trọng. Để hiển thị cửa sổ này (dĩ nhiên trong trường hợp nó chưa hiển thị), chúng ta chọn nó từ thực đơn View. Project Manager cho phép chúng ta quản lý dự án (project) đang làm việc.



Hình 3- Cửa sổ Project Manager

Như hình minh họa trên, cửa sổ project manager hiển thị các tập tin trong dự án đang làm việc. Chúng ta có thể sắp xếp lại thứ tự biên dịch của những file này trong cửa sổ Project Manager và thực hiện nhiều thao tác khác nhờ thực đơn đồ xuống cho từng tập tin hoặc cho cả dự án.

IDE cung cấp hai con đường để phát triển bằng cách kết hợp cửa sổ thiết kế biểu mẫu và cửa sổ mã lệnh. Project Manager được dùng chủ yếu cho quản lý các tập tin trong dự án của C++ Builder.

Các dự án của C++ Builder bao gồm rất nhiều kiểu tập tin khác nhau. C++ Builder tạo một vài tập tin một cách tự động; khi một dự án mới được khởi tạo hay khi một thành phần (như tập tin, hình ảnh ngoài, ...) được thêm vào dự án hiện có. Các tập tin trong dự án mà chúng ta cần quan tâm có thể chia ra thành các chủ đề như sau:

Các tập tin chính của dự án: Khi chúng ta tạo một dự án mới, ba tập tin sau đây được tạo:

- Tập tin dự án: ProjectName .bpr
- Tập tin mã nguồn: ProjectName .cpp
- Tập tin tài nguyên (resource): ProjectName .res

Các tập tin biểu mẫu: Khi chúng ta tạo một biểu mẫu, C++ Builder sẽ sinh ra các tập tin sau:

- Tập tin bố cục (layout): Unitname.dfm
- Tập tin mã nguồn: UnitName.cpp
- Tập tin tiêu đề (header - một tập tin chứa các khai báo hàm, lớp, ...): UnitName.h

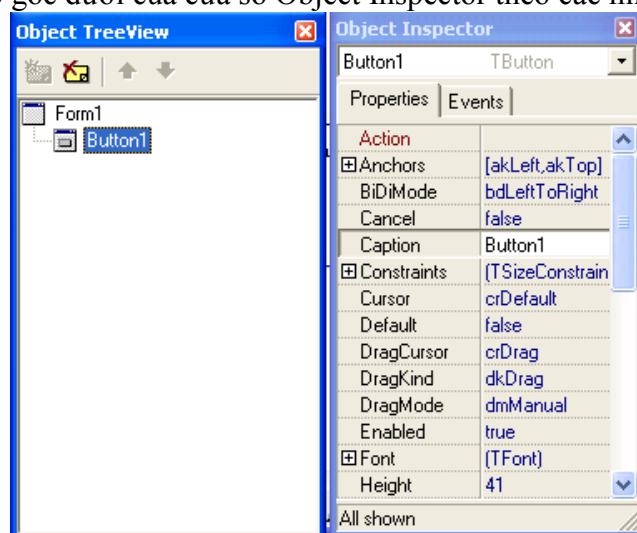
Các tập tin gói: Các gói đơn giản là các thư viện liên kết động được chia sẻ cho nhiều ứng dụng C++ Builder. Nó cho phép chúng ta chia sẻ các lớp, các thành phần và dữ liệu giữa các ứng dụng. Ví dụ, các đối tượng thường dùng trong C++ Builder cư trú trong một gói gọi là VCL. Hầu hết các ứng dụng được tạo bằng C++ Builder đều chia sẻ các đoạn mã lệnh trong gói này, gói này có tên VCL60.bpl.

Khi chúng ta tạo ra các thành phần riêng hoặc thư viện của riêng chúng ta, chúng ta phải tạo chúng trong một dự án gói (package project). Dự án gói chứa một tập tin .bpk giống như tập tin .bpr nhưng nó chỉ dùng cho dự án gói. Tập tin này được tạo ra khi chúng ta chọn thực đơn File/New/Package. Dự án gói sau khi được biên dịch sẽ trở thành thư viện gói (Package Library), chúng sẽ có phần mở rộng .bpl. Đây là thư viện thực thi (nghĩa là không chứa mã lệnh, chỉ chứa mã nhị phân) được sinh ra cho gói. Nó giống như thư viện DLL, ngoại trừ chúng chứa các thành phần được chỉ định trong C++ Builder khi chúng ta thiết kế nó. Mỗi một lần chúng ta biên dịch một gói, một tập tin có phần mở rộng .bpi sẽ được tạo ra, tập tin này gọi là thư viện nhập khẩu (import library). Chúng ta sẽ bàn kỹ hơn về vấn đề này trong một chương khác.

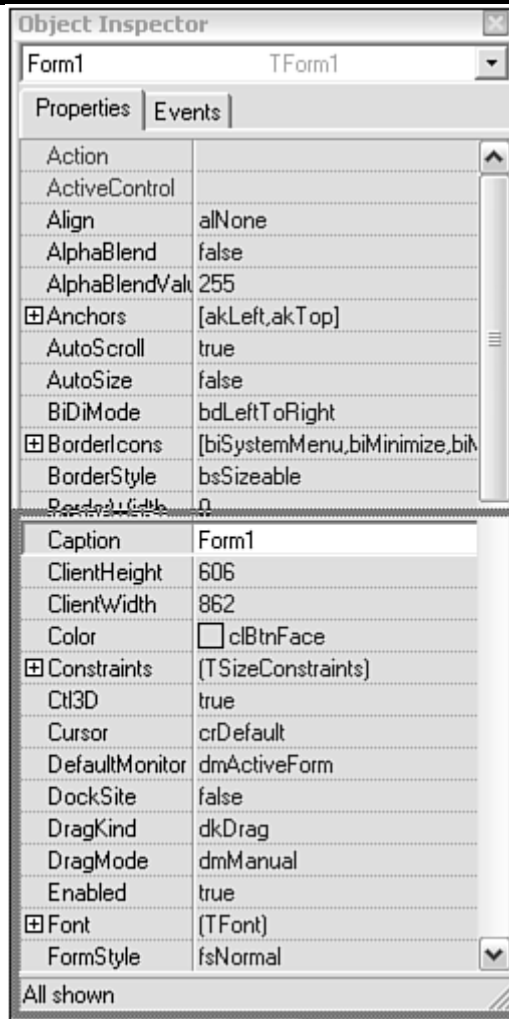
Các tập tin sao lưu để dự trữ: C++ Builder sẽ tiến hành sao lưu các tập tin trong dự án mỗi lần dự án được lưu lại, dĩ nhiên, ngoại trừ lần lưu đầu tiên. Các tập tin này có tiếp đầu ngữ ~ cho phần mở rộng của nó.

* Sắp xếp cửa sổ trong IDE

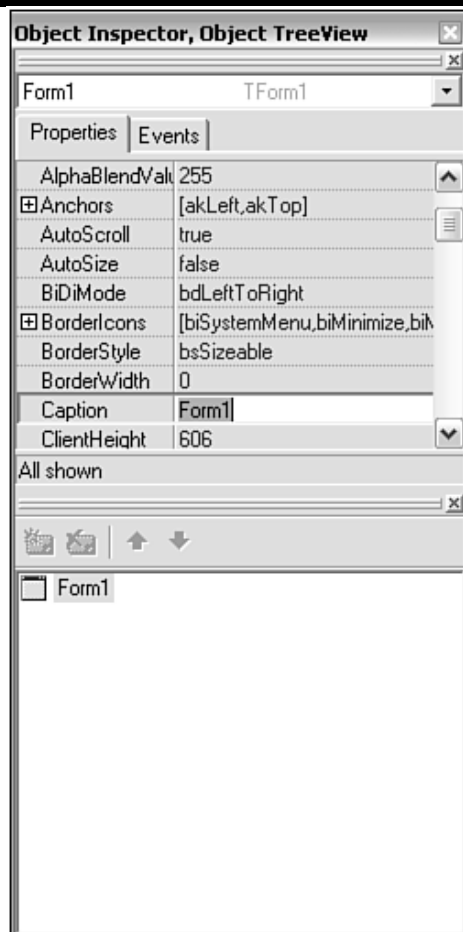
Các cửa sổ trong môi trường IDE có thể neo lại với nhau. Chức năng này cho phép chúng ta tùy biến bố cục (layout) giao diện theo nhu cầu của chúng ta. Thao tác điều chỉnh thật hết sức đơn giản bằng cách kéo và thả các cửa sổ. Chẳng hạn, trong hình minh họa sau, cửa sổ Object Tree View được neo vào góc dưới của cửa sổ Object Inspector theo các hình minh họa sau:



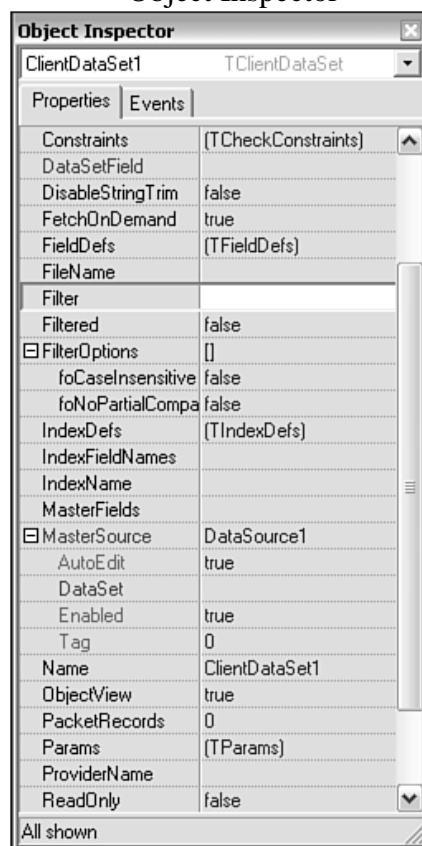
Hình 4 - Hai cửa sổ chưa được neo



Hình 5 - Đang neo cửa sổ



Hình 6 - Đã neo xong cửa sổ
* Object Inspector



Hình 7-Cửa sổ Object Inspector

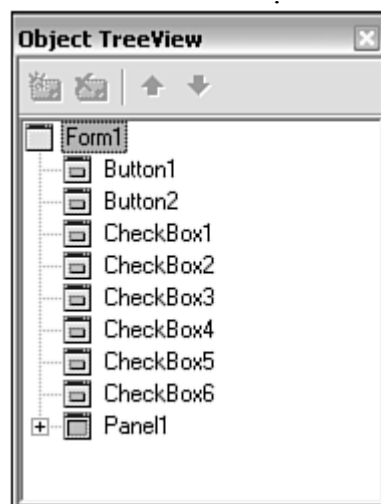
Cửa sổ Object Inspector hiển thị các thuộc tính cho đối tượng hay biểu mẫu đang được chọn. Chúng ta cũng có thể lựa chọn các thành phần từ danh sách đồ xuống (drop-down list) ở trên cùng của cửa sổ Object inspector.

Hình minh họa cho thấy Object Inspector đang hiển thị các thuộc tính của thành phần TclientDataSet, tuy nhiên các thuộc tính có thể sẽ không giống nhau cho mỗi thành phần.

Chúng ta có thể nhấn nút cộng (+) để xem thêm các thuộc tính khác nằm trong một thuộc tính. Object inspector dùng màu sắc để hiển thị bản chất của thuộc tính. Những thuộc tính có tên màu đỏ chỉ rằng nó tham chiếu đến những thành phần khác. Màu xanh lá cây (green) chỉ định thuộc tính này được mang sang từ những đối tượng được tham chiếu bởi thuộc tính này nhằm tiện nghi trong việc sửa chữa các thuộc tính của thành phần đang tham chiếu.

* Object Tree View

Object Tree View là một bộ trợ quan trọng cho Object Inspector. Trong trường hợp các đối tượng có hơn mười hay nhiều hơn, Object Inspector sẽ hiển thị không phù hợp. Trong khi đó, cửa sổ Object Tree View hiển thị với quan hệ cha con và có thể cuộn lên xuống. Dĩ nhiên, nếu trường hợp các đối tượng con quá nhiều, chúng ta vẫn không thể nhìn thấy phần tử cha của nó (đã bị cuộn lên trên), do đó, vẫn khó khăn khi click chuột lên đối tượng cha.

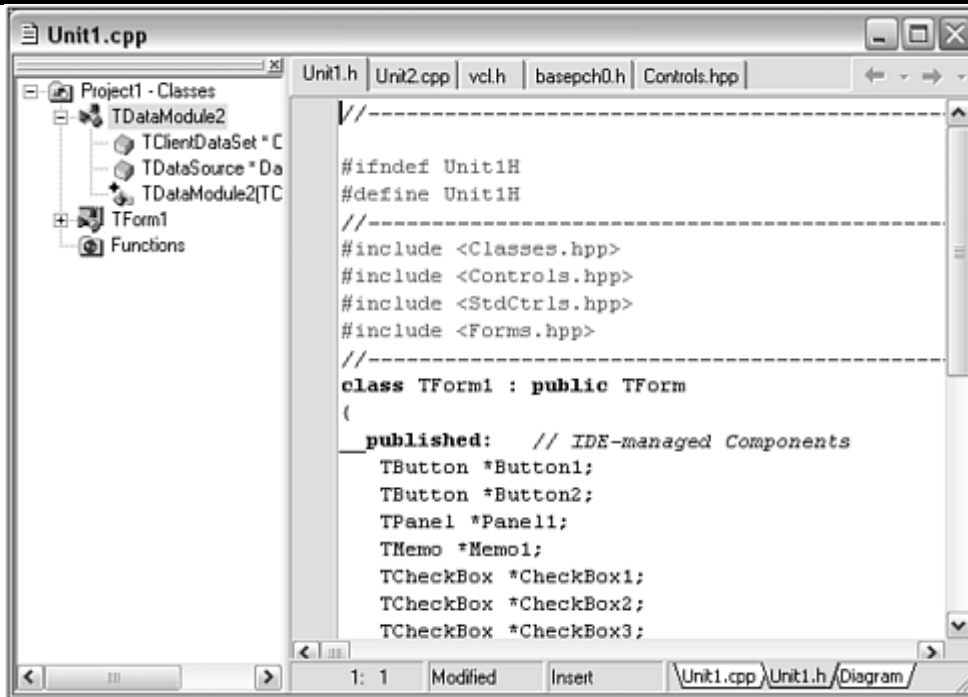


Hình 8-Cửa sổ Object TreeView

Chúng ta có thể sử dụng Object TreeView để điều chỉnh các đối tượng, chẳng hạn như tạo bảng sao.

* Cửa sổ soạn thảo mã lệnh

Cửa sổ soạn thảo mã lệnh được minh họa như hình sau:

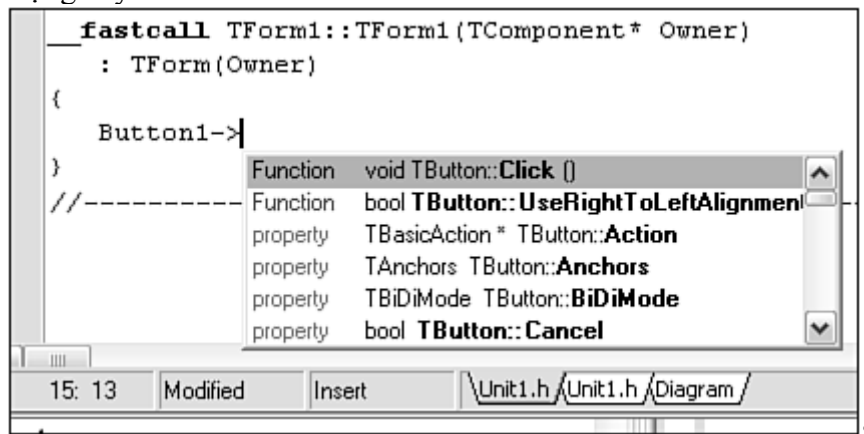


Hình 9-Cửa sổ soạn thảo mã lệnh

Cửa sổ được chia thành nhiều ngăn, mỗi ngăn chứa một tập tin được mở. Nó cho phép mở và chỉnh sửa nhiều loại tập tin khác nhau như các tập tin của C++, các tập tin của C++ Builder, tập tin văn bản, ...

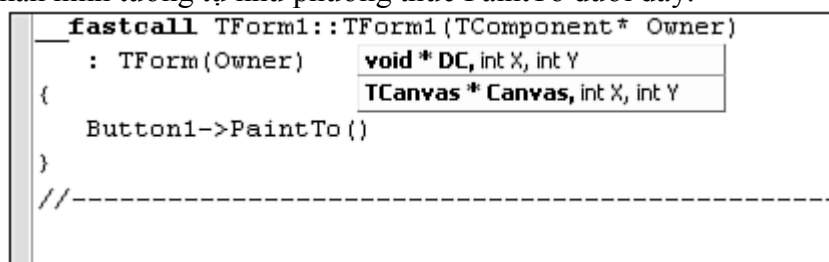
* Code Insight

Code Insight là một thành phần quan trọng của IDE. Thật khó khăn cho các nhà phát triển khi phải nhớ rất nhiều các hàm, các phương thức và cả tham số của chúng nữa. Code Insight cho phép chúng ta xem tất cả những phương pháp và tham số ngay khi chúng ta nhập mã lệnh trong cửa sổ soạn thảo mã lệnh. Hình sau cho thấy Code Insight cho thấy một tập hợp các phương thức hiện có của đối tượng này.



Hình 10-Code insight

Các phương thức có thể trùng tên nhưng danh sách tham số khác nhau. Trong trường hợp này chúng ta sẽ gặp màn hình tương tự như phương thức PaintTo dưới đây.

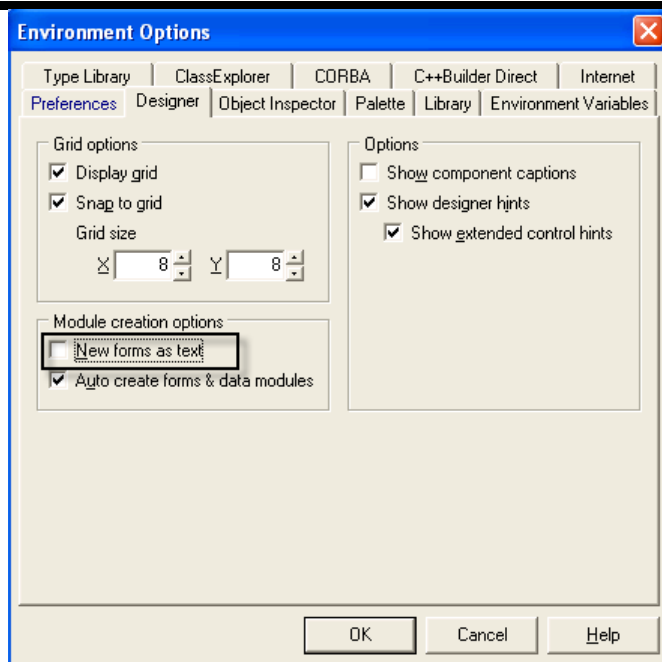


Hình 11 - PaintTo() được nạp chồng

Khi một biểu mẫu được lưu lại, C++ Builder sẽ lưu thiết kế của Form dưới dạng văn bản trong tập tin DFM. Ví dụ:

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 311
  Height = 158
  Caption = 'Text Form'
  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
  object Label1: TLabel
    Left = 13
    Top = 24
    Width = 277
    Height = 20
    Caption = 'This form is saved as text (default)'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = [fsBold]
    ParentFont = False
  end
  object Button1: TButton
    Left = 114
    Top = 72
    Width = 75
    Height = 25
    Caption = 'OK'
    TabOrder = 0
    OnClick = Button1Click
  end
end
```

Ngoài cách lưu trữ trên, chúng ta có thể click chuột phải lên biểu mẫu trong lúc thiết kế, bỏ chọn mục Text DFM; khi đó biểu mẫu sẽ được lưu dưới dạng nhị phân. Chúng ta có thể điều chỉnh chức năng này thành mặc định bằng cách chọn thực đơn Tools/ Environment Options. Trong cửa sổ này chọn ngăn Desinger, bỏ chọn chức năng New Form as Text.



Hình 12-Điều chỉnh để thiết kế biểu mẫu lưu ở dạng nhị phân

Để chuyển một dự án từ Visual C++, chúng ta sử dụng công cụ chuyển đổi của C++ Builder được gọi bằng lệnh Tools/Visual C++ Project Conversion Utility.

*** CodeGuard**

CodeGuard cung cấp các thư viện thực thi để truy bắt lỗi để phát triển ứng dụng trong C++ Builder. Nó giúp cho phép tạo báo cáo về những lỗi mà trình dịch không bắt được vì những lỗi này không vi phạm cú pháp đồng thời hỗ trợ đầy đủ trong các ứng dụng đa tuyến trình.

Image Editor (trình soạn thảo ảnh)

Đây là công cụ cho phép chỉnh sửa nhiều kiểu ảnh khác nhau. Để mở công cụ này, chúng ta chọn Tools/Image Editor.

*** XML Mapper**

XML Mapper là công cụ thiết kế cho phép chúng ta gắn kết tài liệu XML và gói thao tác dữ liệu cho dataset nhằm tạo một ánh xạ cho mỗi phần tử trong tài liệu XML với mỗi trường trong gói thao tác dữ liệu.

Chúng ta cũng có thể tạo sự gắn kết từ tài liệu giản đồ XML với một dataset.

BÀI 2- TỔNG QUAN VỀ VCL

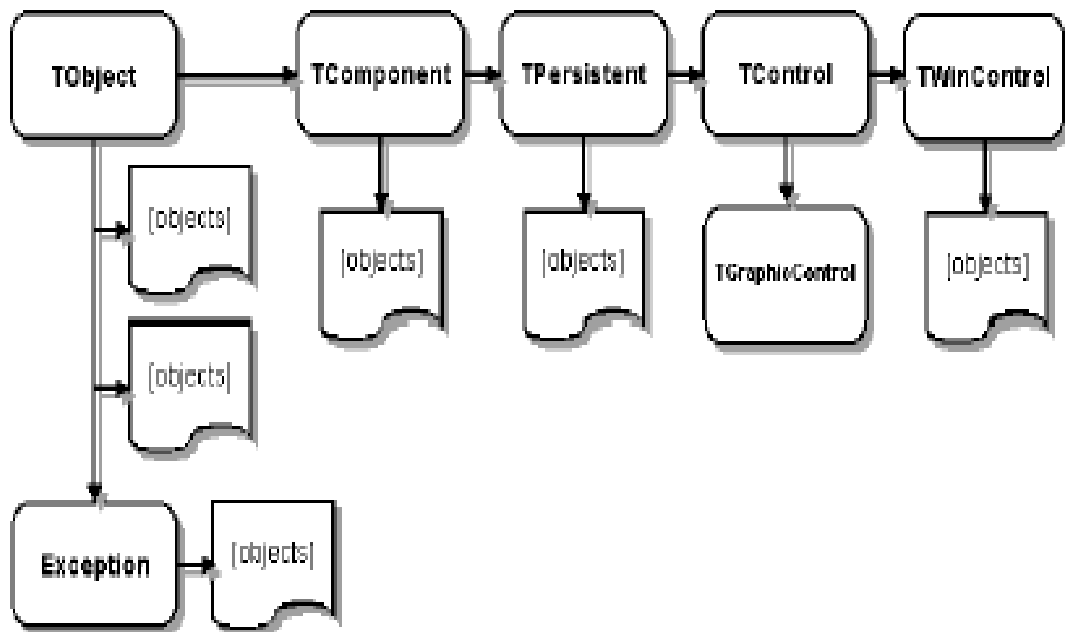
1. Giới thiệu VCL.

VCL là viết tắt của cụm từ Visual Component Library, VCL là thư viện trực quan dùng để lập trình trên môi trường Windows. Tất cả các đối tượng VCL đều được tham chiếu dưới dạng con trỏ và lưu trữ trên vùng nhớ động (Heap). VCL được viết bằng C++ Builder; có thể nói VCL là hạt nhân cho hầu hết các công cụ được xây dựng bởi Borland.

Các đối tượng của VCL được xây dựng theo mô hình phân cấp thừa kế. Cây thừa kế này ngày sẽ được mở rộng thông qua các phiên bản mới.

VCL là cơ sở cơ lập trình PME bao gồm các thuộc tính (properties), phương thức (methods) và sự kiện (Events) giúp phát triển các ứng dụng nhanh (RAD-Rapid Application Development). Người phát triển có thể xây dựng các ứng dụng với ít dòng lệnh nhất bởi lẽ VCL đã được xây dựng để nắm giữ rất nhiều công việc giúp bạn.

2. Mô hình phân cấp VCL.



Hình 13 – Mô hình phân cấp VCL

Mô hình phân cấp và cây thừa kế của C++ Builder có thể tìm thấy trong trang web <http://www.borland.com> hoặc có thể liên lạc với địa chỉ email: lexuanthach@gmail.com để nhận poster về cây thừa kế về VCL.

Sau đây, chúng tôi sẽ trình bày ngắn gọn về một số phân nhánh và cố gắng phân tích một số đối tượng trong cây thừa kế.

Lớp cao nhất trong cây thừa kế là TObject, lớp TObject là lớp trừu tượng. Lớp này cung cấp các khả năng để đáp ứng việc khởi tạo và hủy bỏ của các đối tượng, cung cấp sự nắm giữ thông điệp, và chứa các kiểu lớp, thông tin kiểu thực thi các thuộc tính xuất bản (Runtime Type Information- RTTI). RTTI cho phép chúng ta xác định kiểu của một đối tượng vào thời điểm thực thi thậm chí khi mã lệnh sử dụng con trỏ hay tham chiếu đến đối tượng đó. Ví dụ, C++ Builder chuyển một con trỏ kiểu TObject như một tham số cho mỗi sự kiện của nó. RTTI cũng có thể thực hiện một phép toán ép kiểu để sử dụng một đối tượng hoặc xác định kiểu của đối tượng. Chúng ta có thể để kiểm tra xem kiểu của một đối tượng bằng cách sử dụng phép toán typeid.

Một thuật ngữ của VCL là Persistent data (tạm dịch: dữ liệu liên tục) ám chỉ các được lưu trữ trong các giá trị thuộc tính. Một ví dụ đơn giản nhất là thuộc tính Caption của một nút nhấn hoặc một nhãn. Thuộc tính này có thể được điều chỉnh trong lúc thiết kế và lúc thực thi bằng mã lệnh hay giữa các phiên làm việc.

Các lớp không liên tục (Nonpersitent classes) ám chỉ các lớp đơn giản được thiết kế để thực hiện các hàm cụ thể nhưng nó không cho phép lưu trữ dữ liệu giữa các phiên làm việc.

Một trình bao bọc có thể được diễn giải như một nơi để thay thế cho sự phức tạp của các hàm Windows API. Nó cho phép tạo ra các đối tượng hoặc các thành phần để dễ sử dụng và sử dụng chéo cho nhiều dự án khác nhau.

Một lớp thường được sử dụng là lớp Exception. Nó cung cấp rất nhiều lớp ngoại lệ dựng sẵn để nắm giữ các lỗi ngoại lệ như lỗi chia cho 0, ..

Các nhánh chính của TObject bao gồm TPersistent, TComponent, TControl, TGraphicControl và TWinControl.

TPersistent thêm các phương thức cho TObject nhằm cho phép các đối tượng lưu lại trạng thái của nó trước khi bị huỷ đi và nạp lại trạng thái đó khi nó được tạo lại lần nữa. Lớp này quan trọng trong việc tạo ra những thành phần chứa đựng các thuộc tính có dữ liệu là lớp. Nhánh này cũng bao gồm nhiều lớp khác như: TCanvas, TClipboard, TGraphic, TGraphicsObject, và TStrings. TPersistent rẽ xuống TComponent, nó là nền tảng cho tất cả các thành phần VCL.

Thuộc tính dòng (Property streaming) ám chỉ những thuộc tính này có giá trị được lưu giữ trong tập tin của biểu mẫu (form's file). Khi dự án được mở ra lại thì giá trị sẽ được phục hồi như cũ (chẳng hạn giá trị tại thời điểm thiết kế).

Các đối tượng TComponent cung cấp nền tảng cho việc xây dựng các ứng dụng C++ Builder, chúng được đặt trên bảng công cụ và trở thành cha của các thành phần khác.

Có hai loại thành phần: trực quan và không trực quan. Những thành phần không trực quan không yêu cầu giao diện trực quan, do đó nó phải sinh trực tiếp từ TComponent. Các thành phần trực quan yêu cầu phải được nhìn thấy và tương tác với người sử dụng trong lúc thực thi. TControl thêm các chương trình con để vẽ và các sự kiện cần thiết để định nghĩa một thành phần trực quan. Những đối tượng trực quan chia thành hai nhóm: được cửa sổ hoá (TWinControl) và không được cửa sổ hoá (TGraphicControl).

Các thành phần TGraphicControl có nhiệm vụ vẽ chính nó nhưng không bao giờ nhận được chỉ điểm (focus). Ví dụ như TImage, TLabel, TBevel và TShape.

Thành phần TWinControl cũng tương tự như TGraphicControl ngoại trừ nó có thể nhận được chỉ điểm, cho phép tương tác với người sử dụng. Những thành phần này được hiểu là được cửa sổ hoá, nó có một thẻ quản cửa sổ và có thể chứa trong nó các đối tượng khác (cha của đối tượng khác).

3. Các thuộc tính, biến cố chung với TControl

Trong phần này, chúng tôi liệt kê các thuộc tính, phương thức và sự kiện thường được sử dụng, còn một số thuộc tính, phương thức, sự kiện phức tạp khác chúng ta sẽ bàn luận kỹ hơn khi gặp nó.

3.1. Các thuộc tính

- **Action**: Chỉ định hành động kết hợp với đối tượng.

Những hành động (Action) cho phép một ứng dụng tập trung các đáp ứng cho các lệnh. Khi một điều khiển kết hợp với một hành động, hành động sẽ tự xác định các thuộc tính và sự kiện thích hợp của điều kiện. (chẳng hạn nó có thể tác động lên thuộc tính Enabled hoặc nó chứa mã lệnh đáp ứng cho sự kiện OnClick)

- **Align**: Chỉ định cạnh lề cho đối tượng trong đối tượng chứa nó.

Sử dụng thuộc tính này để đính đối tượng vào các biên của biểu mẫu hay panel.

Giá trị mặc định của Align là alNone, có nghĩa là không có đính.

Nếu giá trị này là alClient thì đối tượng sẽ lấp đầy toàn bộ vùng hiển thị của đối tượng chứa nó.

- **Anchors**: Chỉ định cách thức mà đối tượng sẽ neo vào đối tượng chứa nó.

Sử dụng thuộc tính này để chắc chắn rằng luôn luôn bảo toàn vị trí của nó hiện tại trên đối tượng cha, thậm chí đối tượng cha có bị thay đổi kích cỡ. Khi đối tượng cha thay đổi kích cỡ, đối tượng sẽ thay đổi kích cỡ theo sự liên hệ với các cạnh của đối tượng cha mà nó neo vào.

Ví dụ: Chúng ta có thể neo đối tượng vào cạnh trái và cạnh phải của đối tượng cha (akLeft=true, akRight=true) đối tượng sẽ giãn theo chiều rộng mỗi khi chiều rộng của biểu mẫu thay đổi.

- AutoSize: Chỉ định rằng kích cỡ của đối tượng sẽ tự động thay đổi theo độ lớn của nội dung. Thường thường, giá trị mặc định cho thuộc tính này false.

- BoundsRect: Là thuộc tính có kiểu dữ liệu TRect lưu trữ hình chữ nhật nhỏ nhất để chứa được đối tượng. Đây là phương cách để chúng ta lấy các cạnh của đối tượng.

Ví dụ: Câu lệnh R = Control->BoundsRect;

tương ứng với các câu lệnh sau:

R.Left = Control->Left;

R.Top = Control->Top;

R.Right = Control->Left + Control->Width;

R.Bottom = Control->Top + Control->Height;

- BiDiMode: Hướng hiển thị của đối tượng như từ trái sang phải, từ phải sang trái, Tuy nhiên, thuộc tính này sẽ không được thay đổi trong một số trường hợp như dữ liệu được lấy từ trường có kiểu là flInteger, ...

- Caption/Text: Xâu văn bản hiển thị trên đối tượng.

- ClientOrigin: Toạ độ trên màn hình (Screen) của góc trên trái.

- ClientRect: Kích thước xác định của hình chữ nhật hiển thị đối tượng. (Đo bằng điểm ảnh).

- ClientHeight: Chiều cao của đối tượng.

- ClientWidth: Chiều rộng của đối tượng.

- Color: Màu nền của đối tượng.

- ControlState: Trạng thái hiện tại của đối tượng vào thời điểm thực thi.

- ControlStyle: Định đặt tính của kiểu dáng.

- Cursor: Con trỏ chuột hiển thị khi di chuyển lên đối tượng.

- DockOrientation: Chỉ định cách thức đối tượng kết dính với đối tượng khác trong cùng một đối tượng chứa chúng.

- DragCursor: Con trỏ di rê đối tượng trong chức năng kéo thả.

- DragKind: Kiểu của kéo thả: kéo thả bình thường hay kéo thả kết dính.

- DragMode: Chế độ kéo thả: kéo thả bằng tay hay kéo thả tự động.

- Enabled: Cho phép đối tượng nhận chỉ điểm hay không.

- Floating: Chỉ định cách để một đối tượng nổi lên.

- FloatingDockSiteClass: Xác định lớp của đối tượng tạm thời để chứa đối tượng khi nó nổi lên.

- Font: Kiểu phông chữ cho đối tượng như: tên kiểu chữ, kiểu dáng chữ, kích cỡ, ...

- Height: Độ cao của đối tượng.

- HelpContext, HelpKeyword, HelpType: Chỉ định các giá trị để kết gán hướng dẫn với đối tượng.

- Hint: Dòng văn bản nhắc nhở ngắn thường để tóm lược chức năng của đối tượng.

- HostDockSite: Chỉ định đối tượng mà đối tượng có thuộc tính này sẽ kết dính vào.

- Left: Giá trị tọa độ x của cạnh bên trái.

- LRDockWidth: Chỉ định chiều rộng của đối tượng mỗi khi nó được kết dính vào đối tượng khác.

- MouseCapture: Xác định liệu đối tượng có “chụp” sự kiện của chuột hay không? Khi được phép, mọi sự kiện sẽ đến đối tượng cho đến khi người dùng thả phím chuột ra.

- Name: Tên của đối tượng.

- Parent: Đối tượng cha của đối tượng hiện tại. Nó chỉ đến đối tượng sẽ chứa đối tượng hiện tại.

- ParentBiDiMode: Chỉ định liệu xem đối tượng hiện tại có sử dụng thuộc tính BidiMode của đối tượng cha hay không?

- ParentColor: Chỉ định xem đối tượng có sử dụng màu của đối tượng cha hay không.

- ParentFont: Chỉ định xem đối tượng này có sử dụng màu

- ParentShowHint: Chỉ định xem đối tượng có sử dụng thuộc tính Hint của đối tượng cha hay không.
- PopupMenu: Chỉ đến thực đơn ngữ cảnh (thực đơn hiển thị khi nhấn chuột phải lên đối tượng)
- ShowHint: Chỉ định xem đối tượng có được phép hiển thị giá trị chỉ dẫn ngắn trong thuộc tính Hint hay không.
- TBDockHeight: Chỉ định chiều cao của đối tượng khi nó kết dính vào đối tượng khác.
- Text: Văn bản người sử dụng nhập vào đối tượng hoặc được lập trình viên thiết kế bằng cửa sổ thiết kế hoặc bằng mã lệnh.
- Top: Chỉ định tọa độ Y của cạnh trên của đối tượng.
- Visible: Chỉ định đối tượng có được nhìn thấy hay không nhìn thấy.
- Width: Chiều rộng của đối tượng.
- WindowProc: Chỉ định địa chỉ của hàm sẽ xử lý các sự kiện gửi đến đối tượng.
- WindowText: Văn bản đính kèm với đối tượng.
- Tag: Lưu trữ một số nguyên đính kèm với đối tượng, nó là một phần của đối tượng. Thường giá trị này không có ý nghĩa về mặt xử lý.

3.2. Các phương thức

- ActionChange: Được gọi một cách tự động mỗi khi thuộc tính Action bị thay đổi hoặc các thuộc tính của đối tượng được chỉ trong Action bị thay đổi.
- AdjustSize: Được gọi một cách tự động bởi thuộc tính AutoSize để điều chỉnh lại kích thước của đối tượng cho khớp với dữ liệu hiển thị.
- AssignTo: Gán những giá trị cho đối tượng TAction.
- BeginAutoDrag: Được gọi một cách tự động khi đối tượng bắt đầu được kéo trong chế độ kéo và thả tự động.
- BeginDrag: Được gọi một cách tự động khi đối tượng bắt đầu được kéo trong chế độ kéo và thả tự động. Nếu chúng ta viết mã lệnh để kéo và thả đối tượng bằng tay, chúng ta phải gọi phương thức này để đối tượng đưa vào trạng thái bắt đầu được kéo.
- BringToFront: Đưa đối tượng lên trên các đối tượng khác nằm trong cùng một đối tượng cha.
- CanAutoSize: Kiểm tra xem đối tượng có thể tự động điều chỉnh kích thước hay không?
- CanResize: Kiểm tra xem có thể thay đổi được kích thước của đối tượng.
- Changed: Phát sinh sự kiện OnChange cho đối tượng.
- ChangeScale: Thay đổi tỷ lệ hiển thị của đối tượng.
- Click: Được sử dụng để phát sinh sự kiện OnClick hoặc được gọi một cách tự động bởi sự kiện OnClick.
- ClientToParent: Chuyển tọa độ sang tọa độ của đối tượng cha.
- ClientToScreen: Chuyển tọa độ sang tọa độ của màn hình.
- Create: Khởi tạo đối tượng. Không sử dụng trực tiếp mà phương thức này sẽ được gọi một cách tự động khi khởi tạo bằng từ khóa new.
- DblClick: Được sử dụng để phát sinh sự kiện OnDblClick.
- Destroy: Được gọi để hủy đối tượng.
- DoDock: Bắt đầu quá trình kết dính đối tượng.
- DoEndDock: Kết thúc quá trình kết dính đối tượng.
- DoEndDrag: Kết thúc quá trình rê và thả đối tượng.
- DoMouseWheel: Phát sinh sự kiện cuộn nút giữa của chuột.
- DoMouseWheelDown: Phát sinh sự kiện cuộn xuống của nút giữa chuột.
- DoMouseWheelUp: Phát sinh sự kiện cuộn lên của nút giữa chuột.
- DoStartDock: Bắt đầu quá trình kết dính đối tượng.
- DoStartDrag: Bắt đầu quá trình kéo thả đối tượng.
- DragCanceled: Hủy công việc kéo thả.
- DragDrop: Phát sinh sự kiện OnDragDrop.
- Dragging: Thử kiểm tra xem đối tượng có phải là đang được kéo hay không.
- DragOver: Phát sinh sự kiện OnDragOver.

- EndDrag: Kết thúc kéo và thả đối tượng.
- GetClientOrigin: Trả về tọa độ màn hình góc trên trái của đối tượng.
- GetClientRect: Trả về hình chữ nhật nghĩa vùng hiển thị của đối tượng.
- GetControlsAlignment: Trả về giá trị canh lề của văn bản bên trong đối tượng.
- GetDragImages: Trả về ImageList chứa các hình ảnh mà cont rõ
- GetEnabled: Lấy giá trị của thuộc tính Enabled.
- GetFloating: Lấy giá trị của thuộc tính Floating.
- GetPopupMenu: Lấy giá trị của thuộc tính PopupMenu..
- GetTextBuf: Trả về văn bản được sao chép vào bộ nhớ đệm.
- GetTextLen: Trả về chiều dài văn bản của đối tượng.
- HasParent: Trả về kết quả xem đối tượng có đối tượng cha hay không?
- Hide: Ẩn đối tượng, giống như chúng ta gán thuộc tính Visible=false.
- MouseDown: Phát sinh sự kiện OnMouseDown.
- MouseMove: Phát sinh sự kiện OnMouseMove.
- MouseUp: Phát sinh sự kiện OnMouseUp.
- ParentToClient: Chuyển tọa độ từ đối tượng cha sang tọa độ của đối tượng con.
- Perform: Phát sinh một sự kiện chỉ định.
- Refresh: Làm tươi lại đối tượng.
- Repaint: Vẽ lại đối tượng.
- Resize: Thay đổi kích cỡ của đối tượng.
- ScreenToClient: Chuyển tọa độ màn hình sang tọa độ đối tượng.
- SendCancelMode: Huỷ bỏ trạng thái “modal” của đối tượng.
- SendToBack: Chuyển đối tượng nằm dưới tất cả các đối tượng khác.
- SetBounds: Cài đặt các đường biên cho đối tượng.
- SetDragMode: Cài đặt chế độ kéo thả cho đối tượng.
- SetEnabled: Cài đặt thuộc tính Enabled cho đối tượng.
- SetParent: Cài đặt đối tượng cha cho đối tượng.
- SetTextBuf: Cài đặt văn cho đối tượng.
- SetZOrder: Cài đặt thứ tự nhận thông điệp trong đối tượng cha của nó.
- Show: Hiện đối tượng.
- Update: Xử lý ngay lập tức các sự kiện yêu cầu vẽ lại đối tượng đang chờ đợi.

3.3. Các sự kiện chung

- OnCanResize: Xảy ra khi đối tượng thay đổi kích thước.
- OnClick: Phát sinh khi người dùng nhấn chuột trái lên đối tượng. Sự kiện này phát sinh trong rất nhiều tình huống như:

Người sử dụng chọn một phần tử trong lưới, đường biên, danh sách hoặc hộp danh sách đổ xuống bằng cách nhấn phím mũi tên.

Người sử dụng nhấn phím SpaceBar trong khi một nút nhấn hay một ô đánh dấu có chỉ điểm.

Người sử dụng nhấn phím Enter khi biểu mẫu kích hoạt có nút nhấn mặc định là nút nhấn phát sinh sự kiện này.(được chỉ định bởi thuộc tính Default của biểu mẫu).

Người sử dụng nhấn phím Cancel khi biểu mẫu kích hoạt có nút nhấn huỷ bỏ trở đến đối tượng phát sinh sự kiện này (được chỉ định bởi thuộc tính Cancel của biểu mẫu).

Khi người sử dụng nhấn các phím nóng hay phím tắt của nút nhấn hoặc ô đánh dấu.

Thuộc tính Checked của ô chọn được cài đặt thành true.

Khi phương thức Click của một thực đơn được gọi.

- OnContextPopup: Phát sinh khi thực đơn ngữ cảnh được mở ra. (Thường là nhấn chuột phải để mở thực đơn này).

- OnDblClick: Phát sinh khi nhấn đôi chuột lên đối tượng.

- OnDragDrop: Phát sinh khi đối tượng được kéo.

- OnDragOver: Phát sinh khi có đối tượng kéo di chuyển trên bề mặt của đối tượng phát sinh sự kiện này.

- OnEndDock: Phát sinh khi kết thúc quá trình kết dính.

- OnEndDrag: Phát sinh khi kết thúc quá trình kéo và thả.

- OnMouseDown: Phát sinh khi nhấn chuột xuống.
- OnMouseMove: Phát sinh khi di chuyển chuột trên đối tượng.
- OnMouseUp: Phát sinh khi thả phím chuột ra (sau khi nhấn lên đối tượng và thả ra).
- OnMouseWheel: Phát sinh khi nút cuộn trên con chuột được cuộn.
- OnMouseWheelDown: Phát sinh khi người dùng cuộn xuống bằng nút cuộn của chuột.
- OnMouseWheelUp: Phát sinh khi người dùng cuộn lên bằng nút cuộn của chuột.
- OnResize: Phát sinh khi đối tượng được thay đổi kích thước.
- OnStartDock: Phát sinh khi quá trình kết dính đối tượng được bắt đầu.
- OnStartDrag: Phát sinh khi quá trình kéo và thả đối tượng được bắt đầu.

4. Các thuộc tính, biến cố chung với TWinControl.

Trong phần này, chúng tôi liệt kê các thuộc tính, phương thức và sự kiện thường được sử dụng, còn một số thuộc tính, phương thức, sự kiện phức tạp khác chúng ta sẽ bàn luận kỹ hơn khi gặp nó.

4.1. Thuộc tính chung

- BevelEdges: Xác định những cạnh của đối tượng được tạo cạnh xiên.
- BevelInner: Xác định độ sâu của các cạnh trong của đối tượng có cạnh được tạo cạnh xiên.
- BevelKind: Xác định kiểu cạnh xiên cho cạnh của đối tượng.
- BevelOuter: Xác định độ nổi của các cạnh trong của đối tượng có cạnh được tạo cạnh xiên.
- BevelWidth: Xác định của độ rộng giữa cạnh chìm và cạnh nổi.
- BorderWidth: Xác định độ rộng của đường viền.
- Brush: Chứa các giá trị làm nền cho đối tượng như: màu nền, kiểu nền, ...
- ClientRect: Hình chữ nhật chứa đối tượng.
- ControlCount: Tổng cộng các đối tượng con của đối tượng hiện tại.
- Controls: Chứa các đối tượng con của đối tượng hiện tại.
- Ctl3D: Xác định xem đối tượng có hiển thị ở dạng ba chiều hay hai chiều.
- DefWndProc: Con trỏ trỏ đến hàm mặc định xử lý các sự kiện.
- DockClientCount: Trả về số lượng đối tượng được kết dính vào đối tượng này.
- DockClients: Danh sách các đối tượng được kết dính với đối tượng.
- DockManager:
- Handle: Thẻ quản cửa sổ của đối tượng.
- ParentWindow: Lưu trữ thẻ quản cửa sổ của đối tượng.
- Showing: Trả về giá trị cho biết đối tượng có đang được hiển thị hay không.
- TabOrder: Số thứ tự khi chúng ta nhấn phím Tab để di chuyển chỉ điểm đến đối tượng.
- TabStop: Lưu trữ giá trị cho phép chúng ta di chuyển chỉ điểm đến đối tượng bằng phím Tab hay không.
- WindowHandle: Tương tự như Handle.

Ngoài ra, TWincontrol còn thừa kế một số thuộc tính từ TControl và TComponent.

4.2. Các phương thức chung

- ActionChange: Giống phương thức ActionChange của đối tượng TControl.
- AdjustClientRect: Điều chỉnh kích thước của hình chữ nhật chứa đối tượng.
- AdjustSize: Điều chỉnh kích thước của đối tượng cho khớp với dữ liệu. Phương thức này được gọi một cách tự động nhờ vào thuộc tính AutoSize=true.
- AssignTo: Sao chép các thuộc tính sang một đối tượng khác.
- Broadcast: Gởi một thông điệp đến tất cả các đối tượng con.
- CanFocus: Trả về giá trị xem đối tượng có thể nhận chỉ điểm được hay không.
- CanResize: Trả về giá trị xem đối tượng có thể thay đổi kích cỡ của hay không.
- ChangeScale: Thay đổi tỷ lệ hiển thị của đối tượng.
- ContainsControl: Kiểm tra xem một đối tượng có được chứa trong đối tượng hiện tại hay không.
- ControlAtPos: Trả về đối tượng con tại tọa độ chỉ định.
- CreateHandle: Tạo một thẻ quản cho cửa sổ nằm dưới.

- CreateParams: Tạo một cửa sổ với các tham số tùy chọn, giống như sử dụng hàm CreateWindowEx của Windows API.
- CreateParentedControl: Tạo và gán đối tượng thành đối tượng con của đối tượng hiện tại. Dùng để tạo các đối tượng không trực quan.
- CreateSubClass: Tạo một đối tượng cửa sổ phái sinh từ một lớp sẵn có.
- CreateWindowHandle: Tạo thẻ quản cửa sổ.
- CreateWnd: Tạo thẻ quản cửa sổ.
- DestroyHandle: Huỷ thẻ quản của đối tượng nhưng không huỷ đối tượng.
- DockDrop: Phát sinh sự kiện OnDockDrop.
- DoDockOver: Phát sinh sự kiện DoDockOver.
- DoEnter: Phát sinh sự kiện OnEnter.
- DoExit: Phát sinh sự kiện OnExit.
- DoKeyDown: Phát sinh sự kiện KeyDown.
- DoKeyPress: Phát sinh sự kiện KeyPress.
- DoKeyUp: Phát sinh sự kiện KeyUp.
- DoUnDock: Phát sinh sự kiện OnUnDock.
- FindChildControl: Tìm kiếm đối tượng con bằng cách chỉ định tên của nó.
- FindNextControl: Tìm kiếm đối tượng con kế tiếp của hộp điều khiển bằng phím tab.
- FixupTabList: Phân loại đối tượng con bằng phím tab.
- FlipChildren: Đảo ngược vị trí của đối tượng con. Nghĩa là chúng ta có thể di chuyển đối tượng con trong hộp điều khiển từ vị trí bên trái sang vị trí phải và ngược lại và chúng ta cũng có thể điều chỉnh, canh lề,...
- Focused: xác định xem đối tượng đã được nhập vào chưa.
- GetActionLinkClass: trở về hành động được kết hợp với lớp liên kết.
- GetChildren: kêu gọi một phương thức cụ thể cho mỗi đối tượng con trong hộp điều khiển.
- GetClientOrigin: Trả về giá trị của thuộc tính ClientOrigin.
- GetClientRect: Trả về giá trị của thuộc tính ClientRect.
- GetControlExtents: trả về hình chữ nhật nhỏ nhất mà khít với đối tượng con trong hộp điều khiển.
- GetDeviceContext: Cung cấp một thiết bị cho hộp điều khiển.
- GetParentHandle: Trả về thẻ quản của cửa sổ điều khiển cho cửa sổ điều khiển cha trong hộp điều khiển.
- GetTabOrderList: xây dựng một danh sách trong hộp điều khiển bằng phím tab.
- GetTopParentHandle: trả về cửa sổ điều khiển của cửa sổ đầu tiên mà không liên quan đến hộp điều khiển VCL.
- HandleAllocated: chỉ rõ giá trị của bộ điều khiển.
- HandleNeeded: tạo một đối tượng trong hộp điều khiển nếu đối tượng đó không tồn tại.
- InsertControl: Chèn một hộp điều khiển vào thuộc tính đã được dàn trận.
- Invalidate: Một hộp điều khiển cần được điều chỉnh lại.
- IsControlMouseMsg: dấu hiệu cho biết chuột đã sẵn sàng trong đối tượng con của cửa sổ điều khiển.
- KeyDown: phản ứng lại những sự kiện đã được nhấn phím.
- KeyPress: đáp ứng lại những dữ liệu nhập vào từ bàn phím.
- KeyUp: đáp ứng lại những gì mà phím thả ra.
- MainWndProc: nhận những thông điệp của cửa sổ cho hộp điều khiển.
- NotifyControls: gọi thông điệp đến tất cả các hộp điều khiển con.
- PaintControls: mô tả các đối tượng con trong cửa sổ điều khiển bằng việc sử dụng những thiết bị cụ thể.
- PaintHandler: đáp ứng lại những thông điệp từ WM_PAINT.
- PaintTo: Vẽ đối tượng lên thiết bị như vẽ biểu đồ hoặc vẽ hình ảnh,...
- PaintWindow: hoàn trả lại hình ảnh của cửa sổ điều khiển.
- PaletteChanged: đáp ứng lại sự thay đổi trong hệ thống bảng màu bằng việc nhận biết màu của bảng điều khiển và màu của mỗi đối tượng con.

- ReadState: Sẵn sàng kiểm soát các thuộc tính gắn với giá trị dòng.
 - RecreateWnd: Tạo lại đối tượng Windows ở phía dưới màn hình.
 - ReloadDockedControl: nạp lại một bộ điều khiển neo.
 - RemoveControl: Gỡ bỏ một điều khiển cụ thể từ mảng điều khiển.
 - Repaint: Vẽ lại toàn bộ bộ điều khiển.
 - ResetIme: Khôi phục lại phương pháp soạn thảo dữ liệu (input method editor (IME)) đã hoạt động khi ứng dụng bắt đầu làm việc.
 - ResetImeComposition: quản lý thành phần cấu tạo của phương pháp soạn thảo dữ liệu (input method editor (IME)) để thực hiện một hành động cụ thể.
 - ScaleBy: Thay đổi tỉ lệ điều khiển và đối tượng con.
 - ScaleControls: Chỉ thay đổi tỉ lệ của đối tượng con.
 - ScrollBy: Cuộn nội dung.
 - SelectFirst: Định chỉ điểm đến đối tượng con đầu tiên tính theo giá trị thuộc tính TabOrder.
 - SelectNext: Di chuyển chỉ điểm đến đối tượng con kế tiếp theo thứ tự TabOrder.
 - SetBounds: Cài đặt biên cho đối tượng.
 - SetFocus: Cài đặt chỉ điểm đến đối tượng.
 - SetZOrder: Điều chỉnh điều khiển đến đầu hoặc cuối trong thứ tự Z-Order.
 - ShowControl: Cài đặt một điều khiển hiển thị.
 - Update: Ép đối tượng phải cập nhật lại như vẽ lại đối tượng, ...
 - WndProc: Con trỏ của hàm xử lý thông điệp được gởi đến đối tượng bởi Windows.
- Ngoài ra, còn một số phương thức được phái sinh từ lớp TControl, TPersistent và TObject. Chúng ta có thể tra thêm hướng dẫn của C++ Builder để tìm hiểu kỹ hơn.

4.3. Các sự kiện chung

- OnDockDrop: Phát sinh khi điều khiển khác kết dính vào điều khiển này.
- OnDockOver: Phát sinh khi một điều khiển được kéo di chuyển trên bề mặt điều khiển.
- OnEnter: Phát sinh khi điều khiển nhận được chỉ điểm.
- OnExit: Phát sinh khi điều khiển mất chỉ điểm.
- OnKeyDown: Phát sinh khi người sử dụng nhấn phím.
- OnKeyPress: Phát sinh khi ký tự hiển thị trên điều khiển.
- OnKeyUp: Phát sinh người sử dụng nhả phím.
- OnUnDock: Phát sinh khi huỷ bỏ neo đối tượng.

5. Các đối tượng, thành phần chuẩn của VCL.

Trong mục này, chúng ta sẽ tìm hiểu tóm lược về các thành phần đặc trưng trong môi trường Windows.

5.1. Nút nhấn (Button)

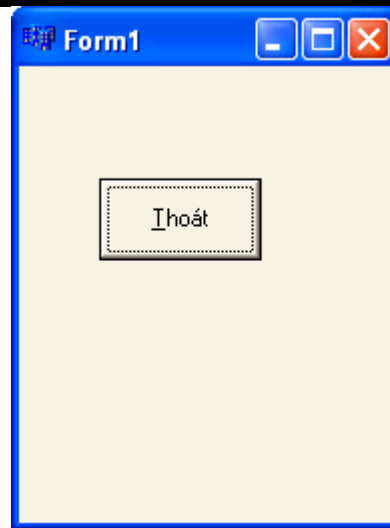


Nút nhấn hoạt động rất đơn giản nhưng lại là thành phần rất quan trọng trong các giao diện sử dụng đồ họa. Nút nhấn được sử dụng với nhiều mục đích khác nhau như các nút trên thanh công cụ, trên thanh tác vụ (taskbar), nút nhấn Start của Windows, hay kết hợp để tạo nên một công cụ mới như thanh trượt (Scrollbar), bao gồm nút nhấn của hai đầu có dạng hình mũi tên.

Nút nhấn được C++ Builder thiết kế với lớp TButton. Sự kiện thường sử dụng nhất của nút nhấn là OnClick. Nút nhấn rất đơn giản, chúng ta không thể thay đổi màu nền, màu chữ cho nút nhấn cũng như thêm hình ảnh vào nút nhấn. Tuy nhiên, chúng ta có thể thay đổi tên kiểu chữ và kích cỡ chữ cho nút nhấn. Để có thể thay đổi được màu nền và màu chữ, chúng ta phải sử dụng các đối tượng thuộc lớp TspeedButton, TbitButton.

Thuộc tính thường dùng của đối tượng nút nhấn như: Caption (văn bản hiển thị trên đối tượng).

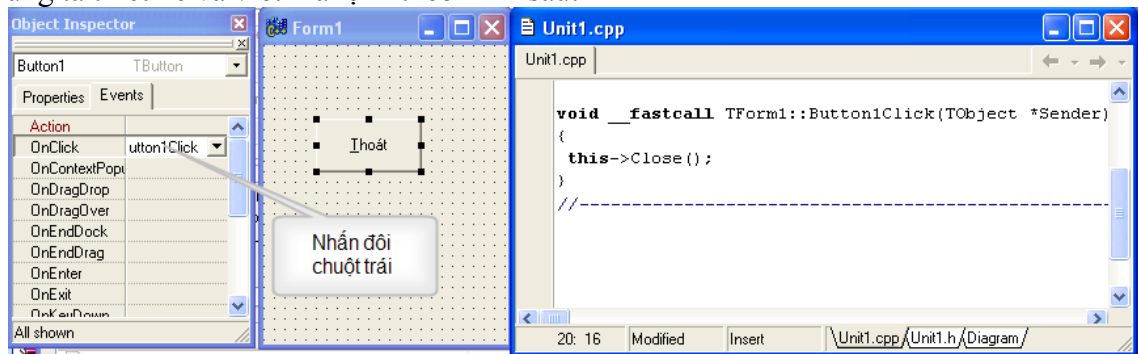
Ví dụ sau, cho thấy cách sử dụng đối tượng nút nhấn trong một chương trình theo hình minh họa sau:



Hình 14-Kết quả chương trình nút nhấn

Khi chúng ta nhấn nút Thoát, chương trình sẽ thoát.

Chúng ta thiết kế và viết mã lệnh theo hình sau:



Hình 15- Thiết kế chương trình

5.2. Nhãn (Label)

A Nhãn là thành phần đơn giản nhất trong thư viện VCL. Đối tượng nhãn chỉ dùng để trình bày một chuỗi văn bản thông thường, nhằm mục đích mô tả thêm thông tin cho các đối tượng. Đối tượng văn bản vẫn là cách đơn giản để đưa kết quả hiển thị ra màn hình dưới dạng một chuỗi văn bản.

Các thuộc tính thường dùng của đối tượng nhãn:

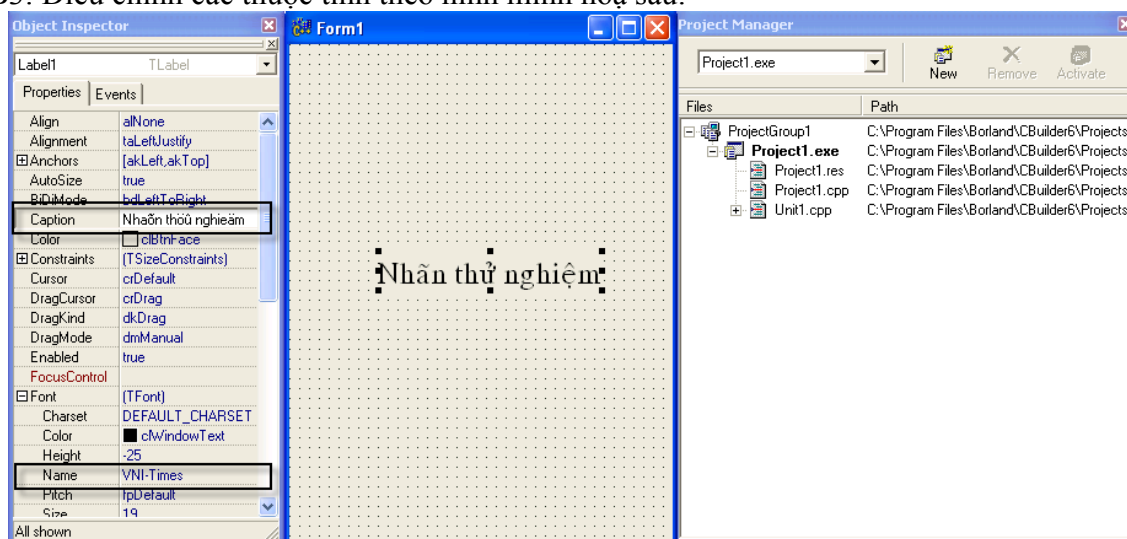
Thuộc tính	Ý nghĩa
Caption	Văn bản hiển thị trên nhãn
Font	Định kiểu chữ, kích cỡ chữ, ...
Transparent	Cho phép màu nền trong suốt hay không

Ví dụ: Xây dựng chương trình cho kết quả như sau:



Hình 16- Kết quả chương trình Label

- B1: Chúng ta tạo ra một dự án mới, chọn thực đơn File/New/Application.
- B2: Đặt một đối tượng Label lên trên biểu mẫu.
- B3: Điều chỉnh các thuộc tính theo hình minh họa sau:



Hình 17-Thiết kế chương trình nhấn

- B4: Nhấn phím F9 để chạy chương trình.

Trong ví dụ đầu tiên này, chúng ta sẽ tiến hành phân tích kỹ hơn ở ví dụ này:

Trước tiên, chúng ta xét mã lệnh được sinh ra tự động với tập tin có phần mở rộng .cpp (cụ thể trong ví dụ này là Project1.cpp)

```
//-----
#include <vcl.h>
#pragma hdrstop
//-----
USEFORM("Unit1.cpp", Form1);
//-----
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
        Application->Initialize();
        Application->CreateForm(__classid(TForm1), &Form1);
        Application->Run();
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
}
```



```

        catch (...)
        {
            try
            {
                throw Exception("");
            }
            catch (Exception &exception)
            {
                Application->ShowException(&exception);
            }
        }
    }
    return 0;
}

```

Kế tiếp, chúng ta khảo sát thêm mã lệnh của biểu mẫu được khai báo với tên Form1, biểu mẫu này có kiểu dữ liệu là lớp TForm1. Tập tin sau là tập tin mã lệnh của Unit1, tập tin này mang tên Unit1.cpp

```

//-----

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}

```

Kế tiếp chúng ta xem xét tập tin .h (tập tin header đã đề cập ở trên) của Unit1.cpp , tập tin này mang tên Unit1.h:

```

//-----

#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
__published:      // IDE-managed Components
    TLabel *Label1;
private:          // User declarations
public:            // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;


```

```
//-----  
#endif
```

Ở tập tin này cho thấy lớp TForm1 có nguồn gốc là lớp TForm, điều này có nghĩa là mọi thành phần trong lớp của TForm đều được chuyển sang TForm1, những thành phần được thêm vào khi thiết kế sẽ được thêm vào TForm1. Chẳng hạn đối tượng mang tên Label1 có kiểu TLabel được khai báo trong quá trình thiết kế Form.

Chú ý: Trong quá trình thiết kế, chúng ta chỉ nên sửa đổi những đoạn mã lệnh mà chúng ta thêm vào, không nên sửa đổi những đoạn mã lệnh mà C++ Builder đã tự sinh ra. Việc làm này cực kỳ nguy hiểm, có khả năng dẫn đến trình biên dịch sẽ không hiểu những gì mà nó đã sinh ra mà bị bạn thay đổi. Việc làm này chỉ có thể thực hiện nếu như chúng ta đã là một lập trình viên cao cấp hoặc chỉ sửa đổi trong trường hợp thật sự cần thiết.

5.3. Ô đánh dấu (Checkbox)

 Ô đánh dấu cho phép chúng ta chọn hoặc bỏ một yêu cầu nào đó bằng cách thể hiện một trong hai trạng thái: được đánh chéo hoặc không đánh chéo. Ô đánh dấu thường thể hiện trạng thái tắt-mở (on/off), có/không (yes/no), hoặc kết hợp nhiều tùy chọn khác nhau.

Ô đánh dấu thuộc lớp TCheckBox.

Thuộc tính thường sử dụng nhất của ô đánh dấu là thuộc tính Checked. Nếu Checked=false, đối tượng ô đánh dấu không được đánh dấu; Checked=true, đối tượng được đánh chéo. Mỗi khi trạng thái của đối tượng đánh dấu bị đổi trạng thái từ đánh chéo sang không đánh chéo hoặc ngược lại, sự kiện OnClick sẽ được phát sinh.

Ví dụ sau bao gồm một nhãn và ba ô đánh dấu, mỗi khi một nút nhấn được đánh chéo, dòng văn bản trong nhãn sẽ đổi theo kiểu dáng chữ được đánh chéo. Kết quả minh họa như hình sau:



Hình 18- Chương trình ô đánh dấu

Trong ví dụ này, chúng ta thiết kế chương trình theo ưu thế của C++ Builder. Chúng ta sẽ thiết kế sự kiện OnClick cho ba đối tượng ô đánh chéo cùng một đoạn mã lệnh.

Cửa sổ thiết kế biểu mẫu như sau:



Hình 19- Thiết kế chương trình ô đánh dấu

Bước 1: Tạo biểu mẫu với các đối tượng liệt kê từ trên xuống dưới như sau:

Đối tượng	Thuộc tính	Giá trị
TLabel	Name	lblDongChu
	Caption	Kiểu dáng chữ
	Font->Name	Tahoma
	Font->Size	18
	Font->Color	clRed
TCheckbox	Name	chkdam
	Caption	Chữ đậm
	Font->Name	Tahoma
	Font->Size	14
TCheckbox	Name	chknghieng
	Caption	Chữ nghiêng
	Font->Name	Tahoma
	Font->Size	14
TCheckbox	Name	chkgachchan
	Caption	Chữ gạch chân
	Font->Name	Tahoma
	Font->Size	14

Bước 2: Nhấn đôi chuột lên đối tượng chkdam để mở cửa sổ viết mã lệnh cho sự kiện OnClick, ta nhận được mã lệnh cho sự kiện này như sau:

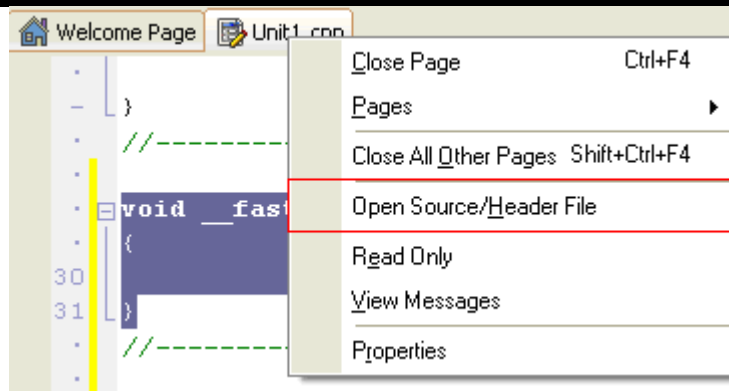
```
void __fastcall TForm1::chkdamClick(TObject *Sender)
{
}

```

Tiến hành điều chỉnh phương thức này từ tên chkdamClick thành checkboxclick nhằm sử dụng chung cho 3 đối tượng theo bước 3.

Bước 3: Đổi tên phương thức chkdamClick thành checkboxclick, viết mã lệnh:

- Điều chỉnh phương thức chkdamClick ở trên thành checkboxclick, công đoạn này là công đoạn nhằm chuẩn bị viết mã lệnh. Tuy nhiên, C++ cung cấp hai tập tin .h và .cpp, trong đó tập tin .cpp chứa mã nguồn, tập tin .h chứa khai báo. Việc sửa đổi ở trên chỉ là thay đổi ở tập tin mã nguồn, chúng ta cần phải thay đổi ở tập tin khai báo cho đồng bộ.
- Nhấn chuột phải lên Tab tập tin soạn thảo, chọn Open Source/Header File như hình sau:



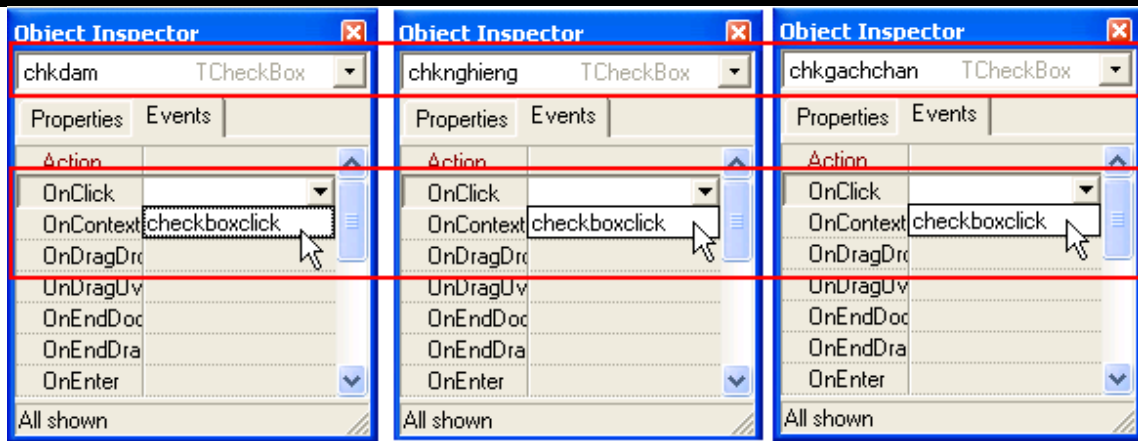
Hình 20-Mở cửa sổ tập tin khai báo

- Ta sẽ nhìn thấy trong ngăn __published dòng dưới và chkdamClick thành checkboxclick.
- Nhấn chuột phải lên Tab tập tin soạn thảo, chọn Open Source/Header File để trở về cửa sổ soạn thảo mã lệnh. Sau đó, thêm đoạn mã lệnh để đạt được kết quả sau:

```
void __fastcall TForm1::checkboxclick(TObject* Sender)
```

```
{
    //TODO: Add your source code here
    TFontStyles style;
    style=this->lblDongChu->Font->Style;
    if (Sender==chkDam)
    {
        if (this->chkDam->Checked)
            style= style << fsBold;
        else
            style = style >> fsBold;
    }
    else if (Sender==chkNghieng)
    {
        if (this->chkNghieng->Checked)
            style = style << fsItalic;
        else
            style = style >> fsItalic;
    }
    else
    {
        if (this->chkGachChan->Checked)
            style = style << fsUnderline;
        else
            style = style >> fsUnderline;
    }
    this->lblDongChu->Font->Style = style;
}
```

Bước 4: Chúng ta gắn phương thức trên với sự kiện OnClick của từng đối tượng ô đánh dấu ở trên như hình minh họa sau:



Hình 21-Gắn phương thức cho sự kiện

- Bước 5: Nhấn phím F9 để chạy chương trình.

Trong đoạn mã lệnh trên, chúng ta chú ý về một kiểu dữ liệu mới được cung cấp trong C++ Builder, đó là kiểu dữ liệu tập hợp (Set), đây là kiểu dữ liệu tương thích với C++ Builder. Để thêm một phần tử vào tập hợp, chúng ta dùng phép toán <<; để loại bỏ một phần tử khỏi tập hợp, chúng ta dùng phép toán >>.

Đoạn mã lệnh trên thực hiện so sánh xem đối tượng phát sinh sự kiện với từng đối tượng (Sender==chkdam, Sender = chknghieng, ...) và kiểm tra thuộc tính Checked để cài đặt giá trị kiểu dáng chữ cho nhãn lbldongchu.

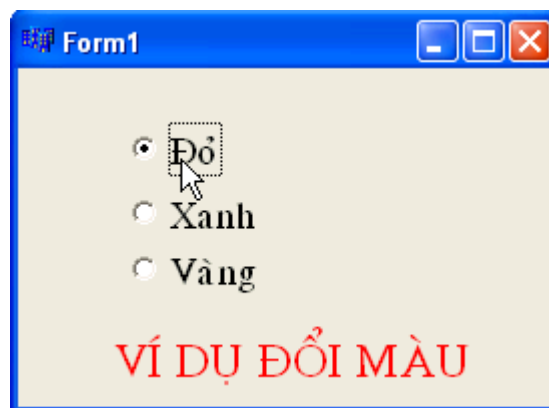
5.4. Ô chọn (RadioButton)

Tương tự như đối tượng ô đánh dấu nhưng đối tượng ô chọn thường đi chung với nhau thành một nhóm và mỗi một thời điểm chỉ có một ô chọn được chọn, không như ô đánh dấu, mỗi một thời điểm có nhiều ô đánh dấu được đánh dấu. Chúng ta có thể tưởng tượng như một nút rà dài trên máy thu thanh, mỗi lần chỉ bắt và nghe được một đài duy nhất trong số nhiều đài mà thôi.

Ô chọn có kiểu dữ liệu là lớp TRadioButton.

Thuộc tính thường sử dụng nhất của ô chọn là Checked, nếu Checked=true thì ô chọn được chọn ngược lại thì ô chọn không được chọn.

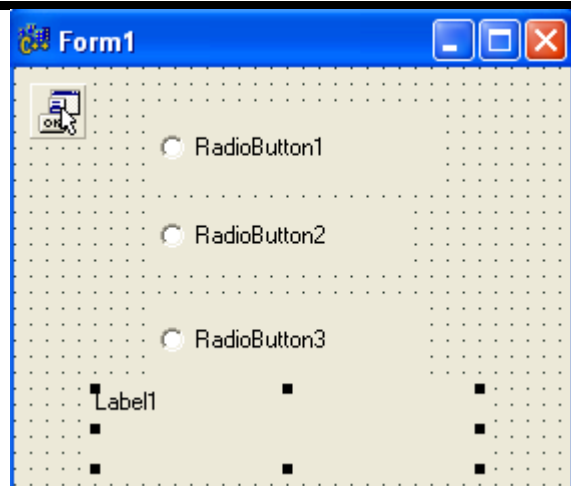
Ví dụ sau cho thấy cách dùng các đối tượng ô chọn kết hợp với việc dùng đối tượng thuộc lớp TActionList để dùng cho sự OnClick. Đối tượng ActionList là đối tượng không trực quan, nó chứa các hành động cho phép sử dụng bởi các thành phần và đối tượng. Chương trình chúng ta có kết quả là thay đổi màu sắc cho nhãn đặt trên biểu mẫu. Mỗi lần chỉ được chọn một màu bằng cách sử dụng ô chọn.



Hình 22-Kết quả chương trình Radio

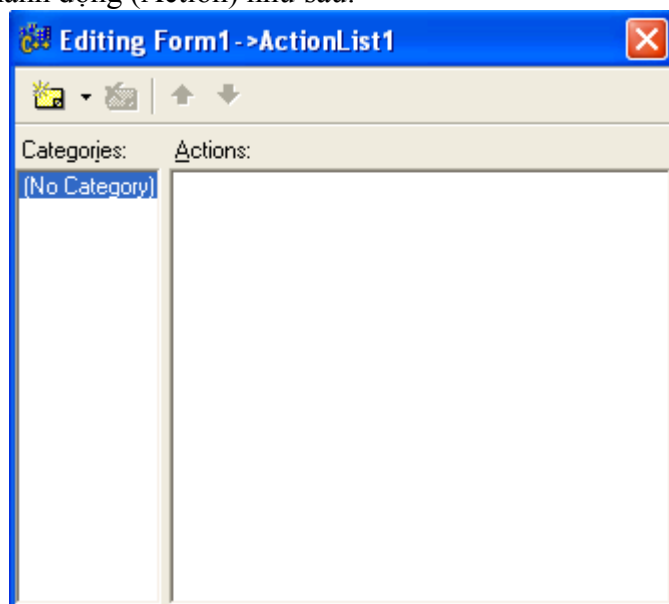
Chúng ta tiến hành các bước thiết kế chương trình trên như sau:

Bước 1: Chúng ta thực hiện thiết kế Form như sau:



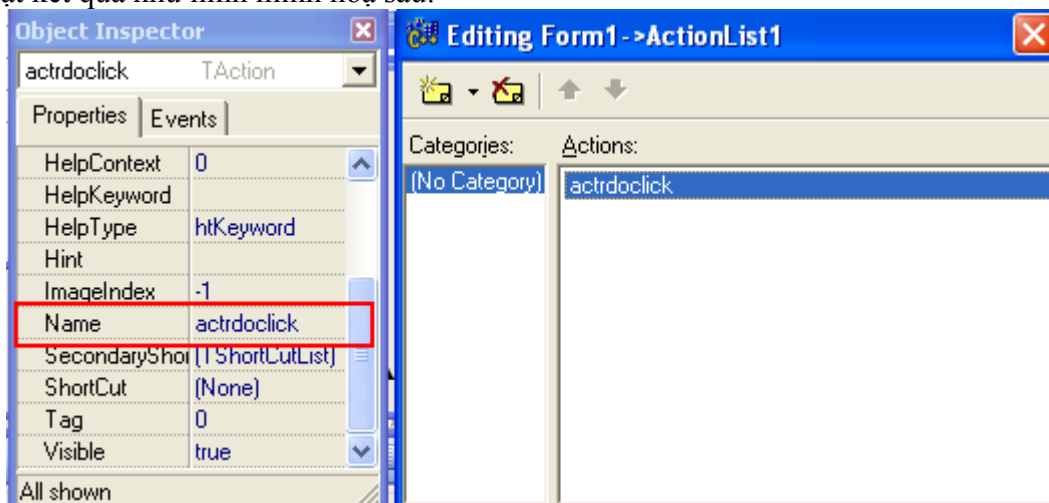
Hình 23- Cửa sổ thiết kế chương trình Radio

Bước 2- Nhấn đôi chuột trái lên đối tượng ActionList1 (🔗), chúng ta sẽ nhận được cửa sổ thiết kế các đối tượng hành động (Action) như sau:



Hình 24-Cửa sổ thiết kế ActionList

Bước 3 - Nhấn nút màu vàng trên góc trái để thêm một hành động mới, đặt tên cho hành động này để đạt kết quả như hình minh họa sau:



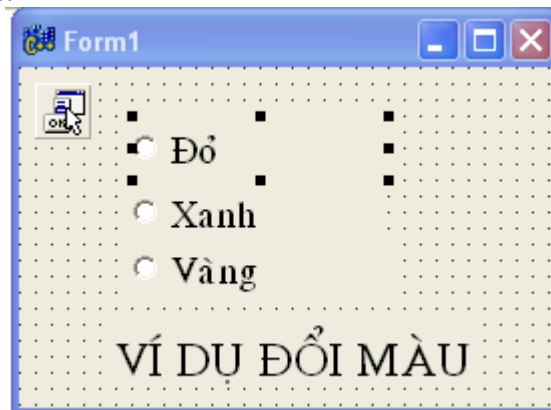
Hình 25-Thiết kế đối tượng actrdclick

- Bước 4: Gắn kết các đối tượng ô chọn với đối tượng actrdclick thông qua thuộc tính Action.

- Bước 5: Điều chỉnh các thuộc tính theo bảng liệt kê các đối tượng từ trên xuống dưới như sau (Chú ý, các thuộc tính Caption được liệt kê giá trị theo bảng mã VNI, bởi C++ Builder 2006 trở về trước vẫn chưa hỗ trợ Unicode):

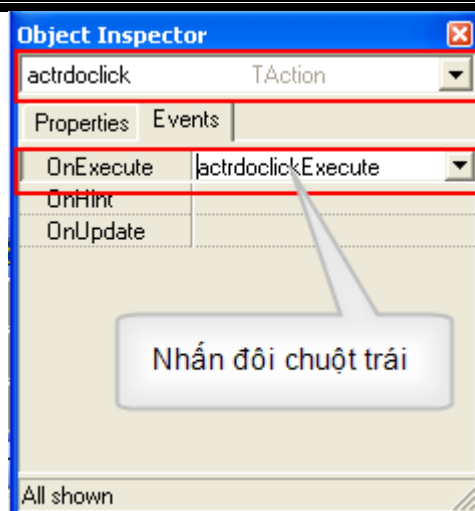
Tên đối tượng	Thuộc tính	Giá trị
ActionList1		
lblDongChu	Font->Size	18
	Font->Color	clRed
rdoDo	Font->Name	Tahoma
	Font->Size	14
	Caption	Đỏ
rdoXanh	Font->Name	Tahoma
	Font->Size	14
	Caption	Xanh
rdoVang	Font->Name	Tahoma
	Font->Size	14
	Caption	Đỏ

Kết quả đạt được như sau:



Hình 26-Thiết kế biểu mẫu cho chương trình radio

- Bước 6: Viết mã lệnh cho sự kiện OnExecute theo hình minh họa và mã lệnh được liệt kê sau đây:



Hình 27-Viết mã lệnh cho actrdclick


Mã lệnh:

```
if (this->rdoDo->Checked)
    this->lblDongChu->Font->Color = clRed;
else
    if (this->rdoVang->Checked)
        this->lblDongChu->Font->Color = clYellow;
    else
        this->lblDongChu->Font->Color = clBlue;
```

Trong đoạn mã lệnh trên, chúng ta có sử dụng các hằng clRed (màu đỏ), clYellow (màu vàng), clBlue (màu xanh) và một từ khoá this, từ khoá này chỉ định rằng chúng ta đang chỉ định lớp hiện tại (tức là biểu mẫu đang thiết kế). Chú ý rằng trong đoạn mã lệnh trên chúng ta đã tính lượt việc so sánh Sender với các đối tượng rdodo, rdoxanh và rdovang giống như cách làm của chúng ta trong bài ô đánh dấu ở trên, bởi một lẽ dễ hiểu, chúng ta đã chú ý viết đoạn mã lệnh trên chỉ để sử dụng cho ba đối tượng rdodo, rdoxanh, rdovang cũng như vào một thời điểm chỉ có thể chọn đúng một ô chọn mà thôi.

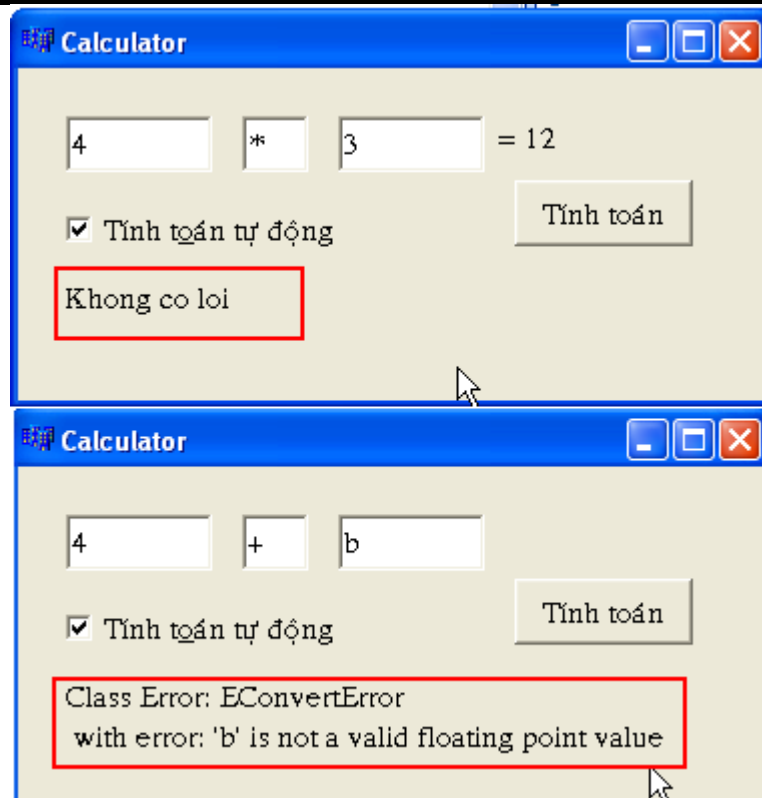
- Bước 7: Nhấn phím F9 để chạy chương trình để đạt được kết quả như yêu cầu.

5.5. Ô văn bản

 Ô văn bản thuộc lớp TEdit, đây là thành phần nhập liệu đơn giản nhất dùng giao diện đồ họa. Chúng ta được phép nhập vào một chuỗi văn bản, chỉnh sửa, di chuyển con nháy để soạn thảo bằng các phím mũi tên. Đối tượng này chỉ cho phép nhập một dòng văn bản đơn.

Thuộc tính thường sử dụng nhất của đối tượng ô văn bản là thuộc tính Text. Nó chứa đựng dòng văn bản hiển thị trên đối tượng cũng như văn bản người dùng nhập vào ô văn bản. Mỗi khi văn bản trên ô văn bản bị thay đổi, sự kiện OnChange sẽ được phát sinh. Ngoài ra, chúng ta còn có thể thay đổi màu nền, màu chữ và kiểu chữ. Chúng ta cũng có thể gán giá trị cho thuộc tính Enabled = false để cấm nhập văn bản vào ô văn bản, lúc này ô văn bản sẽ ở trạng thái chỉ đọc.

Dưới đây là một ví dụ minh họa việc sử dụng các đối tượng ô văn bản, trong ví dụ này chúng ta sẽ sử dụng một phương thức dùng chung cho cả ba đối tượng ô nhập văn bản như cách làm của ví dụ ô đánh dấu ở trên.

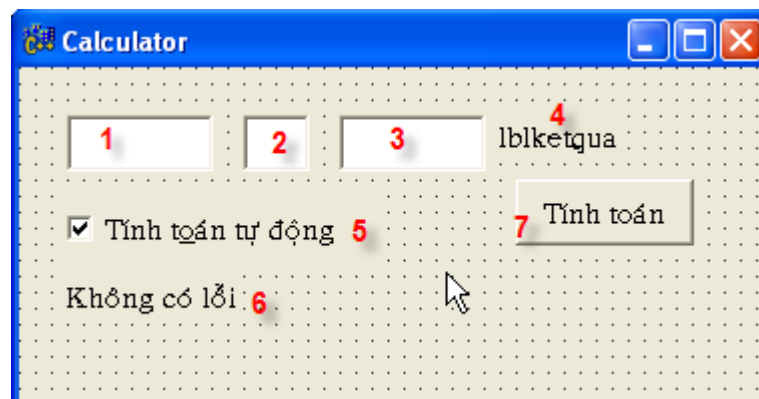


Hình 28-Kết quả chương trình edit

Chú ý rằng, trong chương trình trên chúng ta có truy bắt lỗi một cách tự động. Nút nhấn có nhiệm vụ tính toán để ra được kết quả, còn ô đánh dấu nếu được đánh dấu, chúng ta sẽ truy bắt sự kiện OnChange để tính toán một cách tự động.

Chúng ta bắt tay và thiết kế để đạt được chương trình trên theo các bước sau:

Bước 1: Thiết kế để đạt được biểu mẫu sau (các chữ số dùng để đánh dấu, không phải thiết kế các chữ số):



Hình 29-Cửa sổ thiết kế chương trình Edit

Bảng sau liệt kê thuộc tính và đối tượng theo số đã đánh dấu trong hình:

Số đánh dấu	Đối tượng	Thuộc tính	Giá trị
1	Edit	Name	edtSo1
		Font->Size	11
		Font->Name	Tahoma
2	Edit	Name	edtPhepToan

		Font->Size	11
		Font->Name	Tahoma
3	Edit	Name	edtSo2
		Font->Size	11
		Font->Name	Tahoma
4	Label	Name	lblKetQua
		Font->Size	11
		Font->Name	Tahoma
5	CheckBox	Name	chkTinhToanTuDong
		Font->Size	11
		Font->Name	Tahoma
		Checked	true
6	Label	Name	lblLoi
		Font->Size	11
		Font->Name	Tahoma
7	Button	Name	btnTinhToan
		Font->Size	11
		Font->Name	Tahoma

- Bước 2: Chúng ta viết mã lệnh cho sự kiện OnClick của nút nhấn btntinhtoan để đạt được đoạn mã lệnh sau:

```
void __fastcall TForm1::btnTinhToanClick(TObject *Sender)
{
    float so1, so2;
    float ketqua;
    this->lblLoi->Caption = L"Không có lỗi";
    this->lblKetQua->Caption = "";
    try {
        if (this->edtSo1->Text.Trim() == "")
            so1 = 0;
        else
            so1 = StrToFloat(this->edtSo1->Text.Trim());
        if (this->edtSo2->Text.Trim() == "")
            so2 = 0;
        else
            so2 = StrToFloat(this->edtSo2->Text);
    }
    catch(Exception & E) {
        this->lblLoi->Caption = "Class Error: " + E.ClassName()
            + "\n with error: " + E.Message;
    }
}
```

```

        return;
    }
    ketqua = 0;
    try {
        if (this->edtPhepToan->Text.Trim() == "+")
            ketqua = so1 + so2;
        else if (this->edtPhepToan->Text.Trim() == "-")
            ketqua = so1 - so2;
        else if (this->edtPhepToan->Text.Trim() == "*")
            ketqua = so1 * so2;
        else if (this->edtPhepToan->Text.Trim() == "/")
            if (so2 == 0) {
                this->lblLoi->Caption = "Khong chia duoc. So chia bang 0";
                return;
            }
            else
                ketqua = so1 / so2;
        else {
            this->lblLoi->Caption = "Phep toan bi sai";
            return;
        }
        this->lblKetQua->Caption = "=" + FloatToStr(ketqua);
    }
    catch(Exception & E) // truy bắt lỗi
    {
        this->lblLoi->Caption = "Class Error: " + E.ClassName()
            + "\n with error: " + E.Message;
    }
}

```

- Bước 3: Tạo phương thức editchange để gắn kết với sự kiện OnChange ba đối tượng edtSo1, edtPhepToan, edtSo2 nhằm thực hiện kiểm tra dữ liệu nhập vào và tính toán tự động. (Cách thức thực hiện tương tự bài 5.3 ở trên).

Viết mã lệnh để đạt được đoạn mã lệnh sau:

```

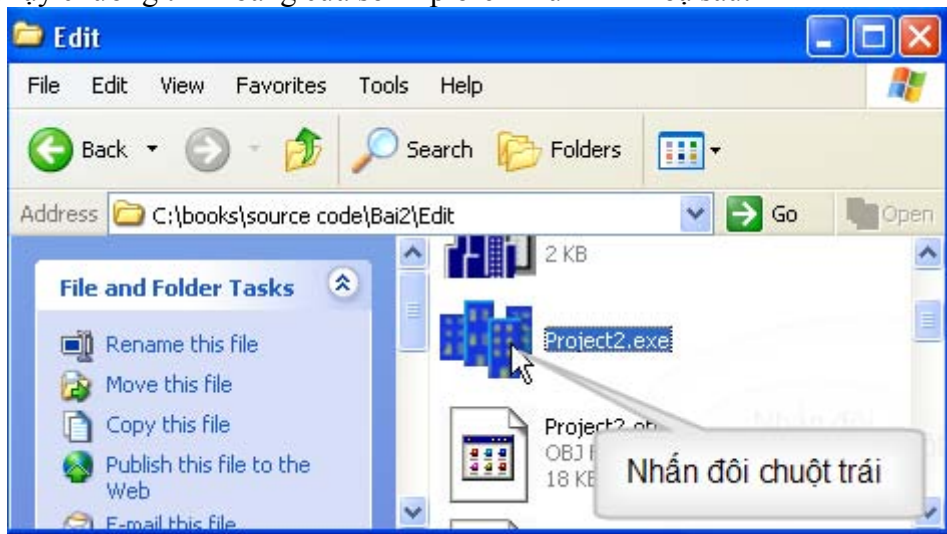
// Kiểm tra dữ liệu trên đối tượng
UnicodeString text;
TEdit *tam;
this->lblLoi->Caption = L"Không có lỗi!";
tam = (TEdit*)Sender;
text = tam->Text.Trim();
if (Sender == edtSo1 || Sender == edtSo2) {
    try {
        StrToFloat(text); // thu chuyen chuoi sang so
    }
    catch(Exception & E) // truong truong hop bi loi
    {
        this->lblLoi->Caption = "Class Error: " + AnsiString(E.ClassName())
            + "\n with error: " + E.Message;
    }
}
else // truong hop la phep toan
    if (text != "+" && text != "-" && text != "*" && text != "/")
        this->lblLoi->Caption = "Phep toan bi sai";
if (!this->chkTinhToanTuDong->Checked)
    return; // nếu không chọn tính tự động thì không tính toán

```

this->btnTinhToanClick(Sender); // gọi sự kiện OnClick của Button.

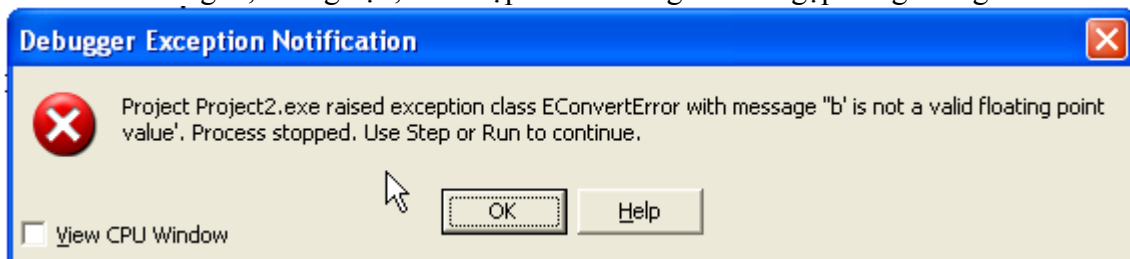
Bước 4: Chọn Project/Build All Projects để biên dịch chương trình ra tập tin .EXE.

Bước 5: Chạy chương trình bằng cửa sổ Explorer như minh họa sau:



Hình 30-Thực thi chương trình Edit

Lý do chính để chạy chương trình trong cửa sổ Explorer nhằm hiển thị truy bắt lỗi, nếu chúng ta chạy chương trình trong môi trường IDE của C++ Builder thì trình bắt lỗi sẽ hoạt động trước trình bắt lỗi của chúng ta, chẳng hạn, khi nhập edtso2 bằng b thì sẽ gặp bảng thông báo sau:



Hình 31-Phát sinh lỗi do Debug

Trong đoạn mã lệnh trên, chúng ta dùng cấu trúc truy bắt ngoại lệ (lỗi phát sinh) trong quá trình thực thi chương trình bằng cấu trúc:

```
try
{
    StrToFloat(text); //thu chuyen chuoai sang so
}
catch (Exception &E) //truong truong hop bi loi
{
    this->lblloi->Caption = "Class Error: " + AnsiString(E.ClassName()) + "\n with error: " + E.Message ;
}
```

Cấu trúc này nói lên rằng:

Chúng ta đang thử chuyển dữ liệu từ chuỗi sang số, chuỗi được lưu trong biến text. Trong trường hợp không thành công (bị lỗi) sẽ gọi đoạn lệnh trong khối catch. Biến E là biến chứa lớp phát sinh ngoại lệ, với tên lớp Exception là tên lớp có cấp cao nhất trong cây kế thừa các ngoại lệ. Để bắt lỗi chính xác hơn, chúng ta phải biết mình đang thao tác với động tác gì và biết được lớp chính xác để truy bắt lỗi. Tham khảo thêm trong cửa sổ hướng dẫn của C++ Builder bằng cách gõ cụm từ khoá “exception classes, VCL,” (không có dấu nháy) vào ngăn Index trong cửa sổ hướng dẫn của C++ Builder .

5.6. Vùng văn bản



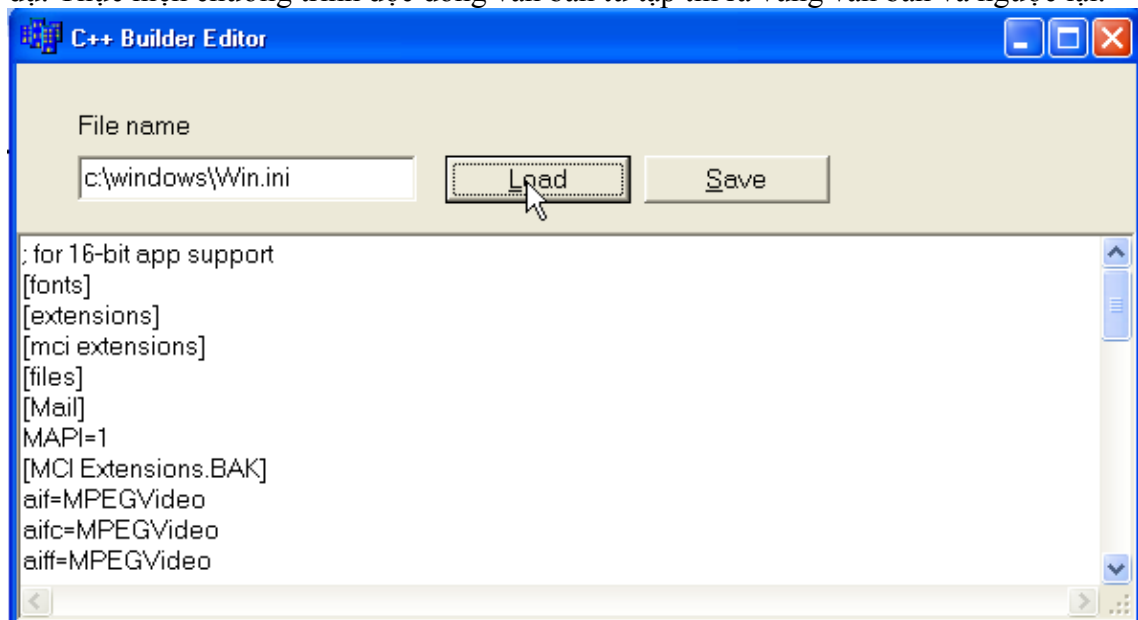
Vùng văn bản (Memo) giải quyết hạn chế của đối tượng ô văn bản (Edit), bởi lẽ ô văn bản chỉ sử dụng một dòng để nhập văn bản trong khi đó Memo cho phép chúng ta nhập nhiều dòng

văn bản. Các dòng văn bản trong vùng văn bản của C++ Builder được lưu trữ trong một thuộc tính tên Lines có kiểu TStrings. Các phương thức và thuộc tính của Lines được liệt kê trong bảng sau:

Thuộc tính	Ý nghĩa
Lines->Add	Thêm vào một dòng mới
Lines->Count	Số dòng trong vùng văn bản
Lines->Clear	Xoá hết tất cả các dòng đang có
Lines->LoadFromFile	Nạp các dòng văn bản từ tập tin
Lines->SaveToFile	Lưu các dòng văn bản vào tập tin

Đối tượng vùng văn bản cũng cung cấp một thuộc tính Text, thuộc tính này chỉ có thể truy cập bằng mã lệnh. Thuộc tính Text chứa tất cả các dòng văn bản gộp lại, mỗi dòng các nhau bởi hai ký tự có mã 10 và 13 (ký tự về đầu dòng và xuống dòng).

Ví dụ: Thực hiện chương trình đọc dòng văn bản từ tập tin ra vùng văn bản và ngược lại.



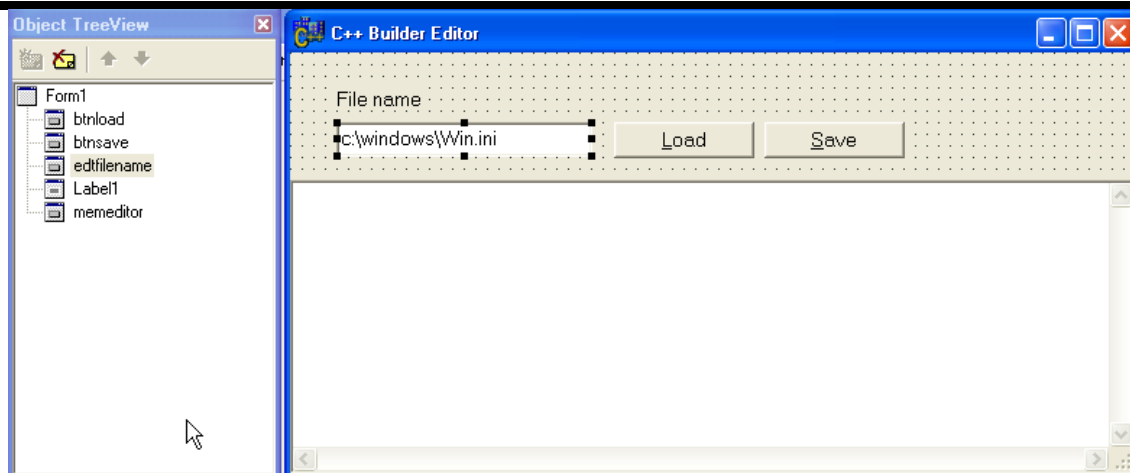
Hình 32-Chương trình Memo

Sự kiện thường sử dụng nhất của vùng văn bản là OnChange, sự kiện này xảy ra mỗi khi vùng văn bản bị sửa đổi hoặc được thêm dữ liệu vào.

Chương trình trên, chúng ta dùng một đối tượng ô văn bản để nhập tên tập tin, hai đối tượng Button để đọc dữ liệu từ tập tin và lưu dữ liệu vào tập tin.

Chúng ta thiết kế chương trình trên với hình minh họa và thuộc tính được liệt kê dưới đây:

B1- Thiết kế biểu mẫu



Hình 33-Thiết kế chương trình Memo

Bảng liệt kê các đối tượng trên biểu mẫu từ trên xuống dưới; từ trái qua phải:

Đối tượng	Tên thuộc tính	Giá trị
Form	Name	Form1
	Caption	C++ Builder Editor
Label	Name	Label1
	Caption	File name
	Font->Size	10
Edit	Name	edtFileName
	Text	C:\Windows\Win.ini
	Font->Size	10
Button	Name	btnLoad
	Caption	&Load
	Font->Size	10
Button	Name	btnSave
	Caption	&Save
	Font->Size	10
Memo	Name	memEditor
	Font->Size	10
	Lines...	""
	Align	alBottom
	Scrollbars	Both

B2- Viết mã lệnh cho sự kiện OnClick của nút btnload như sau:

```
void __fastcall TForm1::btnLoadClick(TObject *Sender)
```

```
{
    this->memEditor->Clear();
    this->memEditor->Lines->LoadFromFile(this->edtFileName->Text);
    this->memEditor->Modified = false; // Đặt trạng thái bị chỉnh sửa là sai
}
```

B3- Viết mã lệnh cho sự OnClick của nút btnsave như sau:

```
if (this->memEditor->Modified)
    this->memEditor->Lines->SaveToFile(this->edtFileName->Text );
```

B4- Nhấn F9 để thực thi chương trình.

Chương trình trên được xây dựng với một vài thuộc tính mới. Chúng ta gặp thuộc tính Anchors, thuộc tính này gọi là neo. Đối tượng memeditor được neo vào góc dưới của form bằng cách điều chỉnh thuộc tính Align bằng alBottom. Thuộc tính ScrollBars sẽ cho phép chúng ta điều chỉnh và lựa chọn thanh cuộn sẽ hiển thị cùng đối tượng vùng văn bản.

5.7. Hộp danh sách



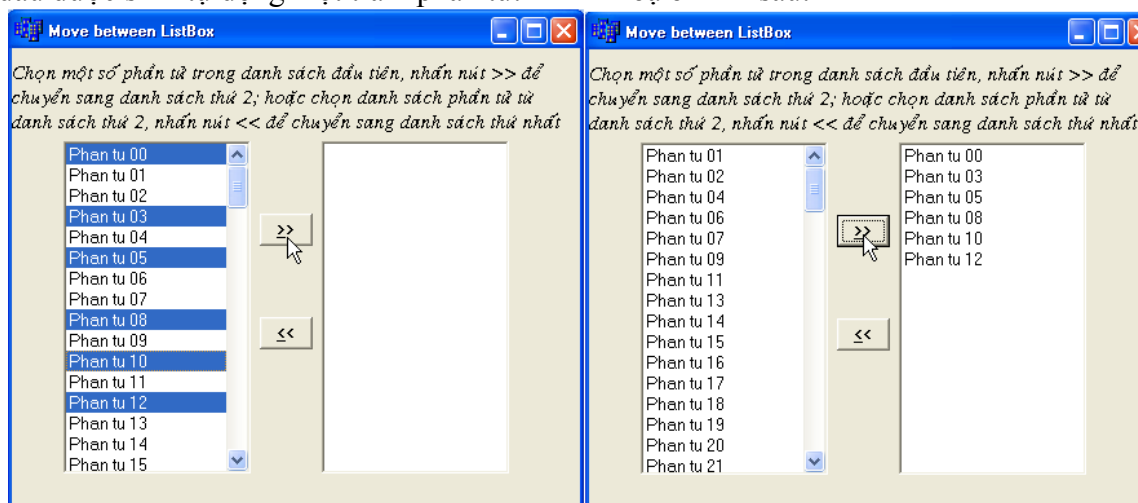
Đối tượng danh sách chọn được C++ Builder thiết kế trong lớp TListBox. Đối tượng này cho phép chúng ta hiển thị một danh sách các phần tử nhằm cho phép người dùng chọn một hay nhiều phần tử từ danh sách.

Các thuộc tính hay dùng của đối tượng danh sách được liệt kê trong bảng sau:

Thuộc tính/phương thức	Ý nghĩa
Items	Chứa danh sách các phần tử
MultiSelect	Cho phép hay không cho phép chọn một lúc nhiều phần tử
Sorted	Các phần tử được hiển thị có sắp xếp hay không
ItemIndex	Chỉ số của phần tử được chọn trong danh sách. Chỉ số phần tử nhỏ nhất là 0. Nếu thuộc tính MultiSelect được chọn thành true thì giá trị mặc định của ItemIndex bằng 0; ngược lại giá trị mặc định của ItemIndex bằng -1, nghĩa là không có phần tử nào được chọn.
Selected[i]	Kiểm tra phần tử có chỉ số i có được chọn hay không. Giá trị trả về là true hay false.
Count	Số phần tử trong danh sách
AddItem	Phương thức cho phép đưa một phần tử là đối tượng vào trong danh sách.
Clear	Xoá hết danh sách phần tử đang hiện có
DeleteSelected	Xoá tất cả các phần tử đang được chọn.
CopySelection	Sao chép phần tử đang được chọn sang một danh sách khác.
MoveSelection	Di chuyển các phần tử đang được chọn sang một danh sách khác.
ItemAtPos	Xác định phần tử tại vị trí con trỏ chuột.
TopIndex	Chỉ số của phần tử ở trên cùng của danh sách.

Các phần tử được lưu trữ trong thuộc tính Items, thuộc tính này sẽ được truy cập và xử lý giống như thuộc tính Lines của đối tượng vùng văn bản ở trên bởi vì chúng có cùng một kiểu dữ liệu là lớp TStringList.

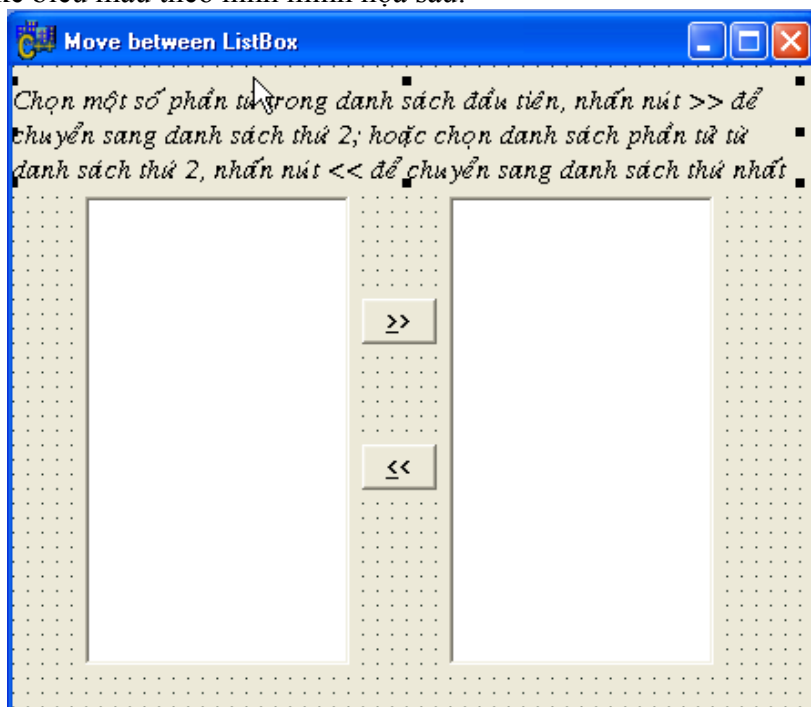
Ví dụ 1: Chương trình sau chứa hai đối tượng danh sách, cho phép chúng ta chọn một số phần tử từ danh sách này và chuyển sang danh sách kia và ngược lại. Các phần tử chứa trong danh sách đầu được sinh tự động một trăm phần tử. Minh họa ở hình sau:



Hình 34-Kết quả chương trình hai ListBox

Chúng ta tiến hành thiết kế chương trình để đạt kết quả như trên:

Bước 1: Thiết kế biểu mẫu theo hình minh họa sau:



Hình 35- Thiết kế biểu mẫu hai ListBox

Bảng sau liệt kê các thuộc tính của các đối tượng từ trên xuống dưới và từ trái sang phải.

Đối tượng	Thuộc tính	Giá trị
Form	Name	frmMain
	Caption	Move between ListBox
Label	Caption	Chọn một số phần tử trong danh sách đầu tiên, nhấn nút >> để

		chuyển sang danh sách thứ 2; hoặc chọn danh sách phần tử từ danh sách thứ 2, nhấn nút << để chuyển sang danh sách thứ nhất.
	Font->Size	10
	Font->Style->fsItalic	true
	Font->Name	Tahoma
	WordWrap	True
ListBox	Name	lstNguon
	Multiselect	true
	Sorted	true
ListBox	Name	lstDich
	Multiselect	true
	Sorted	true
Button	Name	btnMove
	Caption	&>>
Button	Name	btnMoveBack
	Caption	&<<

Bước 2: Viết mã lệnh cho sự kiện OnCreate của đối tượng biểu mẫu để khởi tạo 100 phần tử cho danh sách thứ nhất (lstnguon). Sự kiện này xảy ra khi biểu mẫu được tạo ra trong bộ nhớ mà chưa hiển thị trên màn hình. Mã lệnh như sau:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int i;
    AnsiString t;
    for (i=0;i<=99;i++)
    {
        t= IntToStr(i);
        if (t.Length()<2)
            t="0" + t;
        this->lstNguon->Items->Add(L"Phần tử " + t);
    }
}
```

Bước 3: Viết mã lệnh cho sự kiện của nút btnmove để đạt được đoạn mã lệnh sau:

```
void __fastcall TForm1::btnMoveClick(TObject *Sender)
{
    int i;
    for (i= this->lstNguon->Count-1;i>=0;i--)
        if (this->lstNguon->Selected[i]) //nếu phần tử được chọn
        {
            //thêm giá trị của phần tử được chọn
        }
}
```

```

    this->lstDich->Items->Add(this->lstNgon->Items->Strings[i]);
//xóa phần tử đã thêm vào danh sách lstDich
    this->lstNgon->Items->Delete(i);
}
}

```

Trong đoạn mã lệnh này, chúng ta chú ý chúng ta cho vòng lặp chạy từ các phần tử cuối trở về phần tử đầu, điều này là bắt buộc vì khi chúng ta xóa một phần tử, phần tử ở dưới sẽ tự động dồn lên. Việc làm này nhằm xóa chính xác phần tử được chọn.

Bước 4: Tương tự như ở bước 3, chúng ta sẽ viết mã lệnh cho sự kiện OnClick của nút btnmoveback như sau:

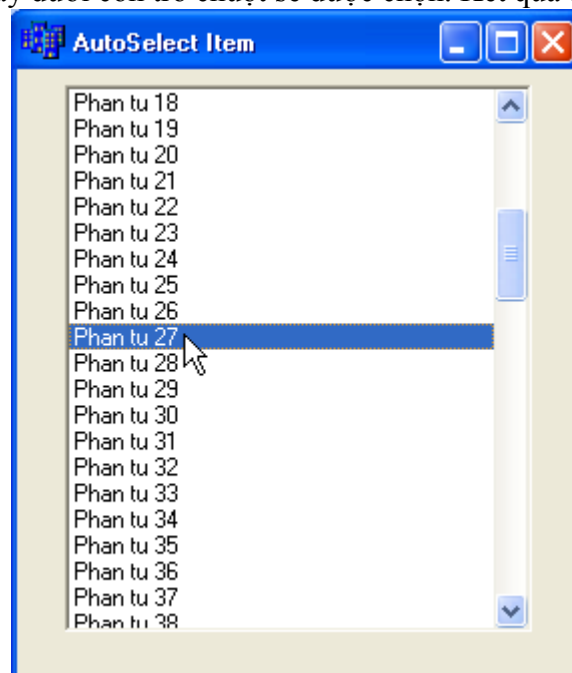
```

void __fastcall TForm1::btnMoveBackClick(TObject *Sender)
{
    int i;
    for (i= this->lstDich->Count-1;i>=0;i--)
        if (this->lstDich->Selected[i])
        {
            this->lstNgon->Items->Add(this->lstDich->Items->Strings[i]);
            this->lstDich->Items->Delete(i);
        }
}

```

Bước 5: Nhấn phím F9 để chạy chương trình..

Ví dụ 2: Chương trình sau sẽ xây dựng một ListBox mà khi chúng ta di chuyển con trỏ chuột trên danh sách, phần tử ngay dưới con trỏ chuột sẽ được chọn. Kết quả được minh họa như sau:



Hình 36-Chương trình AutoSelect

Chúng ta tiến hành thiết kế chương trình theo các bước sau.

Bước 1: Thiết kế biểu mẫu theo bảng liệt kê các thuộc tính được điều chỉnh như sau:

Đối tượng	Thuộc tính	Giá trị
Form	Name	frmMain
	Caption	AutoSelect Item
ListBox	Name	lstOne

Bước 2: Viết mã lệnh cho sự kiện OnCreate của đối tượng TForm1 như sau:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int i;
    AnsiString t;
    for (i=0;i<=99;i++)
    {
        t= IntToStr(i);
        if (t.Length()<2)
            t="0" + t;
        this->lstOne->Items->Add(L"Phần tử " + t);
    }
}
```

Bước 3: Viết mã lệnh MouseMove cho đối tượng lstone như sau:

```
void __fastcall TForm1::lstOneMouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
    int i;
    TPoint p;
    //nếu tọa độ Y của con trỏ chuột bé hơn vị trí trên cùng của danh sách hay
    //lớn hơn vị trí biên dưới của danh sách chúng ta cuộn danh sách lên
    if (Y<this->lstOne->Top)
        this->lstOne->TopIndex = this->lstOne->TopIndex-1;
    else if(Y>this->lstOne->Top+ this->lstOne->Height)
        this->lstOne->TopIndex = this->lstOne->TopIndex-1;
    //xác định chỉ số phần tử dưới tọa độ con chuột
    p = TPoint(X,Y); //Khởi tạo biến p xác định tọa độ con trỏ chuột
    i = this->lstOne->ItemAtPos(p,true);
    this->lstOne->ItemIndex = i;
}
```

Bước 4: Nhấn phím F9 để chạy chương trình.

5.8. Hộp danh sách đổ xuống



Hộp danh sách đổ xuống (ComboBox) được C++ Builder đặt trong lớp TComboBox. Đối tượng này là sự kết hợp của ô văn bản (Edit) và hộp danh sách (ListBox). Nghĩa là, đối với loại danh sách này chúng ta có thể soạn thảo hoặc chọn một phần tử từ danh sách. Mặc dù, hộp danh sách đổ xuống là sự kết hợp của hộp văn bản và hộp danh sách nhưng danh sách đổ xuống không cho phép chọn nhiều phần tử cùng một lúc.

Thuộc tính thường dùng nhất của hộp danh sách đổ xuống được liệt kê trong bảng sau:

Thuộc tính	Ý nghĩa
Text	Nội dung của văn bản được chọn
Items	Danh sách các phần tử
ItemIndex	Chỉ số của phần tử được chọn hiện tại
Sorted	Sắp xếp các phần tử
Style	Chọn các kiểu hiển thị hộp danh sách đổ xuống: csDropDownList: Mục chọn hiện tại chỉ được phép hiển thị.

csDropDown: Mục chọn hiện tại có thể được chỉnh sửa giống như hộp văn bản.

csOwnerDrawFix/csOwnerDrawVariable: Mục chọn hiện hành có thể được lập trình để vẽ.

Sự kiện thường được sử dụng nhất của hộp danh sách đồ xuống là OnChange, sự kiện này phát sinh mỗi khi phần tử được chọn hiện hành bị thay đổi.

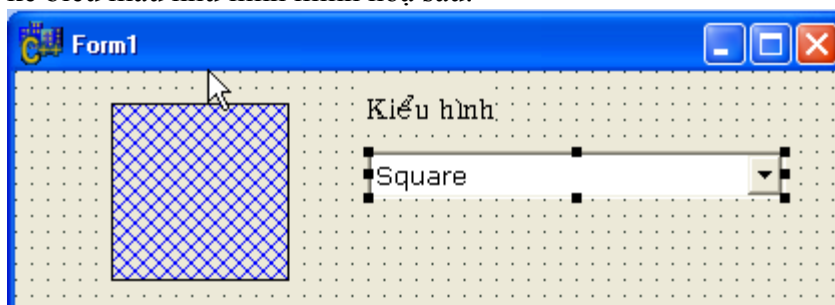
Ví dụ 1: Chúng ta sử dụng hộp danh sách đồ xuống để thay đổi kiểu hình vẽ như hình kết quả sau:



Hình 37- Chương trình Combobox

Chúng ta thiết kế chương trình trên theo các bước sau:

Bước 1: Thiết kế biểu mẫu như hình minh họa sau:



Hình 38- Thiết kế chương trình ComboBox

Các thuộc tính của các đối tượng từ trên xuống dưới và từ trái qua phải đã được thay đổi được chúng tôi liệt kê như sau:

Đối tượng	Thuộc tính	Giá trị
Shape (Ngăn Additional trên bảng công cụ)	Name	shpHinh
	Shape	Square
	Brush->Style	bsDiagCross
Label	Caption	Kiểu hình
	Font->Name	Tahoma
	Font->Size	11
ComboBox	Name	cboKieuHinh
	Items (...)	Square Rectangle Ellipse Round Rectangle

		Circle
		Round Square
	ItemIndex	0
	Style	csDropDownList

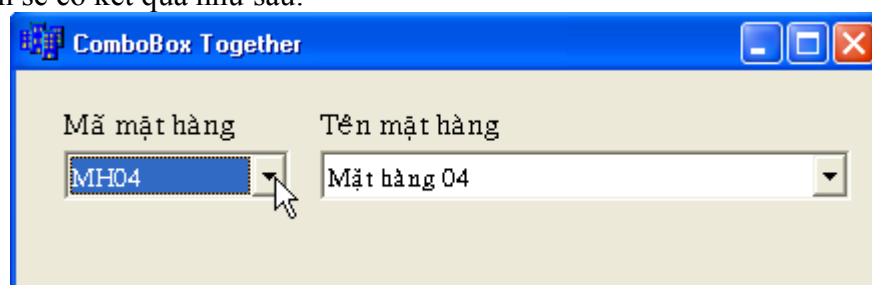
Bước 2: Viết mã lệnh cho sự kiện OnChange của đối tượng cboKieuHinh như sau:
switch (this-> cboKieuHinh ->ItemIndex)

```
{
case 0: //hình vuông
this-> shpHinh ->Shape = stSquare;
break;
case 1: //hình chữ nhật
this-> shpHinh ->Shape = stRectangle;
break;
case 2: //hình Elip
this-> shpHinh ->Shape = stEllipse;
break;
case 3: //hình chữ nhật góc tròn
this-> shpHinh ->Shape = stRoundRect;
break;
case 4: //hình tròn
this-> shpHinh ->Shape = stCircle;
break;
case 5: //hình vuông góc tròn
this-> shpHinh ->Shape = stRoundSquare;
}
```

Bước 3: Nhấn phím F9 để chạy chương trình.

Ví dụ 2: Chương trình sau sẽ cho hai hộp danh sách đồ xuống phụ thuộc vào nhau. Hộp danh sách đồ xuống này thay đổi mục chọn, hộp danh sách đồ xuống kia cũng thay đổi theo nhờ vào chỉ số phần tử đang được chọn hiện tại. Trong ví dụ này, chúng ta giả lập chọn mã hàng ở một hộp danh sách đồ xuống thì tên hàng ở một hộp danh sách đồ xuống khác sẽ hiển thị theo và ngược lại.

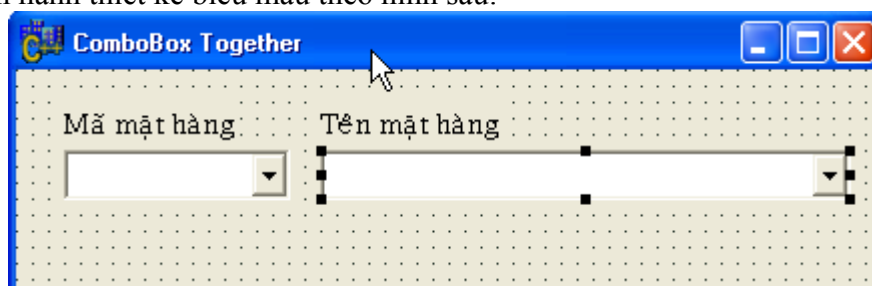
Chương trình sẽ có kết quả như sau:



Hình 39- Chương trình ComboBox Together

Chúng ta tiến hành thiết kế chương trình trên theo các bước sau:

Bước 1: Tiến hành thiết kế biểu mẫu theo hình sau:



Hình 40- Thiết kế chương trình ComboBox Together

Bảng sau liệt kê các thuộc tính đã chỉnh sửa của các đối tượng từ trên xuống dưới, từ trái qua phải:

Đối tượng	Thuộc tính	Giá trị
Form	Caption	ComboBox Together
Label	Caption	Mã mặt hàng
	Font->Name	Tahoma
	Font->Size	11
Label	Caption	Tên mặt hàng
	Font->Name	Tahoma
	Font->Size	11
ComboBox	Name	cboMaHang
	Font->Size	11
	Font->Name	Tahoma
	Items (...)	MH01 MH02 MH03 MH04 MH05
	Style	csDropDownList
ComboBox	Name	cboTenHang
	Font->Size	11
	Font->Name	Tahoma
	Items (...)	Mặt hàng 01 Mặt hàng 02 Mặt hàng 03 Mặt hàng 04 Mặt hàng 05
	Style	csDropDownList

Bước 2: Viết mã lệnh cho sự kiện OnChange của cbomahang như sau:

```
void __fastcall TForm1::cboMaHangChange(TObject *Sender)
{
    this-> cboTenHang ->ItemIndex = this-> cboMaHang ->ItemIndex ;
}
```

}

Bước 3: Viết mã lệnh cho sự kiện OnChange của cboTenHang như sau:

```
void __fastcall TForm1::cboTenHangChange(TObject *Sender)
{
    this->cboMaHang->ItemIndex = this->cboTenHang->ItemIndex;
}
```

Bước 4: Nhấn phím F9 để chạy chương trình.

Ví dụ 3: Điều chỉnh lại ví dụ bằng cách sử dụng phương thức AddItem của ComboBox để đưa vào danh sách các kiểu hình, đây là một phương thức rất thú vị, nó có thể thêm các đối tượng vào trong danh sách đi kèm với một văn bản. Khi đó, chúng ta không thêm phần tử cho Items bằng tay mà chúng ta sử dụng đoạn mã lệnh sau trong sự kiện OnCreate của biểu mẫu.

```
this->cboKieuHinh->AddItem("Square",(TObject *)stSquare);
this->cboKieuHinh->AddItem("Rectangle",(TObject *)stRectangle);
this->cboKieuHinh->AddItem("Ellipse", (TObject *)stEllipse);
this->cboKieuHinh->AddItem("Round Rectangle", (TObject *)stRoundRect);
this->cboKieuHinh->AddItem("Circle", (TObject *)stCircle);
this->cboKieuHinh->AddItem("Round Square",(TObject *)stRoundSquare);
this->cboKieuHinh->ItemIndex = 0;
```

Và sự kiện OnChange của cboKieuHinh được sửa lại như sau:

```
int i;
i= this->cboKieuHinh->ItemIndex;
if (i>=0)
    this->shpHinh->Shape=(TShapeType)this->cboKieuHinh->Items->Objects[i];
```

5.9. Thanh cuộn



Thanh cuộn còn được gọi là ScrollBar. Chúng ta đã từng gặp các ứng dụng có thanh cuộn nhằm mục đích cuộn trang, định vị, đo mức độ công việc, ...

Thanh cuộn được C++ Builder xây dựng trong lớp TScrollBar. Nó bao gồm hai mũi tên ở hai đầu và một nút trượt đặt ở giữa để xác định vị trí. Khoảng cách giữa hai đầu thanh trượt và vị trí định vị được biểu thị bằng các thuộc tính Max, Min. Chúng ta có thể điều chỉnh con trượt bằng chuột hoặc bằng phím mũi tên để di chuyển con trượt di chuyển một đoạn ngắn; dùng phím PgUp, PgDn để di chuyển con trượt di chuyển những đoạn dài.

Thuộc tính thường sử dụng nhất của thanh cuộn ScrollBar được liệt kê ở bảng sau:

Thuộc tính	Ý nghĩa
Min	Giá trị biên dưới của thanh trượt
Max	Giá trị biên trên của thanh trượt
Position	Vị trí hiện tại của con trượt
Kind	Dạng hiển thị thanh cuộn theo dạng ngang hay dạng đứng
SmallChange	Bước nhảy khi nhấn phím mũi tên
PageSize	Bước nhảy khi nhấn phím PgUp, PgDn

Sự kiện thường được sử dụng nhất của đối tượng thanh cuộn là OnChange, sự kiện này xảy ra khi chúng ta đã thay đổi con trượt của thanh cuộn.

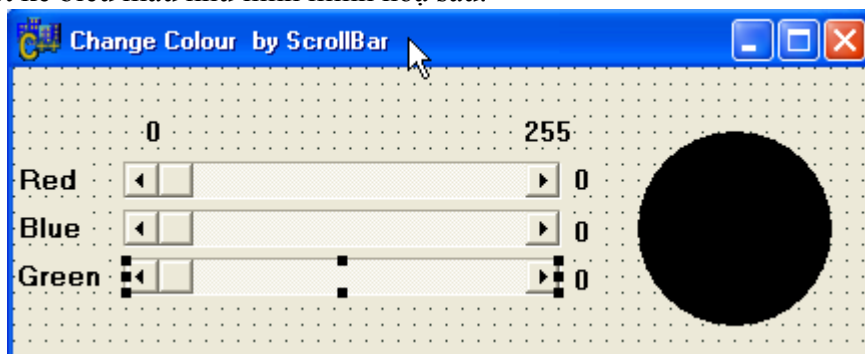
Ví dụ: Chương trình sau là ví dụ đơn giản sử dụng 3 thanh cuộn để thể hiện giá trị cho 3 sắc độ màu: đỏ (red), xanh biển (blue), xanh lục (green). Sự kết hợp 3 màu này sẽ tạo nên màu sơn cho đối tượng hình vẽ (shape) như hình minh họa sau:



Hình 41-Chương trình ScrollBar

Chúng ta tiến hành thiết kế chương trình trên theo các bước sau:

Bước 1: Thiết kế biểu mẫu như hình minh họa sau:



Hình 42-Thiết kế chương trình ScrollBar

Bảng các thuộc tính được điều chỉnh liệt kê như sau (Kể từ bây giờ khi đề cập đến bảng này, chúng ta có quy ước thứ tự các đối tượng liệt kê từ trên xuống dưới, trái qua phải)

Đối tượng	Thuộc tính	Giá trị
Form	Caption	Change colour by ScrollBar
Label	Caption	0
	Font->Size	10
	Font->Style->fsBold	true
Label	Caption	255
	Font->Size	10
	Font->Style->fsBold	true
Label	Caption	Red
	Font->Size	10
	Font->Style->fsBold	true
ScrollBar	Name	scrRed
	Max	255
	Min	0
Label	Caption	0

		Font->Size	10
		Font->Style->fsBold	true
		Name	lblRed
Label		Caption	Blue
		Font->Size	10
		Font->Style->fsBold	true
ScrollBar		Name	scrBlue
		Max	255
		Min	0
Label		Caption	0
		Font->Size	10
		Font->Style->fsBold	true
		Name	lblBlue
Label		Caption	Green
		Font->Size	10
		Font->Style->fsBold	true
ScrollBar		Name	scrGreen
		Max	255
		Min	0
Label		Caption	0
		Font->Size	10
		Font->Style->fsBold	true
		Name	lblGreen
Shape		Name	shpHinh
		Brush->Color	clBlack

Bước 2- Thêm phương thức dùng chung cho 3 sự kiện OnChange của cả 3 đối tượng scrRed,scrBlue,scrGreen:

```
void __fastcall TfrmMain::ScrollChange(TObject * Sender)
{
    //TODO: Add your source code here
    int r,g,b;
    r = this->scrRed->Position;
    g=this->scrBlue->Position;
```

```

b=this->scrGreen->Position;
this->shpHinh->Brush->Color = RGB(r,g,b);
this->lblBlue->Caption = IntToStr(b);
this->lblGreen->Caption = IntToStr(g);
this->lblRed->Caption = IntToStr(r);
}

```

Bước 3: Gắn kết sự kiện OnChange của 3 đối tượng scrRed, scrBlue, scrGreen với phương thức đã tạo ở trên.

Bước 4: Nhấn phím F9 để chạy chương trình.

5.10. Nhóm các ô chọn



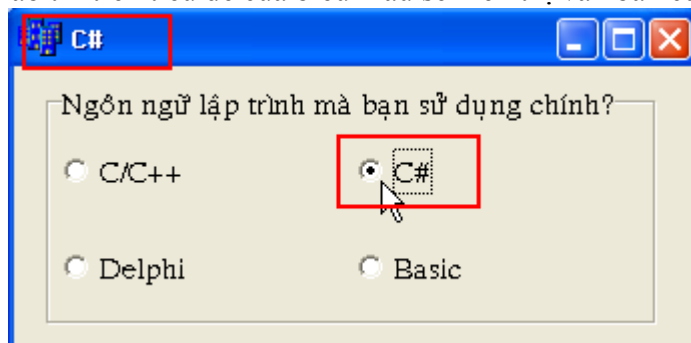
Nhóm ô chọn cho phép chúng ta tạo một nhóm các ô chọn, trong một thời điểm chỉ một ô chọn được chọn. Đối tượng này thường để gom các ô chọn cùng mục đích chọn một chức năng nào đó thành 1 nhóm. Đối tượng này được C++ Builder thiết kế trong lớp TRadioGroup.

Thuộc tính thường sử dụng nhất được liệt kê trong bảng sau:

Thuộc tính	Ý nghĩa
Items	Chứa các ô chọn
ItemIndex	Ô chọn hiện tại
Caption	Tiêu đề của nhóm ô chọn
Columns	Phân cột cho các ô chọn

Sự kiện thường dùng chính là sự kiện OnClick, sự kiện này phát sinh mỗi khi có một ô chọn được chọn.

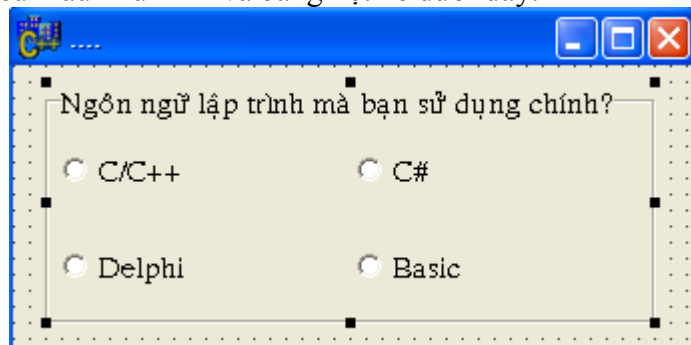
Ví dụ: Chương trình sau là minh họa đơn giản cho việc sử dụng nhóm các ô chọn, mỗi khi chúng ta chọn ô chọn nào thì trên tiêu đề của biểu mẫu sẽ hiển thị văn bản của mục chọn.



Hình 43-Chương trình RadioGroup

Chúng ta tiến hành thiết kế chương trình theo các bước dưới đây:

Bước 1: Thiết kế biểu mẫu như hình và bảng liệt kê dưới đây:



Hình 44-Thiết kế chương trình RadioGroup

Bảng liệt kê các thuộc tính chỉnh sửa:

Đối tượng	Thuộc tính	Giá trị
Form	Caption
RadioGroup	Name	rdgLang
	Caption	Ngôn ngữ lập trình mà bạn sử dụng chính?
	Items(...)	C/C++ C++ Builder C# Basic
	Columns	2
	Font->Name	Tahoma
	Font->Size	11

Bước 2: Viết mã lệnh cho sự kiện OnClick của đối tượng rdgLang như sau:

```
void __fastcall TfrmMain::rdgLangClick(TObject *Sender)
```

```
{
    int i = this->rdgLang->ItemIndex;
    if (i >= 0)
        this->Caption = this->rdgLang->Items->Strings[i];
}
```

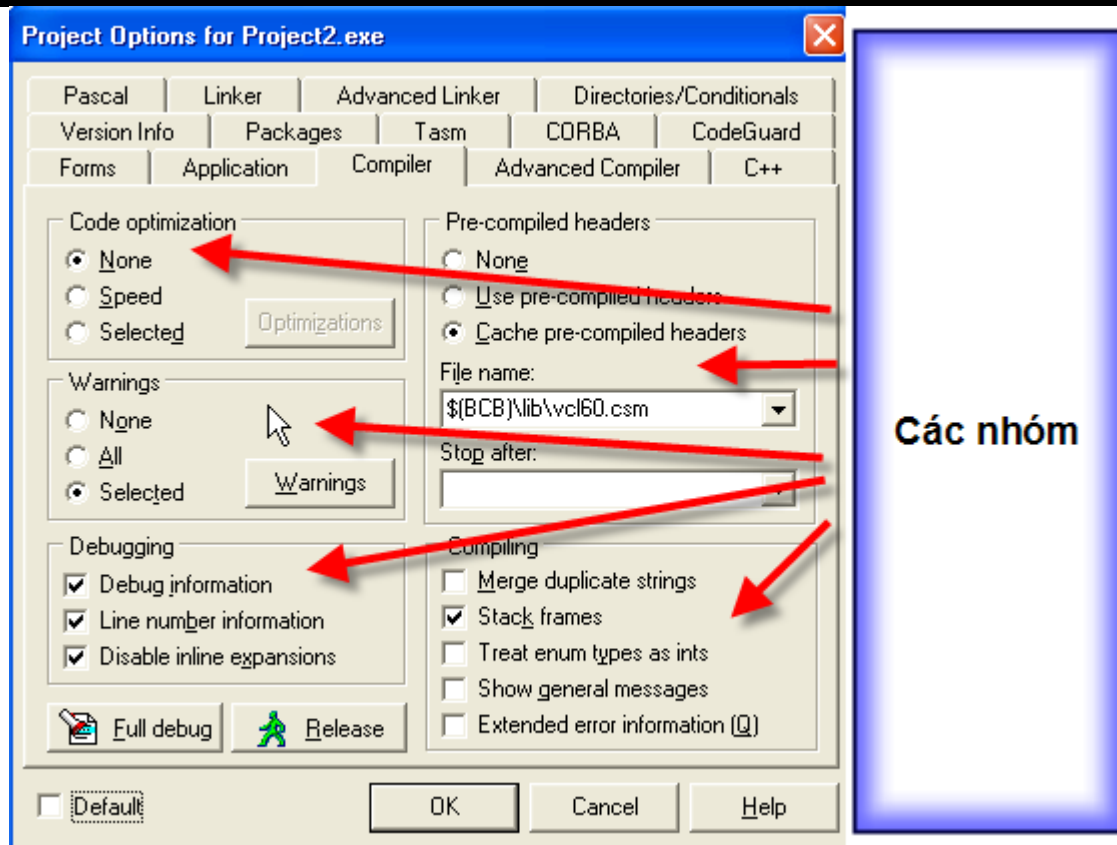
Bước 3: Nhấn phím F9 để chạy chương trình.

5.11. Nhóm các đối tượng



Như ta đã đề cập ở trên, các ô chọn có thể được nhóm thành một nhóm nhờ vào đối tượng RadioGroup còn các đối tượng khác chỉ có thể đặt rời rạc trên biểu mẫu. Đối tượng nhóm các đối tượng (groupbox) được dùng để nhóm các đối tượng có cùng chức năng để phân loại các đối tượng theo chủ đề. GroupBox thường cũng chỉ dùng để gom các đối tượng có cùng chức năng lại mà thôi.

Hình sau minh họa cho chúng ta thấy các nhóm được nhóm lại theo từng chủ đề:



Hình 45-Minh họa các nhóm

Một đối tượng tương tự như GroupBox nhưng không có tiêu đề là đối tượng Panel. Cách hoạt động của Panel giống như GroupBox.

Chúng tôi đã cố gắng liệt kê những đối tượng thường sử dụng nhất. Trong những nội dung tiếp theo, chúng tôi sẽ trình bày các đối tượng khác khi có liên quan đến những đối tượng đó.

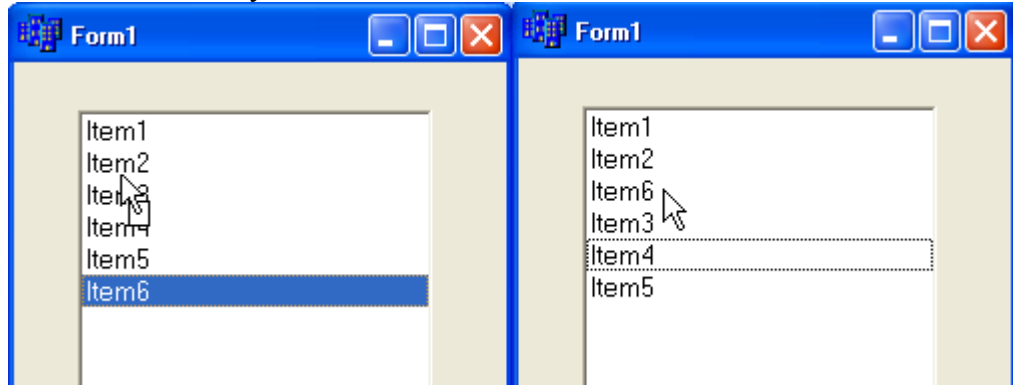
BÀI 3- LÀM VIỆC VỚI CÁC ĐIỀU KHIỂN

1. Thực hiện các hành động rê và thả

Các sự kiện cần phải đáp ứng được khi một đối tượng được kéo và thả đó là: BeginDrag → DragOver → DragDrop → EndDrag. Sự kiện kết thúc của quá trình kéo và thả là DragDrop khi đối tượng được kéo vào có chấp nhận sự việc kéo này hay không; còn sự kiện EndDrag là sự kiện kết thúc quá trình kéo và thả dù cho đối tượng được kéo vào có chấp nhận hay không.

Chúng ta sẽ tiến hành xét một số ví dụ sau để hiểu quá trình kéo và thả.

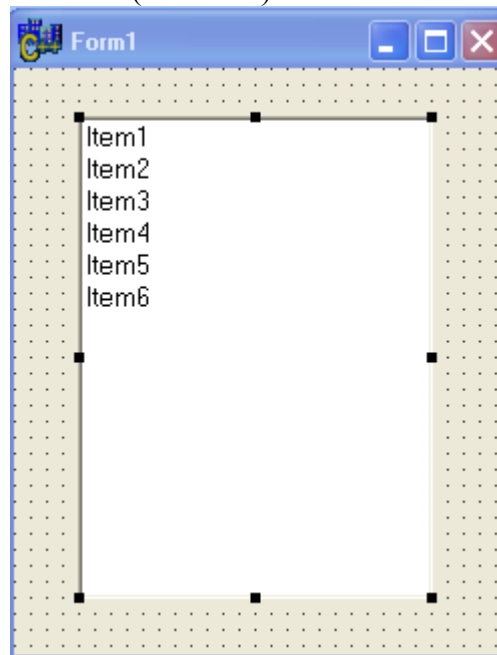
Ví dụ 1: Kéo và thả để thay đổi vị trí các mục chọn trên một ListBox:



Hình 46-Chương trình ListBox Drag and Drop

Chúng ta thiết kế chương trình theo các bước sau:

Bước 1: Tạo Form và một TListBox (ListBox1) như hình sau:



Hình 47- Thiết kế chương trình ListBox Drag and Drop

Trong đó thuộc tính của DragMode của ListBox1 thành dmAutomatic để thao tác kéo thả được tự động phát sinh.

Bước 2: Thêm một biến StartingPoint để lưu giữ vị trí con trỏ chuột khi bắt đầu thao tác kéo thả trong tập tin thư viện .h như trong đoạn mã của tập tin Unit1.h dưới đây:

```
//-----
#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
```

```

#include <StdCtrls.hpp>
#include <Forms.hpp>
//-----
class TForm1 : public TForm
{
__published:      // IDE-managed Components
    TListBox *ListBox1;
    void __fastcall ListBox1MouseDown(TObject *Sender,TMouseButton Button,
TShiftState Shift, int X, int Y);
    void __fastcall ListBox1DragOver(TObject *Sender, TObject *Source, int X, int Y,
TDragState State, bool &Accept);
    void __fastcall ListBox1DragDrop(TObject *Sender, TObject *Source,int X, int Y);
private:      // User declarations
    TPoint StartingPoint; //khai báo thêm biến cục bộ
public:      // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif

```

Bước 3: Viết mã lệnh cho sự kiện OnMouseDown của ListBox1 như sau để tiến hành lưu giữ vị trí đầu tiên của quá trình kéo thả:

```

void __fastcall TForm1::ListBox1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    StartingPoint= TPoint(X,Y); //lấy vị trí con trỏ chuột
}

```

Bước 4: Viết đoạn mã lệnh cho sự kiện OnDragOver của đối tượng ListBox1 để xác định xem đối tượng đang được kéo trên ListBox1 có được phép thả hay không.

```

void __fastcall TForm1::ListBox1DragOver(TObject *Sender, TObject *Source,
    int X, int Y, TDragState State, bool &Accept)
{
    Accept = (Source == ListBox1); //chỉ chấp nhận khi đối tượng đang được kéo là ListBox1.
}

```

Bước 5: Viết đoạn mã lệnh cho sự kiện OnDragDrop (sự kiện này chỉ phát sinh khi đối tượng ListBox1 có Accept = true trong bước 4) của đối tượng ListBox1.

```

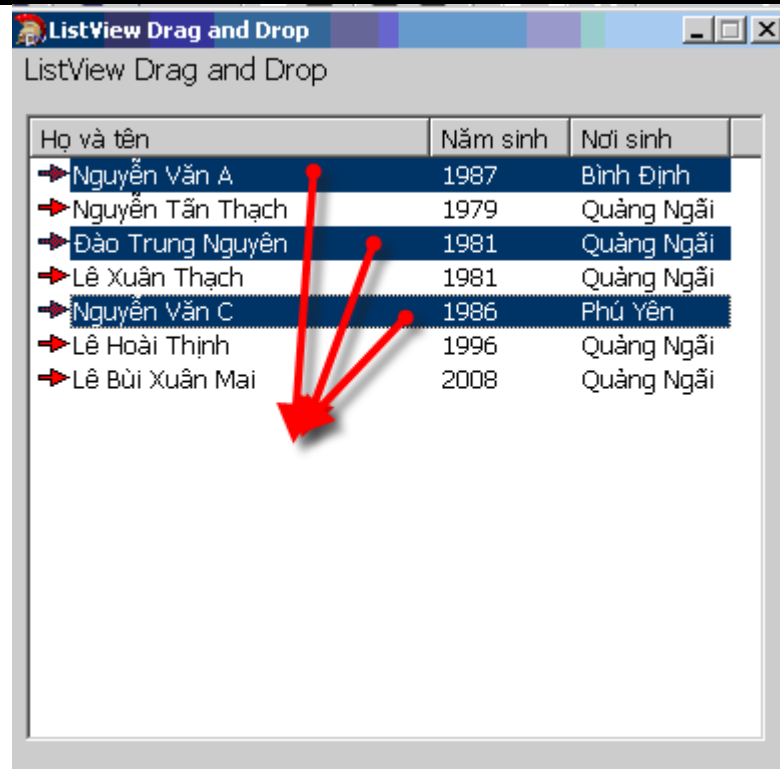
void __fastcall TForm1::ListBox1DragDrop(TObject *Sender, TObject *Source,
    int X, int Y)
{
    TListBox *temp;
    int DropPosition, StartPosition; //chỉ số phần tử đầu và phần tử cuối
    TPoint DropPoint;
    DropPoint = TPoint(X,Y);
    temp = (TListBox *) Source;
    StartPosition =temp->ItemAtPos(StartingPoint,true);
    DropPosition = temp->ItemAtPos(DropPoint,true) ;
    temp->Items->Move(StartPosition, DropPosition) ; //di chuyển phần tử đến vị trí kéo
}

```

Bước 6: Nhấn F9 để thực thi chương trình.

Ở ví dụ trên, chúng ta đã thực hiện thao tác kéo và thả mục chọn của ListBox. Trong ví dụ sau, chúng ta sẽ thực hiện thao tác kéo thả nhiều đối tượng trên một ListView.

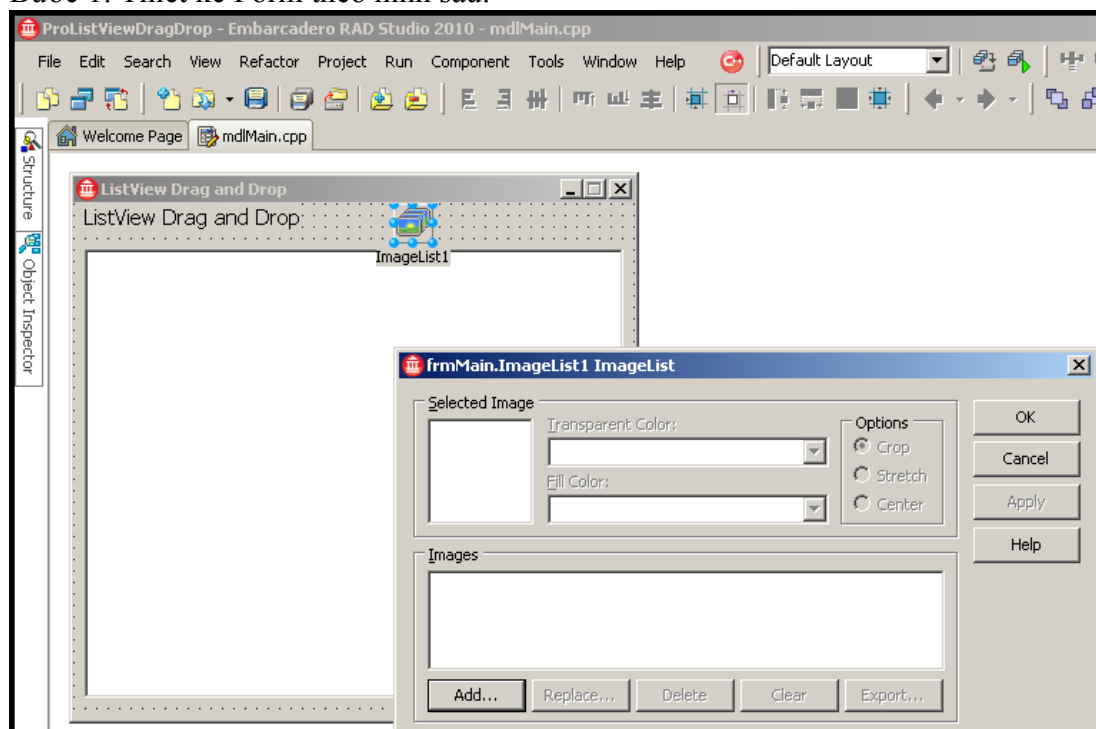
Ví dụ 2: Chương trình thực hiện kéo và thả để sắp xếp lại các mục chọn trên ListView.



Hình 48-ListView Items Drag

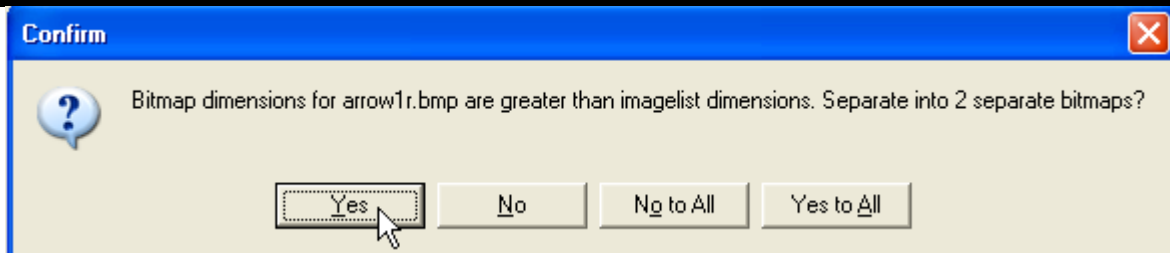
Chúng ta tiến hành thiết kế chương trình theo từng bước như sau.

Bước 1: Thiết kế Form theo hình sau:



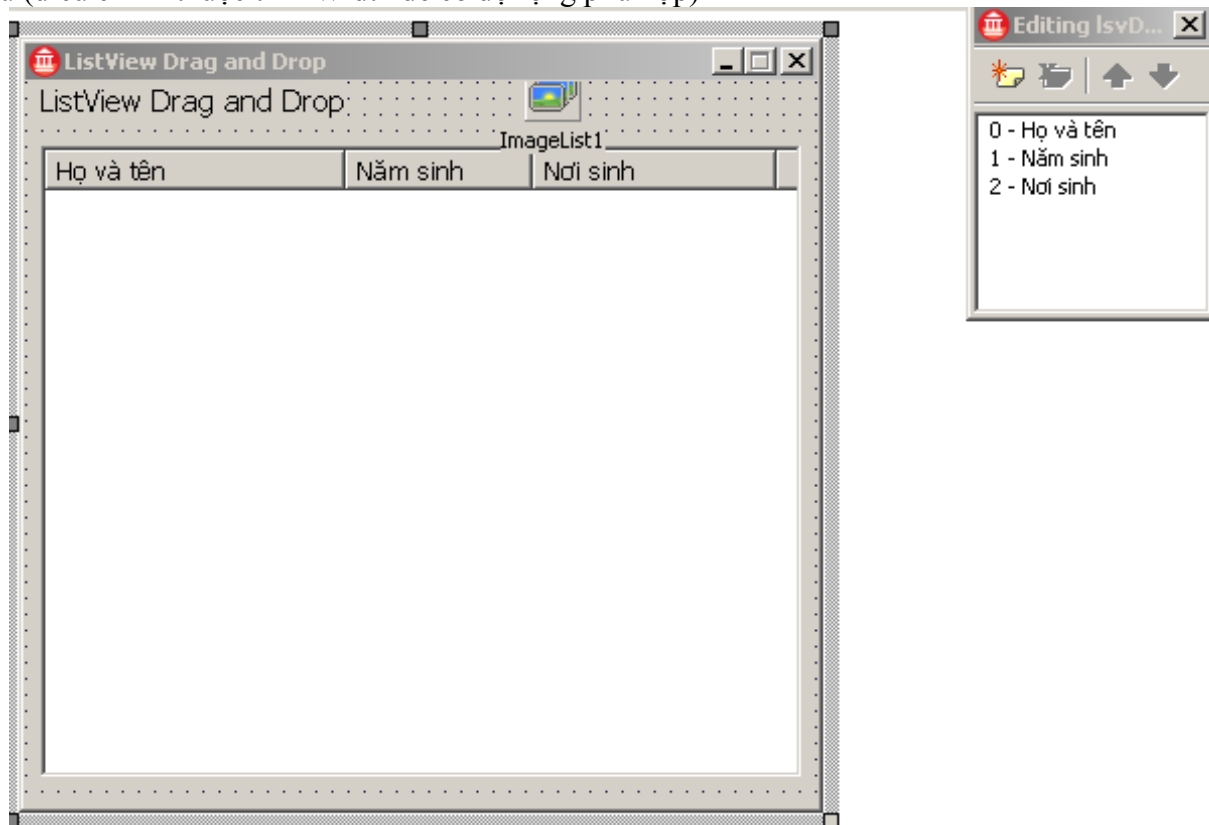
Hình 49- Thiết kế Form của chương trình ListViewItems Drag and Drop

Nhấn đôi chuột lên ImageList1 để mở cửa sổ soạn thảo hình ảnh, nhấn chọn nút Add và chọn hình ảnh theo đường dẫn C:\Program Files\Common Files\CodeGear SharedImages\Buttons\arrow1r.bmp, gấp màn hình sau chúng ta chọn Yes:



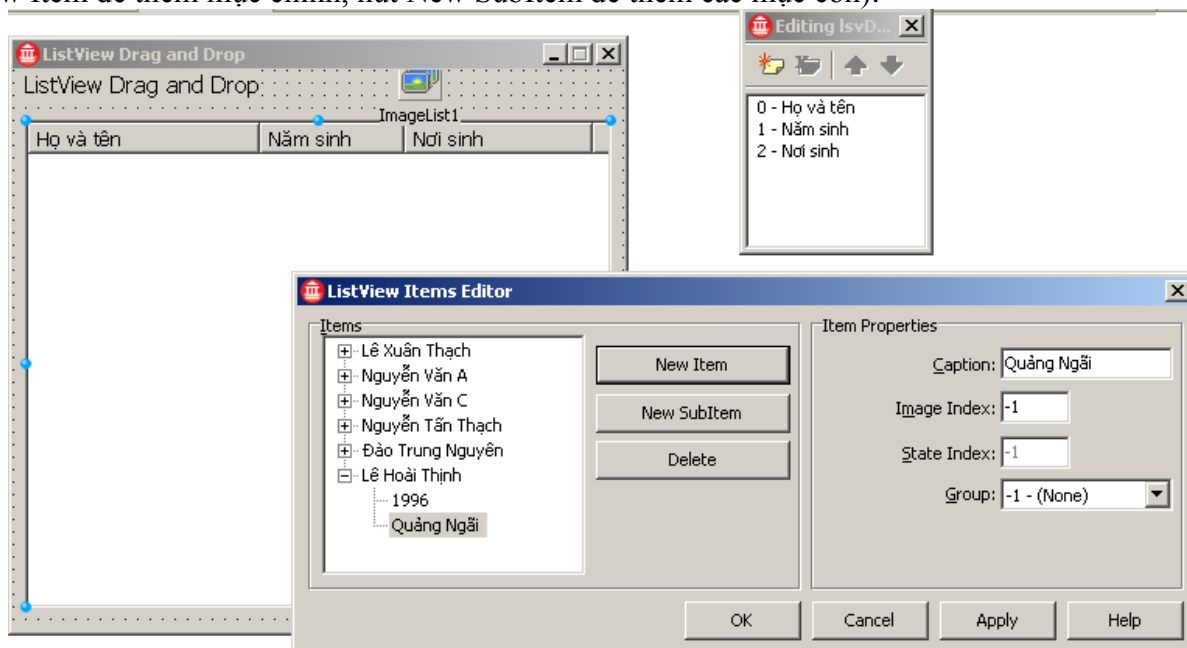
Hình 50-Thêm hình ảnh vào ImageList

Tiếp theo, nhấn đôi chuột vào đối tượng lsvDragDrop để soạn thảo các cột theo hình minh họa sau (điều chỉnh thuộc tính Width để có độ rộng phù hợp)



Hình 51-Soạn thảo các cột cho lsvDragDrop

Chọn thuộc tính Items của lsvDragDrop để đưa vào các mục chọn như hình minh họa sau (nút New Item để thêm mục chính, nút New SubItem để thêm các mục con):



Hình 52-Thêm mục chọn cho lsvDragDrop

Cuối cùng, điều chỉnh thuộc tính của lstDragDrop: RowSelect = true, DragMode=dmAutomatic, MultiSelect = true, ViewStyle = vsReport.

Bước 2: Viết mã lệnh cho sự kiện DragOver của ListView1 như sau:

```
void __fastcall TfrmMain::lsvDragDropDragOver(TObject *Sender, TObject *Source,
    int X, int Y, TDragState State, bool &Accept)
```

```
{
    Accept = (Sender == lsvDragDrop);
}
```

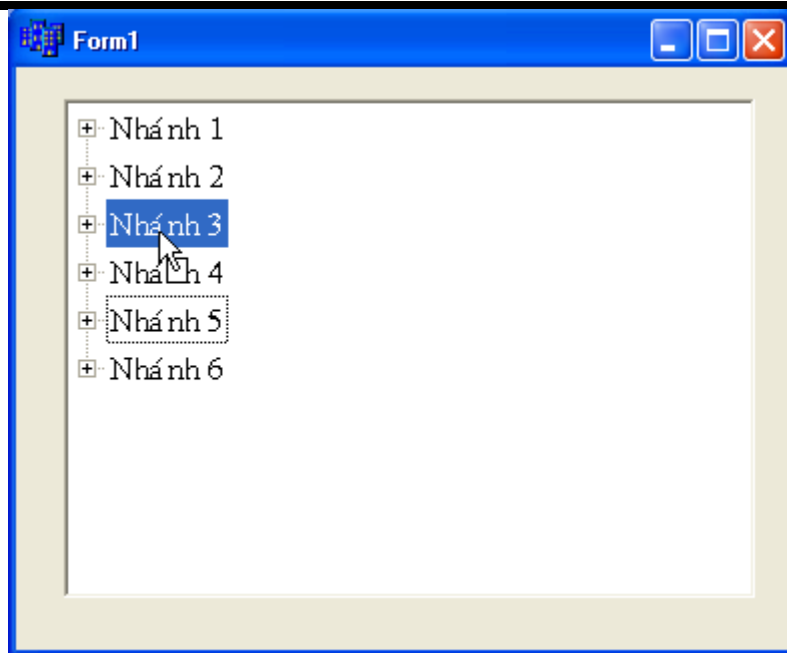
Bước 3: Viết mã lệnh cho sự kiện DragDrop của ListView1 như sau:

```
void __fastcall TfrmMain::lsvDragDropDragDrop(TObject *Sender, TObject *Source,
    int X, int Y)
```

```
{
    TListItem *currentitem, *nextitem, *dragitem, *dropitem ;
    TListView *temp;
    TItemStates selected = TItemStates() << isSelected;
    if(Sender == Source )
    {
        temp=(TListView *)Sender;
        //xác định phần tử tại vị trí chuột
        dropitem = temp->GetItemAt(X,Y);
        currentitem = temp->Selected; //phần tử được chọn đầu tiên
        while( currentitem != NULL)
        {
            nextitem = temp->GetNextItem(currentitem, sdAll, selected) ;
            //tạo ra một phần tử rỗng tại vị trí sẽ chèn vào
            // phần tử này được lưu trong biến dragitem
            if(dropitem!=NULL)
                dragitem = temp->Items->Insert(dropitem->Index);
            else
                dragitem = temp->Items->Add();
            //gán giá trị của currentitem cho dragitem
            dragitem->Assign(currentitem) ;
            delete currentitem;
            currentitem = nextitem;
        }
    }
}
```

Bước 4: Nhấn F9 để thực thi chương trình.

Ví dụ 3: Chương trình sau kéo thả các phần tử trên TreeView.



Hình 53-TreeView Drag and Drop

Mã nguồn chương trình như sau:

```
//-----

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
    int X, int Y, TDragState State, bool &Accept)
{
    Accept = (Sender == TreeView1);
}
//-----

void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
    int X, int Y)
{
    TTreeNode *ToDeleteItem, *DropItem, *CurrentItem, *NextItem;
    int ChildCount, Childrun;

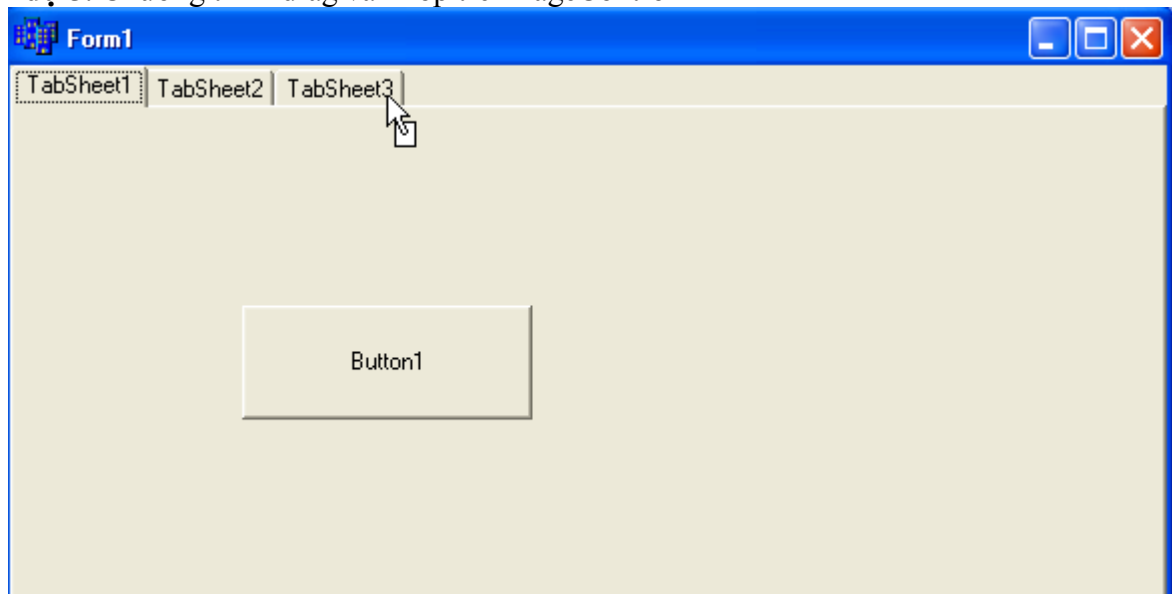
    if(Sender == Source )
    {
        DropItem = TreeView1->GetNodeAt(X, Y);
```

```

    CurrentItem = TreeView1->Selected;
    ToDeleteItem = CurrentItem;
    NextItem = TreeView1->Items->Insert(DropItem, CurrentItem->Text);
    DropItem = NextItem;
    ChildCount = TreeView1->Selected->Count;
    for( Childrun = 1; Childrun <=ChildCount; Childrun ++)
    {
        CurrentItem = TreeView1->Selected->getFirstChild();
        NextItem = TreeView1->Items->AddChild(DropItem, CurrentItem->Text);
        TreeView1->Items->Delete(CurrentItem);
    }
    delete ToDeleteItem;
}
}
//-----

```

Ví dụ 5: Chương trình drag và Drop trên PageControl



Hình 54-Page Control Drag and Drop

Mã lệnh của chương trình như sau:

```

//-----

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TForm1::PageControl1MouseDown(TObject *Sender,

```

```

TMouseButton Button, TShiftState Shift, int X, int Y)
{
PageControl1->BeginDrag(true,0) ;

}
//-----

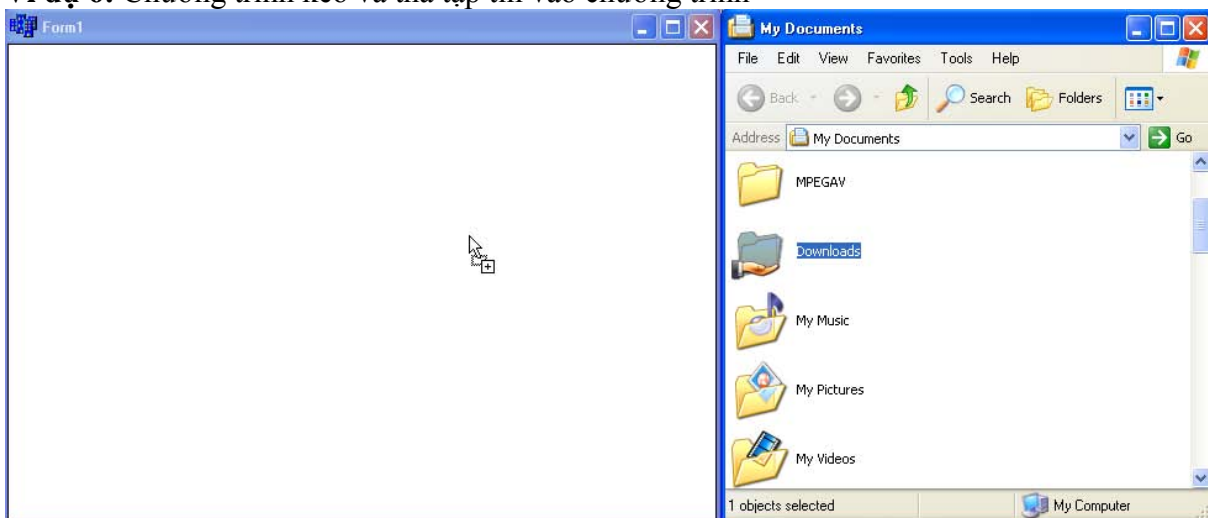
void __fastcall TForm1::PageControl1DragDrop(TObject *Sender,
TObject *Source, int X, int Y)
{
TRect TabRect;
int j;
if (Sender->ClassNameIs("TPageControl"))
for (j = 0; j <= PageControl1->PageCount - 1; j++) {
PageControl1->Perform(TCM_GETITEMRECT, j, (int) & TabRect);
if (PtInRect(TabRect, Point(X, Y))) {
if (PageControl1->ActivePage->PageIndex != j)
PageControl1->ActivePage->PageIndex = j;
return;
}
}
}

//-----
void __fastcall TForm1::PageControl1DragOver(TObject *Sender,
TObject *Source, int X, int Y, TDragState State, bool &Accept)
{
Accept = (Sender == PageControl1);
}
//-----

```

Tuy nhiên, để kéo và thả tập tin, chúng ta phải thực hiện một phương pháp khác đó là phương pháp đón nhận sự kiện WM_DROPFILES. Đây là một phương pháp phức tạp, chúng tôi đưa vào trong tài liệu này như một ví dụ minh họa mà không phân tích quá chi tiết.

Ví dụ 6: Chương trình kéo và thả tập tin vào chương trình



Hình 55-Chương trình Files and Folders Drag

Để thực hiện chương trình này, chúng ta dùng một ListBox và phải thêm vào hai phương thức; một phương thức đáp ứng sự kiện kéo và thả file (thông điệp WM_DROPFILES) và một phương thức dùng để liệt kê tập tin và thư mục con của một thư mục chỉ định. Ngoài ra, còn một số biến được thêm vào ở ngăn private.

Mã lệnh của chương trình:

Tập tin Unit1.h:

```
//-----

#ifndef Unit1H
#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <CheckLst.hpp>
//-----
class TForm1 : public TForm
{
__published:      // IDE-managed Components
    TListBox *ListBox1;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
private:          // User declarations
    //khai báo thêm các biến và phương thức
    //MAX_PATH: hằng số cho biết độ dài tối đa của đường dẫn đến tập tin trong hệ điều hành
    char  DroppedFile[MAX_PATH*2+1] ;
    long  TotalDroppedFiles, Counter ;
    void  *FindHandle ;

    void  ProcessDir(UnicodeString root);
    void  DragProc(TMessage & Msg);
public:           // User declarations
    __fastcall TForm1(TComponent* Owner);
protected:
    BEGIN_MESSAGE_MAP
        VCL_MESSAGE_HANDLER(WM_DROPFILES, TMessage, DragProc)
    END_MESSAGE_MAP(TForm)

};
//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif
Mã lệnh của tập tin Unit1.cpp
//-----

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
```

```

__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    //cho phép ListBox1 chấp nhận kéo và thả tập tin trên nó
    DragAcceptFiles(this->ListBox1->Handle,true);
}
//-----
void TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    DragAcceptFiles(this->ListBox1->Handle,false); //trả về trạng thái ban đầu
}
//-----
void TForm1::ProcessDir(UnicodeString root) {
    TSearchRec filesearch;
    UnicodeString folder;
    folder = root;
    // tìm kiếm các tập tin và thư mục
    if (FindFirst(folder + "\\*.*", faAnyFile, filesearch) == 0) {
        do {
            // nếu là thư mục và khác ".." (thư mục cha) và "." (chính thư mục này) thì
            // đệ quy vào thư mục con
            if ((filesearch.Attr & faDirectory) == faDirectory) {
                if (filesearch.Name != ".." && filesearch.Name != ".") {
                    // yêu cầu chương trình xử lý thông điệp tránh trường hợp //
                    // bị đóng băng với thư mục lớn
                    Application->ProcessMessages();
                    ProcessDir(folder + "\\" + filesearch.Name);
                }
            }
            else {
                this->ListBox1->Items->Add(folder + "\\" + filesearch.Name);
            }
        }
        while (FindNext(filesearch) == 0);
    }
}
// -----
void TForm1::DragProc(TMessage &Msg) {
    /* Chúng ta chuyển tham số $FFFFFFFF để lấy số tập tin được kéo vào */
    TotalDroppedFiles = DragQueryFile((HDROP)Msg.WParam, 0xFFFFFFFF, NULL, 0);
    UnicodeString root;
    for (Counter = 0; Counter <= TotalDroppedFiles - 1; Counter++) {
        // lấy đường dẫn từng tập tin hay thư mục
        DragQueryFile((HDROP)Msg.WParam, Counter, DroppedFile, sizeof
            (DroppedFile));
        // nếu là thư mục thì gọi hàm ProcessDir
        if ((GetFileAttributes(DroppedFile) & FILE_ATTRIBUTE_DIRECTORY)
            == FILE_ATTRIBUTE_DIRECTORY) {

```

```

        root = AnsiString(DroppedFile);
        ProcessDir(root);
    }
    else {
        // nếu là tập tin thì thêm trực tiếp vào ListBox1
        root = AnsiString(DroppedFile);
        this->ListBox1->Items->Add(root);
    }
}
// Giải phóng lượng bộ nhớ chiếm giữ bởi các tập tin đang được kéo
DragFinish((HDROP)Msg.WParam);
}

```

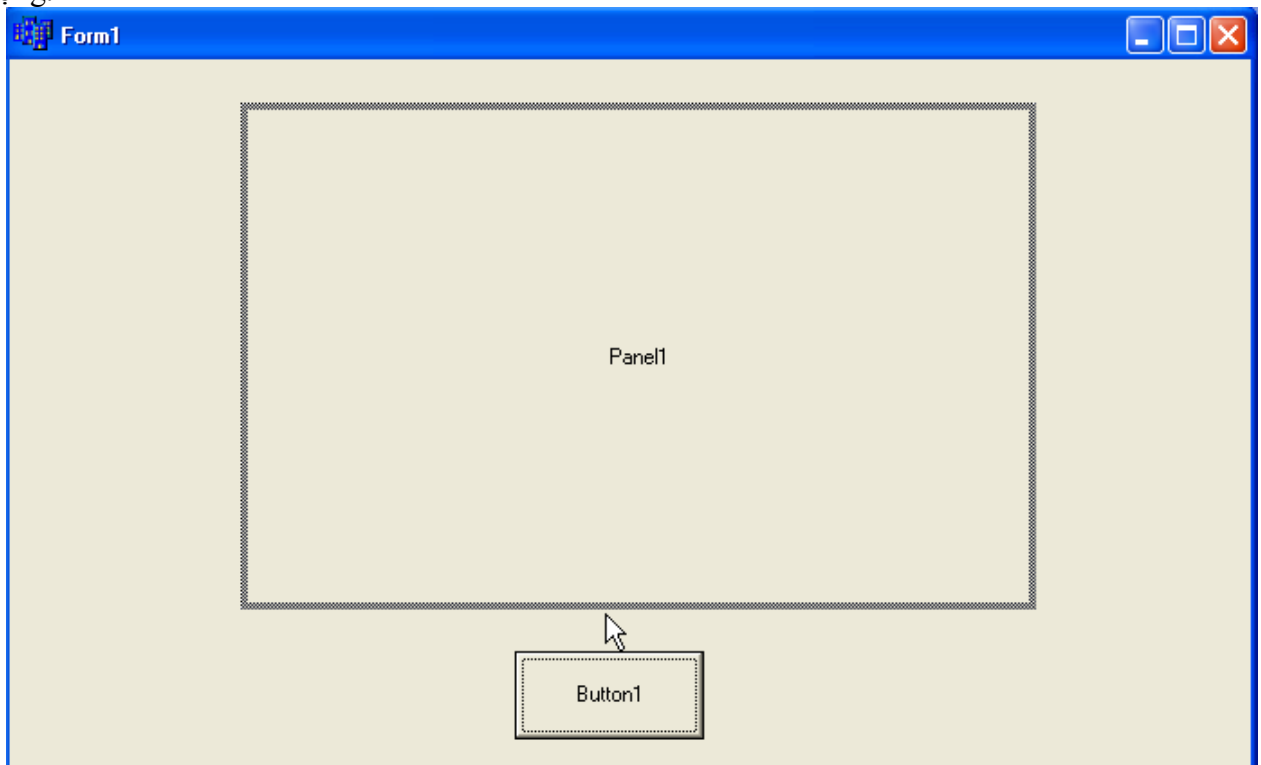
2. Thực hiện các hành động rê và kết dính

Hoạt động rê và kết dính (Drag and Dock) là hành động tương tự như kéo và thả nhưng khi kết dính, đối tượng sẽ được kết dính vào biên của đối tượng khác và nhận đối tượng được kết dính làm đối tượng cha.

Các sự kiện phát sinh trong quá trình kết dính gồm: OnBeginDock, OnDockOver, OnDockDrop, EndDock.

Trong nội dung bài học này, chúng ta sẽ nghiên cứu cách thực hiện các hành động rê và kết dính thông qua một số ví dụ sau:

Ví dụ: Chương trình sau sử dụng phương thức Dock tự động của đối tượng để kết dính đối tượng.



Hình 56-Dock Button

Chúng ta tiến hành thiết kế Form, đặt đối tượng Panel lên trên Form và đổi các thuộc tính DockSite thành true để Panel chấp nhận cho phép kết dính đối tượng lên nó.

Tiếp theo, đặt đối tượng Button lên Form, điều chỉnh các thuộc tính: DragKind=dkDock và DragMode = dmAutomatic.

Khi đó, đối tượng Button1 sẽ có thể được kết dính lên đối tượng Panel1 và nếu chúng ta muốn truy bắt các sự kiện thì chúng ta sử dụng OnBeginDock, OnDockOver, OnDockDrag, OnEndDock.

3. Làm việc với văn bản trong các điều khiển

a) Cài đặt lề cho văn bản

Trong đối tượng RichText hay đối tượng memo, văn bản có thể được canh phải, canh trái hay canh giữa. Để điều chỉnh dạng canh lề của văn bản, chúng ta sử dụng thuộc tính Alignment. Thuộc tính này chỉ có tác dụng khi thuộc tính WordWrap điều cài đặt thành true. Nếu WordWrap được đặt thành false sẽ không có bất cứ canh lề nào được áp dụng.

Để lấy ví dụ, giả sử chúng ta có một menu để thực hiện canh lề trái, phải và giữa cho văn bản. Đoạn mã lện sau minh họa phương pháp chúng ta xử lý các mục chọn và văn bản trong richtext như sau:

```
void TEditForm::AlignClick(TObject *Sender)
{
    TMenuItem *temp;
    Left1.Checked := False; { clear all three checks */
    Right1->Checked = False;
    Center1->Checked = False;
    temp = (TMenuItem *)Sender;
    temp->Checked=true;
    if(Left1->Checked )
        Editor-> Alignment = taLeftJustify
    else if(Right1->Checked )
        Editor-> Alignment = taRightJustify
    else if(Center1->Checked )
        Editor-> Alignment = taCenter;
}
```

b) Thêm thanh cuộn vào thời điểm thực thi

Đối tượng RichEdit và memo có thể chứa thanh cuộn ngang hoặc cuộn dọc hoặc cả hai nếu cần thiết. Khi thuộc tính WordWrap được kích hoạt, chỉ có một thanh cuộn dọc được hiển thị mà thôi.

Để thêm thanh cuộn vào thời điểm thực thi, chúng ta có thể sử dụng đoạn lệnh ví dụ như sau:

```
void __fastcall TEditForm::WordWrap1Click(TObject *Sender)
{
    Editor->WordWrap := not WordWrap; { toggle word-wrapping */
    if(WordWrap )
        ScrollBars:= ssVertical { wrapped requires only vertical */
    else
        ScrollBars := ssBoth; { unwrapped might need both */
        WordWrap1.Checked := WordWrap; { check menu item to match property */
}
```

c) Thêm đối tượng Clipboard

Hầu hết các chương trình xử lý văn bản đều cho phép di chuyển đoạn văn bản chọn lựa qua lại giữa các ứng dụng, thậm chí các văn bản trong các ứng dụng khác nhau. Đối tượng Clipboard của C++ Builder là hiện thân của một Clipboard trong Windows và nó cũng bao gồm các phương thức cắt, copy và dán văn bản (thậm chí các định dạng khác, chẳng hạn như hình ảnh). Đối tượng Clipboard được khai báo trong thư viện Clipbrd.hpp (hoặc Clipbrd.h).

Để thêm một đối tượng Clipboard vào một ứng dụng chúng ta phải thêm câu lệnh khai báo thư viện này lên đoạn khai báo thư viện của C++ Builder (#include Clipbrd.hpp).

d) Đoạn văn bản được chọn

Trước khi chúng ta có thể gởi bất cứ đoạn văn bản nào vào Clipboard, đoạn văn bản này phải được chọn trước. Khi người sử dụng chọn một đoạn văn bản, nó hiển thị bằng cách đánh dấu (thường là một vùng nền đen, chữ trắng)

Các thuộc tính hỗ trợ để chúng ta xử lý đoạn văn bản lựa chọn như sau:

Thuộc tính	Ý nghĩa
SelText	Chứa chuỗi văn bản được chọn.
SelLength	Chứa chiều dài văn bản được chọn.
SelStart	Chứa vị trí bắt đầu của một chuỗi.

e) Chọn tất cả văn bản

Để chọn tất cả văn bản có trong điều khiển, chúng ta có thể sử dụng phương thức Select All. Đây là một phương pháp hiệu quả để chọn các văn bản đang hiển thị ở vùng khuất. Thường thường, khi người sử dụng chọn văn bản hay sử dụng chuột và bàn phím.

f) Cắt, sao chép và dán văn bản

Các ứng dụng sử dụng thư viện Clipbrd đều có thể cắt, sao chép và dán văn bản, đồ họa và đối tượng thông qua Clipboard của Windows. Chúng ta có thể sử dụng các đoạn mã lệnh tương tự như sau:

```
void __fastcall TEditForm::CutToClipboard(TObject *Sender)
{
    Editor->CutToClipboard();
}
void __fastcall TEditForm::CopyToClipboard(TObject *Sender)
{
    Editor->CopyToClipboard();
}
void __fastcall TEditForm::PasteFromClipboard(TObject *Sender)
{
    Editor->PasteFromClipboard();
}
```

g) Xóa đoạn văn bản lựa chọn

Chúng ta cũng có thể xóa đoạn văn bản đã được lựa chọn mà không phải cắt nó đưa vào Clipboard. Để làm được điều này, chúng ta phải gọi phương thức ClearSelection. Ví dụ, để xóa đoạn văn bản đã được lựa chọn chúng ta có thể sử dụng đoạn lệnh như sau:

```
void __fastcall TEditForm::Delete(Sender: TObject);
{
    RichEdit1->ClearSelection();
}
```

4. Thêm đồ họa vào điều khiển

Các điều khiển list-box, combo-box và menu có một kiểu dáng đặc biệt gọi là “kiểu tự vẽ”, có nghĩa là chúng ta tự vẽ các phần tử, văn bản của phần tử bằng mã lệnh thực thi. Đây là phương cách hay dùng để đưa các đồ họa vào điều khiển.

Thông thường, để tạo một đối tượng tự vẽ trong C++ Builder, chúng ta phải qua ba bước sau:

- Chỉ định để hệ thống biết là đối tượng tự vẽ.

Cả hai danh sách listbox và combobox đều có một thuộc tính Style. Style cho phép chỉ định phương pháp vẽ các phần tử là mặc định hay tự vẽ. Grid (lưới) sử dụng một thuộc tính DefaultDrawing để cho phép hay không cho phép việc hệ thống vẽ các mục chọn một cách mặc định.

Listbox và Combobox có thêm kiểu tự vẽ gồm: fixed và variable được diễn giải trong bảng sau:

Kiểu tự vẽ	Ý nghĩa	Giá trị
Fixed	Mỗi phần tử có cùng chiều cao, chiều cao này được định bởi thuộc tính ItemHeight.	lbOwnerDrawFixed, csOwnerDrawFixed
Variable	Mỗi phần tử có thể có chiều cao khác nhau và phụ thuộc vào dữ liệu tại thời điểm thực thi.	lbOwnerDrawVariable, csOwnerDrawVariable

- Thêm các đối tượng hình ảnh vào trong một String List.

Một string list đều có khả năng lưu giữ một danh sách các đối tượng kèm theo một danh sách chuỗi. Ví dụ, trong một ứng dụng quản lý tập tin, chúng ta có thể thêm hình ảnh để chỉ ổ đĩa với từng tên của ổ đĩa. Để thực hiện điều này, chúng ta cần phải thêm một số hình ảnh vào trong ứng dụng, sau đó sao chép chúng vào một string list. Chúng ta cũng có thể sử dụng các hình ảnh trong các đối tượng hình ảnh ẩn, chẳng hạn như:

- + Thêm một đối tượng hình ảnh vào form chính.
- + Cài đặt tên cho đối tượng.
- + Cài đặt thuộc tính Visible thành false.
- + Tải hình ảnh bằng thuộc tính Picture.

Đối tượng hình ảnh này sẽ ẩn khi ứng dụng được thực thi.

Một khi chúng ta sử dụng hình ảnh trong ứng dụng, chúng ta có thể đính kèm nó với danh sách văn bản trong String list. Chúng ta có thể thêm đồng thời cả đối tượng và văn bản đính kèm vào trong string list.

Ví dụ dưới đây sẽ cho phép chúng ta tìm hiểu cách thêm hình ảnh vào một string list. Đây là một phần của chương trình quản lý tập tin, trong đó, mỗi ổ đĩa sẽ có một hình ảnh và một ký tự đi kèm. Để thêm hình ảnh, chúng ta truy bắt sự kiện OnCreate của Form như đoạn lệnh dưới

```
void __fastcall TFMForm::FormCreate(TObject *Sender)
{
    char Drive;
    int    AddedIndex;
    for( Drive = 'A'; Drive <= 'Z'; drive++) /* iterate through all possible drives */
    {
        switch(GetDriveType(Drive + ':'))
        { /* positive values mean valid drives */
            DRIVE_REMOVABLE: // add a tab
                AddedIndex = DriveTabSet->Tabs->AddObject(Drive, Floppy->Picture->Graphic);
            DRIVE_FIXED: //add a tab *
                AddedIndex = DriveTabSet->Tabs->AddObject(Drive, Fixed->Picture->Graphic);
            DRIVE_REMOTE: // add a tab
                AddedIndex = DriveTabSet->Tabs->AddObject(Drive, Network->Picture->Graphic);
            if (Drive == DirectoryOutline.Drive) then //current drive?
                DriveTabSet.TabIndex := AddedIndex; //then make that current tab
        }
    }
}
```

- Vẽ các phần tử.

Khi một đối tượng được cài đặt kiểu dáng thành tự vẽ, Windows sẽ không vẽ đối tượng đó trên màn hình mà nó sẽ phát sinh sự kiện cho mỗi phần tử trong đối tượng. Chúng ta phải viết mã lệnh để vẽ các phần tử này tương ứng với các sự kiện vẽ phần tử. Chỉ sử dụng một đoạn mã lệnh thống nhất để vẽ cho tất cả các phần tử

Trước khi cho phép ứng dụng vẽ đối tượng, Windows sẽ phát sinh một sự kiện đơn vị đo của phần tử. Sự kiện này nói lên rằng phần tử sẽ được hiển thị ở đâu trên đối tượng. Windows sẽ tính toán kích thước của phần tử (thông thường, nó đủ lớn để hiển thị văn bản của đối tượng trong font chữ hiện tại). Chúng ta phải nắm giữ các thông tin này và thay đổi hình chữ nhật mà

windows chọn. Ví dụ, nếu chúng ta muốn vẽ một hình ảnh thay cho văn bản của phần tử thì chúng ta đổi hình chữ nhật cho khớp với kích thước của hình ảnh. Để đổi kích thước của phần tử tự vẽ, chúng ta phải viết mã lệnh đáp ứng cho sự kiện đo phần tử. Tùy thuộc vào từng đối tượng, tên này có thể khác nhau. Đối với listbox và combobox, tên của sự kiện đo phần tử là OnMeasureItem. Lưới (Grid) không có sự kiện đo phần tử. Sự kiện này có hai tham số quan trọng: chỉ số của phần tử và kích thước của phần tử đó.

Kích cỡ có thể thể hiện theo nhiều cách khác nhau, đối với listbox và combobox. Kích thước chứa độ cao của phần tử, còn độ rộng luôn luôn bằng độ rộng của đối tượng. ví dụ:

```
void __fastcall TFMForm::DriveTabSetMeasureTab(TObject *Sender, int Index,
int &TabWidth)
{
    int    BitmapWidth;
    TBitmap *temp;
    temp =(TBitmap *)DriveTabSet->Tabs->Objects[Index];
    BitmapWidth = temp->Width;
    TabWidth= 2 + BitmapWidth;
}
```

Sau đó tiến hành vẽ đối tượng theo sự kiện OnDrawTab (trong listbox và combo box là OnDrawItem)

```
void __fastcall TFMForm::DriveTabSetDrawTab( TObject *Sender, TCanvas *TabCanvas,
TRect R, int Index, bool Selected)
    TBitmap *Bitmap;
{
    Bitmap = (TBitmap *)DriveTabSet->Tabs->Objects[Index];

    TabCanvas->Draw(R.Left, R.Top + 4, Bitmap); /* draw bitmap */
    TabCanvas->TextOut(R.Left + 2 + Bitmap.Width, /* position text */
R.Top + 2, DriveTabSet.Tabs[Index]); /* and draw it to the right of the
bitmap */
}
```

Trong trường hợp một số đối tượng có hỗ trợ thuộc tính Image hay thuộc tính Images, chúng ta có thể đưa hình ảnh trực tiếp khi thiết kế đối tượng bằng cách tải hình ảnh vào hoặc thông qua ImageList. Riêng nội dung vẽ các phần tử tự vẽ chúng tôi sẽ đề cập sâu hơn trong bài 7.


BÀI 4- TƯƠNG TÁC GIỮA NGƯỜI DÙNG VÀ ỨNG DỤNG

1. Sử dụng các hộp thoại

Các hộp thoại Windows:

Một hộp thoại là một cửa sổ dùng để tương tác với máy tính. Bản thân hộp thoại không mang một ý nghĩa gì. Các điều khiển nằm trên nó thực hiện luật tương tác giữa người sử dụng và máy tính.

Trước tiên, một hộp thoại có những đặc trưng sau đây:

Nó được trang bị một nút nhấn Close . Nút nhấn này cho phép người sử dụng hủy bỏ hộp thoại. Thông thường, nút nhấn này sẽ được cấu hình giống như người sử dụng nhấn Cancel hoặc nhấn phím Esc.

Nó không thể thu nhỏ, phóng to, hay trả lại trạng thái ban đầu (restored). Một hộp thoại không có nút hệ thống nào khác ngoài nút Close.

Nó là modal. Người sử dụng thường không cho phép tiếp tục các hành động khác cho đến khi anh ta đóng hộp thoại này.

Nó cung cấp một cách cho người sử dụng đóng hay hủy bỏ hộp thoại. Đa số hộp thoại có một nút Ok và một nút Cancel, tùy thuộc trên ứng dụng phát triển. Khi các hộp thoại có nút Ok và Cancel, nút Ok được cấu hình để có thể kích hoạt bằng phím Enter; nút Cancel được kích hoạt bằng phím Esc.

Hộp thoại thông điệp (Message Boxes)



Hình 57-Hộp thoại minh họa

Một hộp thoại thông điệp là một hộp thoại nhỏ dùng để hiển thị một thông điệp và cung cấp một hay nhiều nút nhấn. Nó thường dùng để cung cấp một thông tin tới người sử dụng hay yêu cầu một quyết định từ người sử dụng. Bằng cách nhấn một trong những nút nhấn, người sử dụng tạo ra một quyết định và chương trình tiếp tục.

Các hộp thoại thông điệp được tạo từ một hàm dựng sẵn từ VCL và thư viện Win32. Chúng ta tiến hành chuẩn bị để tạo một ứng dụng mẫu về các hộp thoại thông điệp theo các bước sau:

Tạo một dự án mới với form mặc định của nó.

Đổi thuộc tính caption của form thành Message Boxes Configurations

Từ ngăn công cụ Standard, chọn đối tượng Edit và đặt nó lên Form.

Đổi tên của đối tượng này thành edtMessage và xóa nội dung trong thuộc tính Text.

Hàm ShowMessage

Hàm ShowMessage() cung cấp các hộp thoại cơ bản nhất của Borland. Hàm này có một tham số chuỗi và không trả về bất cứ giá trị nào. Nó được sử dụng để hiển thị một thông điệp đến người sử dụng và người sử dụng nhấn nút Ok để chấp nhận. Cú pháp của hàm ShowMessage() là:

```
void __fastcall ShowMessage(const AnsiString Message);
```

Một hộp thoại thông điệp được tạo ra với hàm ShowMessage() sử dụng tên của dự án làm tiêu đề của nó. Thông điệp để hiển thị là một chuỗi được cung cấp bởi người sử dụng. Đây là một ví dụ:

```
//-----
void __fastcall TForm1::btnShowMsgClick(TObject *Sender)
{
    ShowMessage("Welcome to the Sellers Bank.");
}
//-----
```

Chuỗi này cũng có thể dẫn xuất từ một đối tượng khác chẳng hạn như nội dung của một hộp thoại thảo, một vùng nhớ (memo) hoặc bất kỳ đối tượng văn bản nào khác. Đây là một ví dụ:

```
//-----
void __fastcall TForm1::btnMsgFromEditClick(TObject *Sender)
{
    ShowMessage(edtMessage->Text);
}
//-----
```

Chuỗi này cũng có thể là một sự ghép nối nhiều chuỗi khác nhau:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ShowMessage("The name " + AnsiString("") +
        edtMessage->Text + AnsiString("") + " is not in our records.");
}
//-----
```

Sử dụng hàm ShowMessage()

Từ ngăn công cụ Standard, chọn đối tượng Button và đặt nó lên form.

Thay đổi thuộc tính của Button thành Show &Msg và đổi tên của nó thành btnShowMsg.

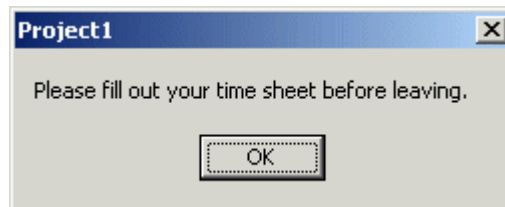
Nhấn đôi chuột lên nút btnShowMsg để viết mã lệnh cho sự kiện Click

Nhấn phím tab và cài đặt đoạn mã lệnh sau:

```
//-----
void __fastcall TForm1::btnShowMsgClick(TObject *Sender)
{
    ShowMessage("Please fill out your Time Sheet before leaving.");
}
//-----
```

Để kiểm thử chương trình, nhấn phím F9.

Nhấn vào nút Show Msg:



Hình 58-Hộp thoại hiển thị khi nhấn vào nút Show Msg

Nhấn Ok để đóng form .

Để lưu dự án, trên thực đơn chính, chọn File/Save All

Tìm thư mục để lưu ví dụ này.

Nhấn nút Create new Folder. Gõ Message Boxes và nhấn phím Enter hai lần để hiển thị thư mục mới trong hộp đồ xuống Save.

Nhấn nút Save để lưu Unit.

Gõ Messages để thay tên của dự án và nhấn Enter

Để hiển thị thông điệp nhiều hơn một dòng, thay đổi mã lệnh thành như sau:

```
//-----
void __fastcall TForm1::btnShowMsgClick(TObject *Sender)
{
    ShowMessage("Please fill out your Time Sheet before leaving.\n"
        "Make sure you sign and send it to Human Resources.");
}
//-----
```

Chạy chương trình và xem kết quả:

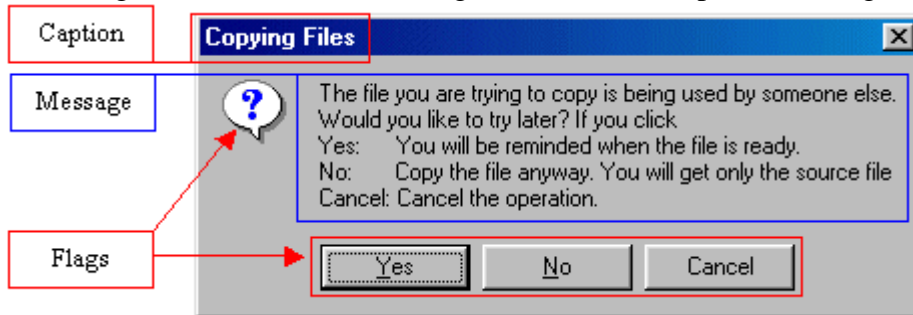


Hình 59-Hộp thoại nhiều dòng

Hàm MessageBox

Hàm MessageBox() được dẫn xuất từ Win32. Cú pháp của nó là:

```
int __fastcall MessageBox(const char * Message, const char * Caption, int Flags);
```



Hình 60-Minh họa các tham số của hàm MessageBox()

MessageBox() được tạo ra với ba tham số. Tham số đầu tiên, Message, là một chuỗi kết thúc rỗng chỉ định thông điệp mà người sử dụng sẽ đọc. Chuỗi Message có thể là một câu tĩnh. Nó có thể được kiểm tạo từ một đối tượng khác. Hoặc nó có thể là một sự kết hợp của nhiều chuỗi khác nhau bằng cách sử dụng các hàm và phép toán chuỗi C/C++.

Chúng ta có thể tạo một hộp thoại thông điệp đơn giản tương tự như việc sử dụng hàm ShowMessage() để hiển thị một hộp thoại thông điệp đơn giản với nút OK. Trong trường hợp này, chúng ta chỉ cần cung cấp một tham số duy nhất Message. Cài đặt hai tham số còn lại là NULL. Đây là ví dụ:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Application->MessageBox( "This operation can only be "
                             "performed by an administrator.", NULL, NULL);
}
```

Tham số thứ 2, cũng là một chuỗi, là tiêu đề sẽ hiển thị trên thanh tiêu đề của hộp thoại. Chúng ta cũng có thể cài đặt nó khi tạo hộp thoại thông điệp hay chúng ta có thể xây dựng nó vào lúc thực thi. Nếu chúng ta không có tiêu đề, chúng ta có thể cài đặt giá trị của tham số này thành NULL. Trong trường hợp này, tiêu đề sẽ hiển thị thành Error. Do đó, để tạo một hộp thoại ít buồn chán, cung cấp tham số Caption. Đây là ví dụ:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Application->MessageBox("Make sure the music is playing.",
                             "CD PLayer Instructions", NULL);
}
```

Tham số thứ ba chỉ định các cờ chúng ta muốn hiển thị trên hộp thoại: một hay nhiều nút nhấn và một số hình ảnh tùy chọn. Chúng ta cũng có thể tạo một hộp thoại đơn giản với một nút duy nhất OK. Trong trường hợp đó, cài đặt tham số thứ ba thành MB_OK. Đây là ví dụ:







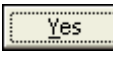


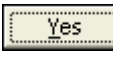

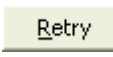


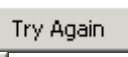
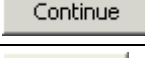

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
```

```

Application->MessageBox("Make sure the music is playing.",
    "CD PLayer Instructions", MB_OK);
}
//-----

```

Để hiển thị nhiều hơn một nút nhấn, sử dụng các hằng số nguyên để ấn định một nhóm các nút cho phép. Sau đây là các hằng số và các nút nhấn của nó:

Hằng số nguyên	Các nút nhấn
MB_OK	
MB_OKCANCEL	 
MB_ABORTRETRYIGNORE	  
MB_YESNOCANCEL	  
MB_YESNO	 
MB_RETRYCANCEL	 
MB_CANCELTRYCONTINUE	  
MB_HELP	

Ví dụ, để tạo một hộp thoại hiển thị nút Yes và nút No, chúng ta có thể viết:





```

//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Application->MessageBox("Do you hear any music now or any sound at all?",
        "CD Player Instructions", MB_YESNO);
}
//-----

```

Nếu chúng ta cung cấp MB_HELP như một nút nhấn duy nhất, hộp thoại sẽ hiển thị với một nút OK và một nút Help.

Chúng ta có thể hiển thị một biểu tượng bằng các sử dụng một trong các hằng số nguyên Win32. Mặc dù, chúng ta có thể sử dụng bất kỳ biểu tượng với bất kỳ nút nhấn nào, chúng ta phải khéo trong việc hiển thị biểu tượng tương ứng với ngữ cảnh của thông điệp. Các giá trị của các biểu tượng được liệt kê:

Giá trị	Biểu tượng	Thích hợp khi
MB_ICONEXCLAMATION MB_ICONWARNING		Cảnh báo người sử dụng một hành động thi hành trên ứng dụng
MB_ICONINFORMATION MB_ICONASTERISK		Thông báo người sử dụng một trạng thái không phê phán
MB_ICONQUESTION		Hỏi một câu hỏi với câu trả lời: Yes hoặc No hay Yes, No hoặc Cancel
MB_ICONSTOP MB_ICONERROR		Một tình huống phê phán hay có lỗi xảy ra. Biểu tượng này tương ứng khi thông tin cho người sử



dùng một sự kết thúc hay một sự từ chối một hành động.

Các biểu tượng này được sử dụng chung với các hằng số nút nhấn. Để kết hợp các cờ này, sử dụng phép toán bit OR “|”. Đây là một ví dụ:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Application->MessageBox("Do you hear any music now or any sound at all?",
        "CD Player Instructions",
        MB_YESNOCANCEL | MB_ICONQUESTION);
}
//-----
```

Khi một hộp thoại được cấu hình để hiển thị một hay nhiều nút nhấn, hệ điều hành sẽ cài đặt quyết định nút nào là nút mặc định. Nút mặc định sẽ có một đường viền đậm và được đặt tách rời với các nút khác. Nếu người sử dụng nhấn phím Enter, hộp thoại sẽ phát sinh sự kiện như người sử dụng nhấn nút trên nút mặc định. May mắn thay, nếu hộp thoại có nhiều hơn một nút, chúng ta có thể quyết định nút nhấn nào sẽ trở thành mặc định. Để chỉ định nút mặc định, sử dụng một trong những hằng số sau:

Giá trị	Nếu hộp thoại có nhiều nút nhấn, nút nhấn mặc định sẽ là
MB_DEFBUTTON1	Nút đầu tiên
MB_DEFBUTTON2	Nút thứ hai
MB_DEFBUTTON3	Nút thứ ba
MB_DEFBUTTON4	Nút thứ tư

Để chỉ định nút nhấn mặc định, sử dụng phép toán bit OR để kết hợp các hằng số nguyên để chỉ định nút nhấn mặc định với các hằng số nút nhấn và biểu tượng. Sau đây là một ví dụ:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Application->MessageBox("Do you hear any music now or any sound at all?",
        "CD Player Instructions",
        MB_YESNOCANCEL | MB_ICONQUESTION | MB_DEFBUTTON2);
}
//-----
```

Sau khi đọc thông điệp hiển thị trên một hộp thoại, người sử dụng sẽ nhấn một trong những nút nhấn và hộp thoại sẽ đóng lại. Mỗi một nút nhấn có một hằng số nguyên được gán và đồng bộ bởi trình dịch. Chúng ta có thể sử dụng các giá trị này để tìm xem nút nhấn nào đã được nhấn. Điều này có nghĩa là hàm MessageBox() trả về một giá trị số nguyên được liệt kê trong bảng sau:

MessageBox() trả về	Nếu người sử dụng nhấn
IDOK	OK hoặc phím Enter
IDCANCEL	Cancel hoặc phím Esc
IDABORT	Abort
IDRETRY	Retry
IDIGNORE	Ignore
IDNO	No
IDYES	Yes
IDCONTINUE	Continue
IDTRYAGAIN	Try Again

Do đó, chúng ta có thể sử dụng một trong những giá trị nguyên này để hành động phụ thuộc trên nút nhấn được nhấn:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if( Application->MessageBox(
        "Do you hear any music now or any sound at all?",
        "CD Player Instructions",
        MB_YESNOCANCEL | MB_ICONQUESTION) == IDNO )
        Panel1->Caption = "We will stop these tests now. "
            "Let the machine rest!";
}
//-----
```

Sử dụng hàm *MessageBox()*

Thêm đối tượng Button lên form.

Đổi Caption của đối tượng Button thành Simple &1 và tên của nó thành btnSimpleMsg.

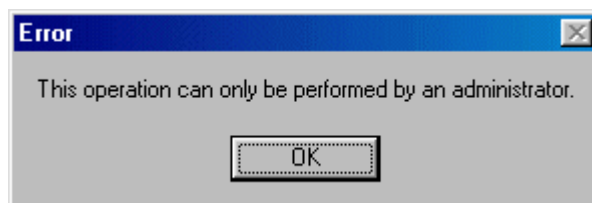
Nhấn đôi chuột lên nút nhấn để viết mã lệnh cho sự kiện Click

Nhấn phím Tab và gõ:

```
Application->MessageBox("This operation can only be performed by an administrator.",
NULL, NULL);
```

Để kiểm thử form, nhấn phím F9.

Nhấn nút Simple 1 để xem hộp thoại thông điệp. Chú ý rằng hộp thoại có một nút nhấn OK và một tiêu đề Error:



Hình 61-Mình họa kết quả sử dụng hàm *MessageBox()*

Nhấn OK và đóng form.

Nhấn phím F12 để hiển thị form.

Thêm một nút nhấn khác vào form.

Đổi tiêu đề (caption) của nó thành Simple &2 và tên của nó thành btnSimple2.

Nhấn đôi chuột vào nút Simple 2.

Nhấn Tab và gõ:

```
Application->MessageBox("Make sure the music is playing.", "CD PLayer Instructions",
MB_OK);
```

Kiểm thử form và về lại Borland C++ Builder.

Thêm một nút nhấn khác vào form.

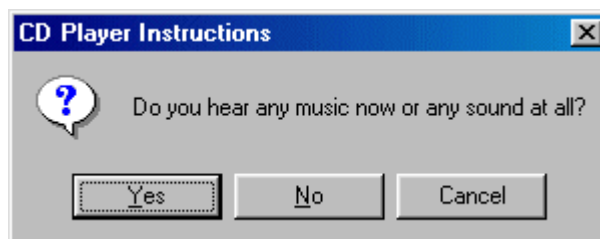
Đổi tiêu đề của nó thành &Question và tên của nó thành btnQuestion.

Nhấn đôi chuột lên nút Question.

Nhấn phím Tab và gõ:

```
Application->MessageBox("Do you hear any music now or any sound at all?",
"CD Player Instructions", MB_YESNOCANCEL | MB_ICONQUESTION);
```

Kiểm thử form:



Hình 62-Hộp thoại Question

Trở lại Borland C++ Builder.

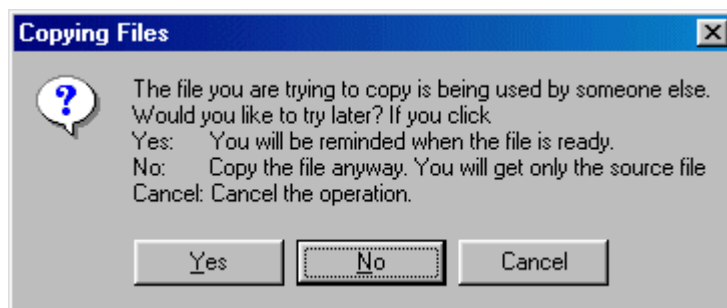
Thêm một nút nhấn khác vào Form.

Đổi Caption thành &Default và tên của nó thành btnDefault.

Nhấn đôi chuột và cài đặt đoạn mã sau:

```
//-----
void __fastcall TForm1::btnDefaultClick(TObject *Sender)
{
    Application->MessageBox(
        "The file you are trying to copy is being "
        "used by someone else.\n"
        "Would you like to try later? If you click\n"
        "Yes: You will be reminded when the file is ready.\n"
        "No: Copy the file anyway. You will get only the source file\n"
        "Cancel: Cancel the operation.", "Copying Files",
        MB_YESNOCANCEL | MB_ICONQUESTION | MB_DEFBUTTON2);
}
//-----
```

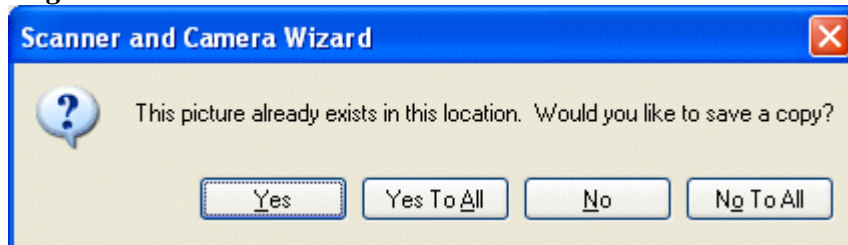
Kiểm thử Form:



Hình 63-Hộp thoại có nút nhấn mặc định

Trở lại Borland C++ Builder

Hàm MessageDlg:



Hình 64-Mình họa kết quả của hàm MessageDlg

Hàm MessageDlg() là một hộp thông điệp tinh chỉnh của Borland và nó cung cấp một sự thay thế tốt cho hàm MessageBox() của Win32.



Cú pháp của hàm MessageDlg() là:

```
int __fastcall MessageDlg(const AnsiString Message, TMsgDlgType IconType,
    TMsgDlgButtons Buttons, int HelpContext);
```


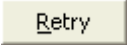
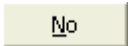
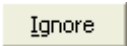
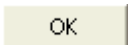

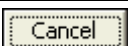
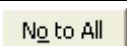

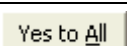
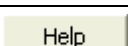
Tham số đầu tiên, Message, là thông điệp gửi đến người sử dụng. Nó có thể là một câu đơn giản, một đoạn hay sự kết hợp nhiều chuỗi.

IconType là một biểu tượng được sử dụng để làm nổi bật hộp thoại. Biểu tượng này được cài đặt bằng cách sử dụng một hằng số nguyên trong số các hằng sau:

Giá trị	Biểu tượng
mtWarning	
mtError	

mtInformation	
mtConfirmation	
mtCustom	None

Tham số Buttons được sử dụng kiểu của nút nhấn sẽ hiển thị trong hộp thoại. Các nút nhấn được định nghĩa sử dụng tập hợp TMsgDlgButtons như sau:

Giá trị	Nút nhấn	Giá trị	Nút nhấn
mbYes		mbRetry	
mbNo		mbIgnore	
mbOK		mbAll	
mbCancel		mbNoToAll	
mbAbort		mbYesToAll	
mbHelp			

Tham số cuối cùng được sử dụng nếu một tập tin hướng dẫn được cho phép, trong trường hợp này chúng ta muốn chỉ định một phần chỉ mục liên quan đến hộp thoại này. Hộp thoại này sử dụng tên của dự án làm tiêu đề của nó.

Sử dụng hàm MessageDlg:

Thêm một nút nhấn vào form.

Thay đổi tiêu đề của nó thành Msg Di&alog 1 và tên của nó thành btnMsgDialog1.

Nhấn đôi chuột lên nút nhấn Msg Dlg1.

Nhấn phím Tab và gõ:

```
MessageDlg("All songs on the CD have been copied. Now it will be ejected.",
    mtInformation, TMsgDlgButtons() << mbOK, 0);
```

Kiểm thử form

Thêm một nút nhấn vào form

Thay đổi tiêu đề của nó thành Msg Dia&log2 và tên của nó thành btnMsgDlg2.

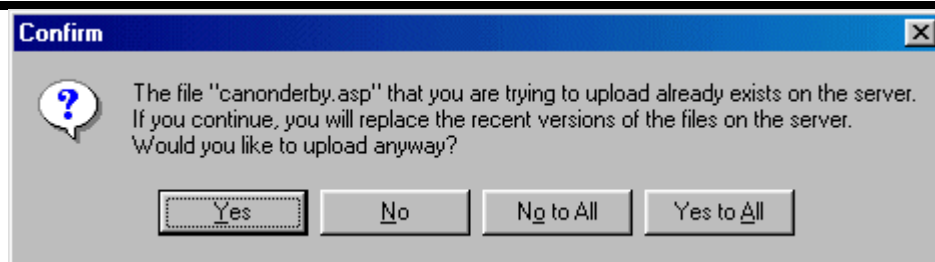
Nhấn đôi chuột lên nút Msg Dialog2 và cài đặt đoạn mã sau:

```
//-----
void __fastcall TForm1::btnMsgDlg2Click(TObject *Sender)
{
    MessageDlg("The file " + AnsiString("\\") +
        edtMessage->Text + AnsiString("\\") +
        " that you are trying to upload "
        "already exists on the server.\n"
        "If you continue, you will replace the recent versions "
        "of the files on the server.\n"
        "Would you like to upload anyway?",
        mtConfirmation, TMsgDlgButtons() << mbNoToAll
            << mbNo << mbYes
            << mbYesToAll, 0);
}
//-----
```

Để kiểm thử form, nhấn F9.

Trong hộp văn bản hộp thoại, chúng ta gõ canonderby.asp

Nhấn chuột lên nút Msg Dialog 2:



Hình 65-Kết quả hiển thị của hàm MessageDlg()

Trở lại Borland C++ Builder

Hộp thoại nhập liệu

InputBox() cho phép chúng ta hiển thị một hộp thông điệp yêu cầu một thông tin từ người sử dụng. Hộp thoại này được trang bị một hộp văn bản và hai nút nhấn. Hộp văn bản chấp nhận một chuỗi từ người sử dụng. Người sử dụng có thể gõ bất kỳ văn bản nào. Khi người sử dụng nhấn nút OK, hộp nhập liệu sẽ trả về nội dung của hộp văn bản của nó. Nếu người sử dụng nhấn Cancel hoặc phím Esc, nội dung của hộp văn bản sẽ được hủy bỏ.

Cú pháp của hàm InputBox() là

```
AnsiString __fastcall InputBox(const AnsiString Caption,
                               const AnsiString Prompt, const AnsiString Default);
```

Tham số Caption chỉ định chuỗi sẽ hiển thị trên thanh tiêu đề của hộp thoại. Prompt là một câu sẽ hiển thị đến người sử dụng để yêu cầu thông tin gõ vào hộp soạn thảo. Tham số Default là một giá trị gợi ý chúng ta, nó có thể hiển thị trong hộp soạn thảo để hướng dẫn người sử dụng kiểu của giá trị mong chờ. Nếu chúng ta không muốn chỉ định một giá trị mặc định, chúng ta có thể cài đặt giá trị của nó thành rỗng. Sau đây là ví dụ:

```
//-----
void __fastcall TForm1::btnInputClick(TObject *Sender)
{
    InputBox( "Temperature Conversion",
             "Type the temperature in Celsius:", "");
}
//-----
```

Mặc dù tham số Default là một chuỗi, chúng ta có thể khởi tạo nó bằng bất kỳ giá trị nào mà lớp AnsiString có thể sử dụng trong kiến tạo của chúng. Có nghĩa là giá trị default có thể là một ký tự, một số thực hay một số nguyên:

```
//-----
void __fastcall TForm1::btnInputClick(TObject *Sender)
{
    InputBox( "Temperature Conversion",
             "Type the temperature in Celsius:", 102);
}
//-----
```

Nếu chúng ta chỉ định tham số Default và người sử dụng nhấn nút OK mà không thay đổi nội dung của hộp soạn thảo, trình dịch phải sẽ quyết định giá trị mặc định như giá trị hợp lệ

Sau khi sử dụng hộp thoại, người sử dụng sẽ nhấn OK, nhấn phím Enter, nhấn Cancel hay phím Esc. Nếu người sử dụng nhấn Ok hay nhấn phím Enter, hàm trả về giá trị mà người sử dụng đã gõ vào trong hộp soạn thảo. Điều này cho phép chúng ta viết một lệnh điều kiện để kiểm tra giá trị mới được trả về bởi việc nhấn nút Ok trên hộp thoại InputBox:

```
//-----
void __fastcall TForm1::btnInputClick(TObject *Sender)
{
    Edit1->Text = InputBox("Distance and Measurement",
                          "Enter the distance in kilometer:", 0.00);
}
//-----
```

//-----

Nếu người sử dụng nhấn Cancel hay phím Esc, những gì hiển thị trong hộp soạn thảo sẽ được bỏ qua. Nhưng hàm sẽ trả về giá trị mặc định được truyền vào tham số Default.

Nếu giá trị trả về dự định cho việc tính toán toán học, ngày, hay giờ, chúng ta phải chuyển nó sang dạng tương ứng.

Sử dụng hộp thoại InputBox:

Thêm một nút nhấn vào form.

Đổi tiêu đề của nó thành Input &Box và tên của nó thành btnInputBox.

Nhấn đôi chuột lên nút Input Box.

Nhấn Enter và gõ:

```
InputBox("Student Registration", "Type the Student's Gender", "");
```

Nhấn F9 để kiểm thử form. Nhấn nút InputBox. Chú ý rằng nội dung của hộp soạn thảo của nó là rỗng.

Đóng Input Box và form.

Cung cấp một giá trị mặc định cho người sử dụng, thay đổi nội dung của hàm trước như sau:

```
InputBox("Student Registration", "Type the Student's Gender", "Male");
```

Kiểm thử form và nhấn nút Input Box.

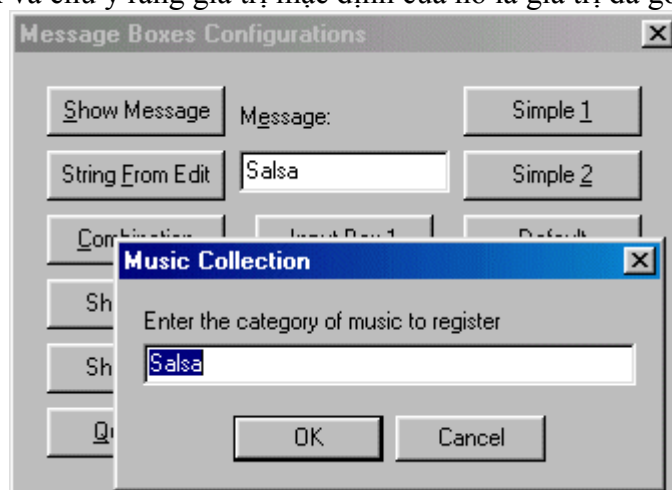
Để sử dụng nội dung của một đối tượng khác, chẳng hạn như một hộp soạn thảo, như giá trị mặc định, thay đổi đoạn mã trên như sau:

```
InputBox("Music Collection", "Enter the category of music to register", edtMessage->Text);
```

Để kiểm thử form, nhấn phím F9.

Gõ Salsa vào hộp soạn thảo Message.

Nhấn nút Input Box và chú ý rằng giá trị mặc định của nó là giá trị đã gõ vào hộp soạn thảo:



Hình 66-Hộp thoại InputBox có giá trị mặc định

Gõ Irish Folks và nhấn OK

Đóng form.

Để sử dụng và áp dụng kết quả của một Input Box vào một đối tượng khác, chẳng hạn, để dán nó vào trong một hộp soạn thảo, sửa đoạn mã trên thành:

```
edtMessage->Text = InputBox("Employees Records", "Employee's Department", "Human Resources");
```

Để kiểm thử form, nhấn F9.

Nhấn nút Input Box. n

Gõ Accounting và nhấn Enter. Chú ý rằng Accounting hiển thị trong hộp soạn thảo Message.

2. Hộp thoại Windows dùng chung

Các hộp thoại Windows dùng chung được cung cấp bởi C++ Builder bao gồm:



Hộp thoại mở tập tin: Dùng để mở các tập tin, được khai báo trong lớp TOpenDialog.



Hộp thoại lưu tập tin: Dùng để lưu tập tin, được khai báo trong lớp TSaveDialog.



Hộp thoại mở tập tin hình ảnh: Tương tự như hộp thoại mở tập tin nhưng dùng để mở các tập tin hình ảnh, được khai báo trong TOpenPictureDialog.



Hộp thoại lưu tập tin hình ảnh: Tương tự như hộp thoại lưu tập tin nhưng dùng để lưu các tập tin hình ảnh, được khai báo trong lớp TSavePictureDialog.



Hộp thoại chọn font chữ: Dùng để mở hộp thoại lựa chọn và điều chỉnh font chữ, được khai báo trong lớp TFontDialog.



Hộp thoại chọn màu: Dùng để mở hộp thoại chọn màu được khai báo trong lớp TColorDialog.



Hộp thoại lựa chọn máy in: Dùng để mở hộp thoại lựa chọn máy in được khai báo trong lớp TPrinterDialog.



Hộp thoại cài đặt máy in: Dùng để mở hộp thoại điều chỉnh các thông số về trang in được khai báo trong lớp TPrinterSetupDialog.



Hộp thoại tìm kiếm: Dùng để mở hộp thoại tìm kiếm văn bản, được khai báo trong lớp TFindDialog.



Hộp thoại tìm kiếm và thay thế: Dùng để mở hộp thoại tìm kiếm và thay thế, được khai báo trong lớp TReplaceDialog.

Các hộp thoại này được dùng cho những mục đích khác nhau. Mỗi lập trình viên đều có thể sử dụng các hộp thoại theo chức năng chương trình của mình.

3. Tạo và quản lý các hộp thoại

Trong mục trên, chúng ta đã xem qua giới thiệu sơ lược về các chức năng của các hộp thoại. Trong nội dung phần này, chúng ta sẽ nghiên cứu ngắn gọn về phương pháp sử dụng các hộp thoại đã liệt kê ở trên.

Trước khi đi vào nội dung chi tiết về việc sử dụng các hộp thoại, chúng ta cần phải hiểu được phương thức quan trọng sau:

Phương thức Execute(): Phương thức này trả về một trong hai giá trị TRUE hoặc FALSE tương ứng với việc đóng hộp thoại bằng cách sử dụng nút quyết định hay nút hủy bỏ; chẳng hạn nút OK (Open) được gọi là nút quyết định và nút Cancel gọi là nút hủy bỏ. Khi nhấn nút OK, hộp thoại sẽ được đóng lại và trả về giá trị TRUE; khi nhấn nút Cancel, hộp thoại sẽ được đóng lại và trả về giá trị FALSE.

Hộp thoại mở tập tin và hộp thoại lưu tập tin:

Các thuộc tính quan trọng của hộp thoại mở tập tin được liệt kê trong bảng sau:

Thuộc tính	Ý nghĩa
DefaultExt	Phần mở rộng của mặc định khi hộp thoại mở tập tin được mở ra
FileName	Tên tập tin được chọn trong hộp thoại mở tập tin trong hộp thoại mở tập tin
Files	Danh sách các tập tin được chọn trong hộp thoại mở tập tin
Filter	Bộ lọc các phần mở rộng tập tin
FilterIndex	Chỉ số mặc định của bộ lọc phần mở rộng tập tin
InitialDir	Thư mục mặc định được sử dụng khi hộp thoại tập tin được mở.
Options	Thuộc tính điều chỉnh các tùy chọn; chứa danh sách các tùy chọn được liệt kê dưới đây: ofReadOnly: Chọn mở ở chế độ chỉ đọc. ofHideReadOnly: Cho phép hay không cho phép hiển thị nút đánh dấu Open as Read-Only.

	<p>ofShowHelp: Hiện thị nút Help trên hộp thoại.</p> <p>ofNoValidate: Bỏ đánh dấu cho các ký tự không hợp lệ. Cho phép chọn một tập tin với các ký tự không hợp lệ.</p> <p>ofAllowMultiSelect: Cho phép người sử dụng chọn nhiều tập tin cùng một lúc.</p> <p>ofExtensionDifferent: Cờ này được bật vào lúc thực khi bất cứ khi nào chọn một tập tin có phần mở rộng khác với phần mở rộng mặc định DefaultExt. Nếu chúng ta sử dụng tùy chọn này, chúng ta cần phải cài đặt lại nó.</p> <p>ofPathMustExist: Phát sinh một thông báo lỗi nếu người sử dụng chọn một đường dẫn tập tin không tồn tại.</p> <p>ofFileMustExist: Phát sinh một thông báo lỗi nếu người sử dụng cố gắng chọn một tập tin không tồn tại.</p> <p>ofCreatePrompt: Phát sinh một cảnh báo nếu người sử dụng chọn một tập tin không tồn tại, hộp thoại sẽ hỏi tạo một tập tin mới với tên chỉ định.</p> <p>ofNoReadOnlyReturn: Phát sinh một thông báo lỗi nếu người sử dụng cố gắng chọn một tập tin chỉ đọc.</p> <p>ofEnableSizing: Cho phép hộp thoại có thể được thay đổi kích thước.</p> <p>ofDontAddToRecent: Cấm tập tin thêm vào danh sách các tập tin recently.</p> <p>ofShowHidden: Cho phép hiển thị các tập tin ẩn trong hộp thoại.</p>
Title	Tiêu đề của hộp thoại.

Các ví dụ việc sử dụng mã lệnh để điều khiển các thuộc tính:

- Thuộc tính DefaultExt:

Thuộc tính này không chấp nhận phần mở rộng nhiều hơn ba ký tự. Phần mở rộng được lưu trữ trong thuộc tính này không bao gồm dấu chấm phía trước.

```
SavePictureDialog1->DefaultExt = GraphicExtension(__classid(Graphics::TBitmap));
```

```
SavePictureDialog1->Filter = GraphicFilter(__classid(Graphics::TBitmap));
```

```
if (SavePictureDialog1->Execute())
```

```
    // save the graphic
```

- Thuộc tính FileName, Title:

Thuộc tính FileName trả về tập tin bao gồm cả đường dẫn.

Thuộc tính Title cho phép gán giá trị tiêu đề cho hộp thoại.

Chúng ta hãy xem xét các ví dụ sau:

Ví dụ 1: Đoạn mã lệnh sau mở nội dung tập tin và đưa vào một Memo:

```
if (OpenDialog1->Execute())
```

```
    Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

```
else
```

```
    Memo1->Clear();
```

Ví dụ 2: Đoạn mã lệnh sau là đoạn mã lệnh cho sự kiện OnClick của Button1 để xóa tập tin được chọn. Trong đoạn mã lệnh này, chúng ta sử dụng hàm FileExists để kiểm tra xem tập tin được chọn có tồn tại hay không rồi tiến hành xóa.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
```

```
    //Đổi tiêu đề của hộp thoại thành Delete File
```

```
    OpenDialog1->Title = "Delete File";
```

```
    if (OpenDialog1->Execute())
```

```
    {
```

```
        if (FileExists(OpenDialog1->FileName))
```

```
            DeleteFile(OpenDialog1->FileName);
```

```
    }
```

```
}
```

- Thuộc tính Files, Filter, FilterIndex:

Thuộc tính Files có kiểu TStrings chứa danh sách các tập tin.

Thuộc tính Filter có kiểu String chứa đoạn văn bản để thực hiện bộ lọc. Thuộc tính này có thể được điều chỉnh bằng cửa sổ Object Inspector.

Thuộc tính FilterIndex có kiểu là số nguyên, chỉ định chỉ số mặc định của thuộc tính Filter nhằm chọn Filter lựa chọn.

Thuộc tính Options liệt kê các tùy chọn tác động trên hộp thoại.

Chúng ta xem xét các ví dụ sau:

Ví dụ 1: Đoạn mã lệnh sau sẽ hiển thị tất cả các tập tin vào một ListBox:

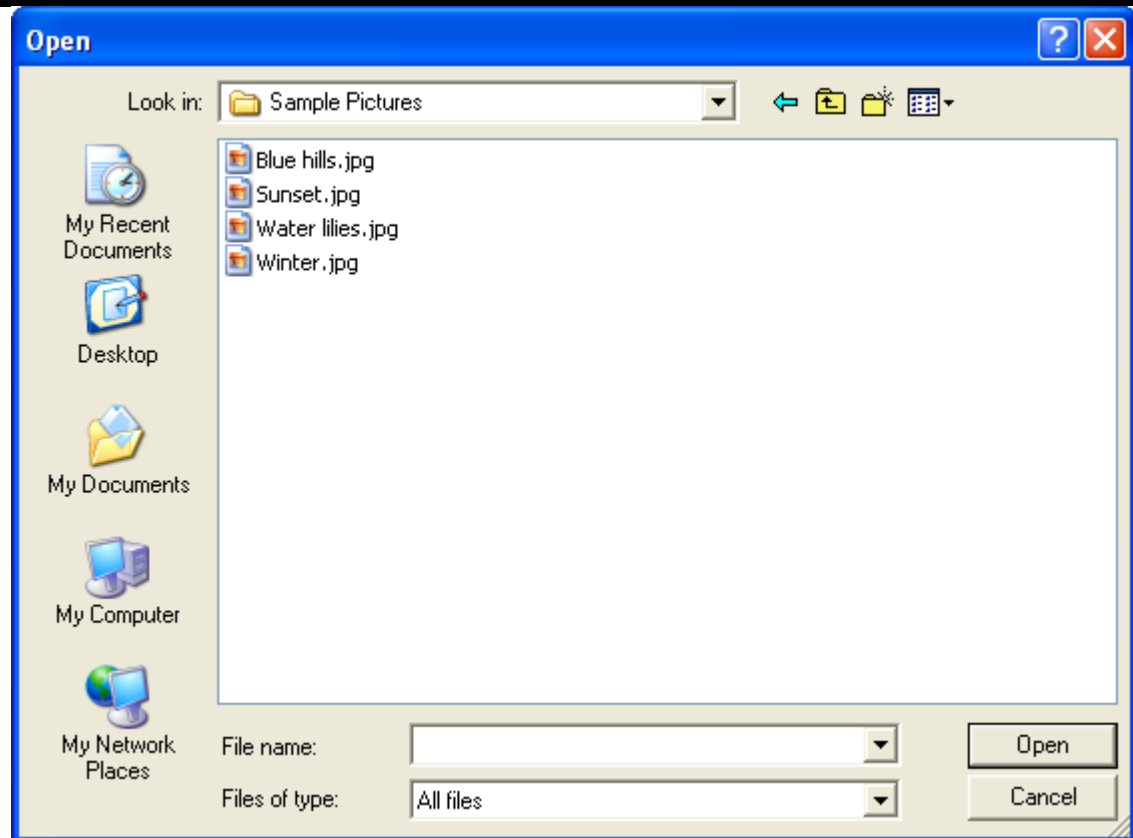
ListBox1->Items->Assign(OpenDialog1->Files);

Ví dụ 2: Đoạn mã lệnh sau dành cho sự kiện OnClick của nút Button1 và đọc dòng đầu tiên của các tập tin để liệt kê vào trong một ListBox.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
    FILE *stream;
    char FirstLine[512];
    OpenDialog1->Options.Clear();
    //cho phép chọn nhiều tập tin và kiểm tra tập tin được chọn phải tồn tại
    OpenDialog1->Options << ofAllowMultiSelect << ofFileMustExist;
    //tạo hai bộ lọc các tập tin *.txt và tập các tập tin bất kỳ
    OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
    //cho phép bộ lọc *.* có tác dụng khi mở hộp thoại
    OpenDialog1->FilterIndex = 2; // start the dialog showing all files
    //mở hộp thoại và kiểm tra xem nhấn nút Open hay không?
    if (OpenDialog1->Execute()) //nếu nhấn nút Open
    {
        //chạy một vòng lặp từ tập tin đầu tiên đến tập tin cuối cùng
        for (int I = 0; I < OpenDialog1->Files->Count; I++)
        {
            //mở một tập tin với chỉ số tương ứng
            stream = fopen(OpenDialog1->Files->Strings[I].c_str(), "r");

            if (stream)
            {
                // đọc dòng đầu tiên của tập tin
                fgets(FirstLine, sizeof(FirstLine), stream);
                //thêm dòng này đầu tiên vào Memo
                Memo1->Lines->Append(FirstLine);
                fclose(stream);
            }
        }
    }
}
```

Hình 67-Hình minh họa hộp thoại mở tập tin

- Hộp thoại Font:

Hộp thoại Font cho phép chúng ta chọn Font chữ, kích cỡ, kiểu chữ. Các thuộc tính quan trọng hộp thoại Font:

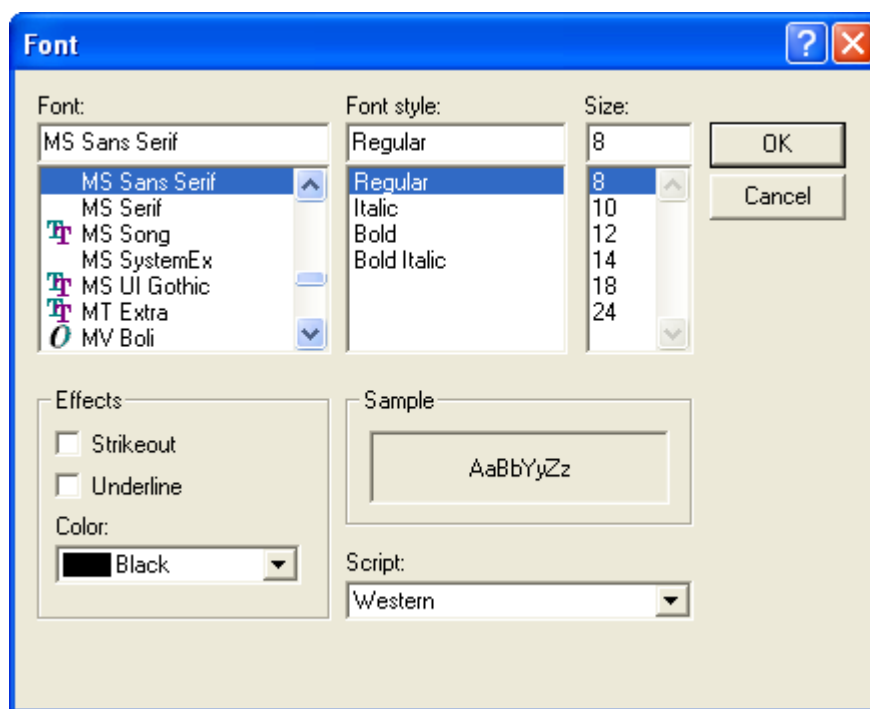
Thuộc tính	Ý nghĩa
Charset	Chỉ định tập ký tự
Color	Màu của font chữ
Handle	Thẻ quản của hộp thoại
Height	Chiều cao của font chữ. Chúng ta có thể tính toán sử dụng công thức này: $\text{Font.Height} = -\text{Font.Size} * \text{Font.PixelsPerInch} / 72$
Name	Tên của font chữ
Pitch	Chỉ định độ rộng của các ký tự có kích thước giống nhau hay không?
PixelsPerInch	Số điểm trên một inch. Nếu thay đổi sẽ ảnh hưởng đến thuộc tính Height và Size
Size	Kích thước font chữ
Style	Kiểu dáng font chữ

Đây là ví dụ về việc cài đặt thuộc tính FontStyle của Font chữ của Label thành fsBold và fsUnderline:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    Label1->Font->Style = TFontStyles() << fsBold << fsUnderline;
}
```

Ví dụ sau cho thấy làm thế nào để hủy bỏ tất cả các kiểu dáng của font chữ:

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Label1->Font->Style = TFontStyles();
}
```



Hình 68-Hình minh họa hộp thoại font chữ

- Hộp thoại Color:

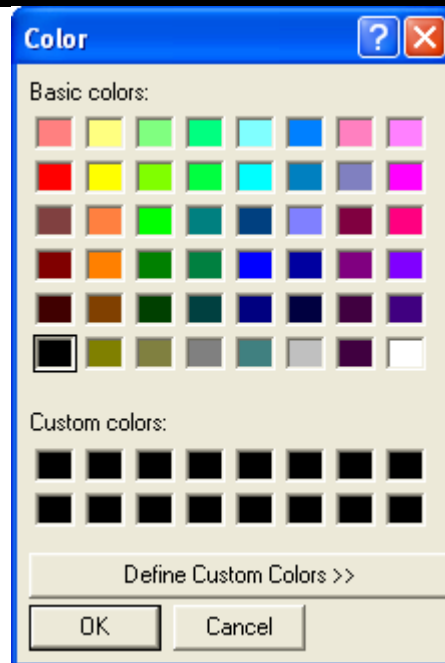
Hộp thoại này cho phép chọn màu chữ. Các thuộc tính thông dụng được liệt kê sau đây:

Thuộc tính	Ý nghĩa
Color	Trả về giá trị màu được chọn có kiểu TColor
CustomColors	Chứa đựng các giá trị màu tùy chọn ở dạng giá trị cơ số 16.
Options	<p>cdFullOpen: Hiển thị các màu tùy chọn khi hộp thoại được mở ra.</p> <p>cdPreventFullOpen: Hủy các nút định nghĩa màu mới trong hộp thoại, người sử dụng không thể định nghĩa các màu mới.</p> <p>cdShowHelp: Thêm nút Help vào hộp thoại.</p> <p>cdSolidColor: Trực tiếp Windows sử dụng một màu đặt (solid color) gần nhất so với màu được chọn.</p> <p>cdAnyColor: Cho phép người sử dụng chọn các màu không đặt (non-solid color), những màu có thể hiển thị tương xứng bởi sự phối màu.</p>

Đoạn mã lệnh sau của sự kiện OnClick trên nút Button1 để mở hộp thoại màu và đổi màu của hình vẽ Shape1.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
  if (ColorDialog1->Execute())
    Shape1->Brush->Color = ColorDialog1->Color;
}
```



Hình 69-Hình minh họa hộp thoại font chữ

- Hộp thoại Printer:

Hộp thoại này cho phép chọn máy in để có thể từ đó in dữ liệu ra máy in. Thuộc tính quan trọng Printer được liệt kê trong bảng sau:

Thuộc tính	Ý nghĩa
Collate	Trả về giá trị cho biết nút đánh dấu Collate có được đánh dấu hay không?
Copies	Trả về số lượng bản in
FromPage	Chỉ định trang bắt đầu
MaxPage	Trả về số lượng trang lớn nhất các trang sẽ in
MinPage	Trả về số lượng trang nhỏ nhất các trang sẽ in
Options	Các tùy chọn khi in, là tập các giá trị sau: poDisablePrintToFile: Hủy bỏ nút đánh dấu Print To File. poHelp Displays: Hiển thị nút Help trong hộp thoại. poPageNums: Cho phép nút radio Pages được sử dụng để chỉ định vùng vùng trang in. poPrintToFile: Hiển thị nút đánh dấu Print To File trong hộp thoại. poSelection: Cho phép nút chọn Selection cho phép người sử dụng in văn bản được chọn. poWarning: Phát sinh một thông báo cảnh báo nếu người sử dụng cố gắng gửi một lệnh để gỡ bỏ máy in.
PrintRange	Là kiểu liệt kê cho phép chọn vùng trang in: prAllPages: Nút radio All được chọn. prSelection: Nút radio Selection được chọn. prPageNums: Nút radio Pages được chọn.
PrintToFile	Nút đánh dấu Print To File được chọn hay không.
ToPage	Chỉ định trang in kết thúc

Một số ví dụ thể hiện một số áp dụng điều chỉnh và sử dụng hộp thoại PrinterDialog.

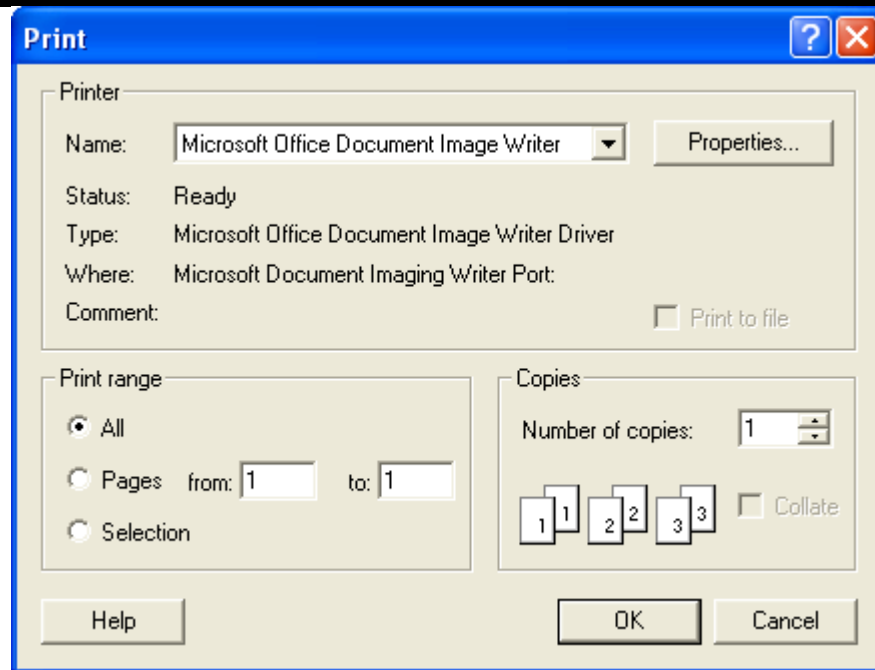
Ví dụ này sử dụng một nút nhấn, một Page control và một hộp thoại Print trên một form. Khi người sử dụng nhấn nút, hộp thoại printer sẽ hiển thị. Người sử dụng có thể chọn bất kỳ tập con

các trang để in nội dung trên Page Control (Đối tượng này nằm trong ngăn Win32). Để thực hiện được ví dụ này, chúng ta phải khai báo thêm thư viện <Printers.hpp>:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    PrintDialog1->Options.Clear();
    //hiển thị Page Number và Selection
    PrintDialog1->Options << poPageNums << poSelection;
    //trang đầu tiên là 1
    PrintDialog1->FromPage = 1;
    //số lượng trang ít nhất là 1
    PrintDialog1->MinPage = 1;
    //tổng số lượng trang in là trang cuối
    PrintDialog1->ToPage = PageControl1->PageCount;
    //số lượng trang lớn nhất là tổng số trang.
    PrintDialog1->MaxPage = PageControl1->PageCount;
    if (PrintDialog1->Execute())
    {
        int Start, Stop;
        // kiểm tra xem vùng mà người sử dụng muốn in
        switch (PrintDialog1->PrintRange)
        {
            case prSelection:

                Start = PageControl1->ActivePage->PageIndex;
                Stop = Start;
                break;
            case prPageNums:
                Start = PrintDialog1->FromPage - 1;
                Stop = PrintDialog1->ToPage - 1;
                break;
            default: // prAllPages
                Start = PrintDialog1->MinPage - 1;
                Stop = PrintDialog1->MaxPage - 1;
                break;
        }
        // bây giờ, in các trang
        Printer()->BeginDoc();
        for (int i = Start; i <= Stop; i++)

        {
            PageControl1->Pages[i]->PaintTo(Printer()->Handle, 10, 10);
            if (i != Stop)
                Printer()->NewPage();
        }
        Printer()->EndDoc();
    }
}
```



Hình 70-Hình minh họa hộp thoại font chữ

- Hộp thoại tìm kiếm:

Thuộc tính	Ý nghĩa
FindText	Văn bản cần tìm kiếm
Options	Các tùy chọn tìm kiếm. Có thể tra cứu thêm trong tài liệu hướng dẫn.
Position	Vị trí hiển thị hộp thoại

Ví dụ sau đây yêu cầu một TRichEdit, một nút nhấn và một hộp thoại TFindDialog.

Nhấn lên nút nhấn sẽ hiển thị một hộp thoại tìm kiếm trên góc phải của rich edit. Điền văn bản tìm kiếm vào ô Find what và nhấn nút Find Next sẽ chọn chuỗi trung đầu tiên trong Rich Edit. Trong ví dụ này, chúng ta có sử dụng sự kiện OnFind:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
    FindDialog1->Position = Point(RichEdit1->Left + RichEdit1->Width, RichEdit1->Top);
    FindDialog1->Execute();
}
```

```
void __fastcall TForm1::FindDialog1Find(TObject *Sender)
```

```
{
    int FoundAt, StartPos, ToEnd;
    // begin the search after the current selection
    // if there is one
    // otherwise, begin at the start of the text
    if (RichEdit1->SelLength)
        StartPos = RichEdit1->SelStart + RichEdit1->SelLength;
    else
```

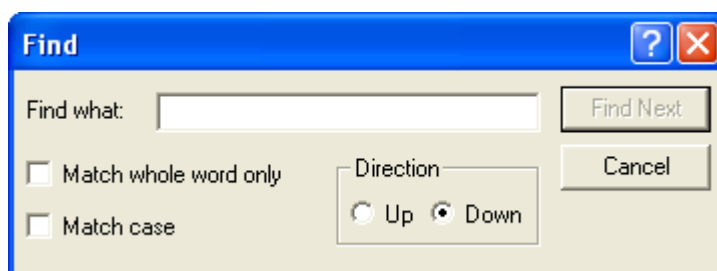
```
    StartPos = 0;
```

```
    // ToEnd is the length from StartPos
```

```
// to the end of the text in the rich edit control
```

```
ToEnd = RichEdit1->Text.Length() - StartPos;
```

```
FoundAt = RichEdit1->FindText(FindDialog1->FindText, StartPos, ToEnd,
TSearchTypes() << stMatchCase);
if (FoundAt != -1)
{
    RichEdit1->SetFocus();
    RichEdit1->SelStart = FoundAt;
    RichEdit1->SelLength = FindDialog1->FindText.Length();
}
}
```



Hình 71-Hình minh họa tìm kiếm

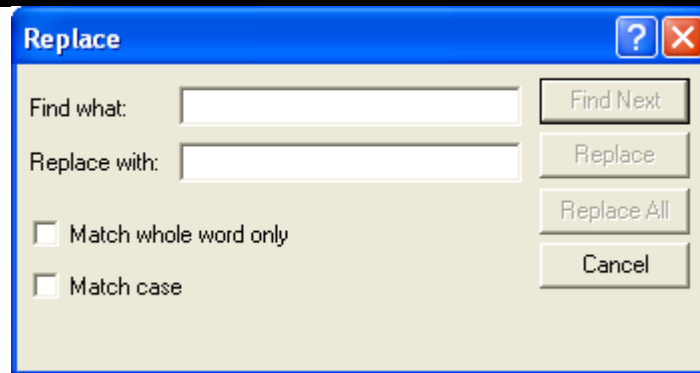
- Hộp thoại ReplaceDialog:

Trong hộp thoại này, chúng ta được gộp thêm thuộc tính ReplaceText là văn bản thay thế. Ngoài ra còn bổ sung thêm sự kiện OnReplace để thay thế văn bản. Sau đây là ví dụ về việc sử dụng sự kiện OnReplace:

Đoạn mã lệnh sau tìm kiếm một đối tượng TMemo gọi là Memo1 và thay thế văn bản FindText với ReplaceText. Nó sử dụng thuộc tính SelStart, SelLength và SelText.

```
void __fastcall TForm1::ReplaceDialog1Replace(TObject *Sender)
```

```
{
    TReplaceDialog *dlg = (TReplaceDialog *)Sender;
    /* perform a global case-sensitive search for FindText in Memo1 */
    int SelPos = Memo1->Lines->Text.Pos(dlg->FindText);
    if (SelPos > 0)
    {
        Memo1->SelStart = SelPos - 1;
        Memo1->SelLength = dlg->FindText.Length();
        // Replace selected text with ReplaceText
        Memo1->SelText = dlg->ReplaceText;
    }
    else
        MessageBeep(0);
}
```



Hình 72-Hình minh họa tìm kiếm và thay thế

Hộp thoại PrinterSetupDialog chúng ta có thể nghiên cứu thêm trong tài liệu hướng dẫn.

4. Sử dụng đa biểu mẫu

Trong nội dung này, chúng ta sẽ nghiên cứu phương pháp sử dụng đa biểu mẫu trong ứng dụng. Đầu tiên, chúng ta đề cập đến thuộc tính `FormStyle` và trải dài qua suốt nội dung phần này các ví dụ thực hiện theo từng bước.

Thuộc tính `FormStyle` cho phép chúng ta chọn giữa `fsNormal` (Form bình thường) và các thuộc tính dùng để tạo ứng dụng nhiều tài liệu (MDI-Multiple Document Interface).

Chúng ta có thể đặt giá trị thành `fsStayOnTop` để Form trở thành Form luôn luôn nằm trên tất cả các Form khác, đây là một loại Form rất đặc biệt mà Windows chỉ cho phép duy nhất một Form được ở trạng thái Stay On Top. Do đó, chúng ta không thể tạo hai form cùng StayOnTop vào một thời điểm.

Ví dụ: Tạo ứng dụng MDI theo từng bước. MDI là một loại ứng dụng trong các hệ điều hành cũ. Tuy nhiên, ứng dụng này vẫn mang một tính chất rất hay.

Cấu trúc MDI cho phép một ứng dụng có nhiều Form được quản lý qua một Form cha. Thường được quản lý bởi một thực đơn trong ứng dụng. Về mặt kỹ thuật chúng ta có khái niệm sau:

- Một cửa sổ chính của ứng dụng giống như một thùng chứa để chứa các Form con.
- Một Form đặc biệt gọi là MDI Client, vỏ bọc của toàn bộ vùng hoạt động của ứng dụng.
- Có nhiều form con có cùng kiểu hay khác kiểu, mỗi Form sẽ hiển thị trong vùng làm việc của Form.

C++ Builder cho phép phát triển các ứng dụng MDI rất dễ dàng, chúng ta phải tạo ít nhất hai form, một form được cài đặt thuộc tính `FormStyle` là `fsMDIForm` (Form cha) và một form được cài đặt thuộc tính `fsMDIChild` (Form con).

Khi đó, chúng ta thêm mã lệnh cho chức năng New của thực đơn File như sau (TChildForm là lớp của form con mang tên ChildForm):

```
TChildForm ChildForm;
ChildForm=new TChildForm(Application);
```

Một thành phần quan trọng của ứng dụng MDI là gắn kết một thực đơn Windows cho tất cả các cửa sổ bằng thuộc tính `WindowsMenu` của form chính.

Ngoài ra, ứng dụng MDI sẽ gộp menu của các Menu của các cửa sổ con vào cửa sổ chính khi chúng ta sử dụng bằng cách điều chỉnh các thuộc tính `GroupIndex` của các mục chọn trên menu. Chẳng hạn, các mục chọn của menu chính có `GroupIndex =0`, cửa sổ con 1 có thuộc tính `GroupIndex=1`.

```
void TMainForm::New1Click(TObject *Sender)
TChildForm *ChildForm;
{
    Counter++;
    ChildForm=new TChildForm(Application);
    ChildForm->Caption = ChildForm->Caption + " " + IntToStr(Counter);
}
```

Chúng ta tiến hành xây dựng menu hoàn chỉnh như sau:

Đầu tiên là thực đơn Windows có ít nhất ba thành phần chọn với tiêu đề Cascade, Tile và Arrange Icons. Chúng ta có thể sử dụng một số phương thức dựng sẵn của TForm chỉ để phục vụ cho chương trình MDI:

- Phương thức Cascade xếp các cửa sổ con theo tầng, cửa sổ này chồng lớp.
- Phương thức Tile sắp xếp các cửa sổ không chồng lớp theo chiều ngang hay chiều đứng.
- Phương thức ArrangeIcons sẽ sắp xếp các cửa sổ con theo kiểu biểu tượng.

Chúng ta sử dụng kết hợp với TActionList để thêm các hành động cho từng cửa sổ thông qua thuộc tính Action bằng các sử dụng các hành động dựng sẵn.

Chúng ta cũng hiểu một số thuộc tính:

- ActiveMDIChild là một thuộc tính chỉ đọc cho phép truy cập form MDI đang được kích hoạt. Thuộc tính này có thể bị thay đổi khi chúng ta dùng các phương thức Next, Previous để di chuyển đến Form con kế tiếp hoặc form con trước đó.

- ClientHandle là thuộc tính nắm giữ thẻ quản của cửa sổ con đang được mở trong vùng làm việc của form chính.

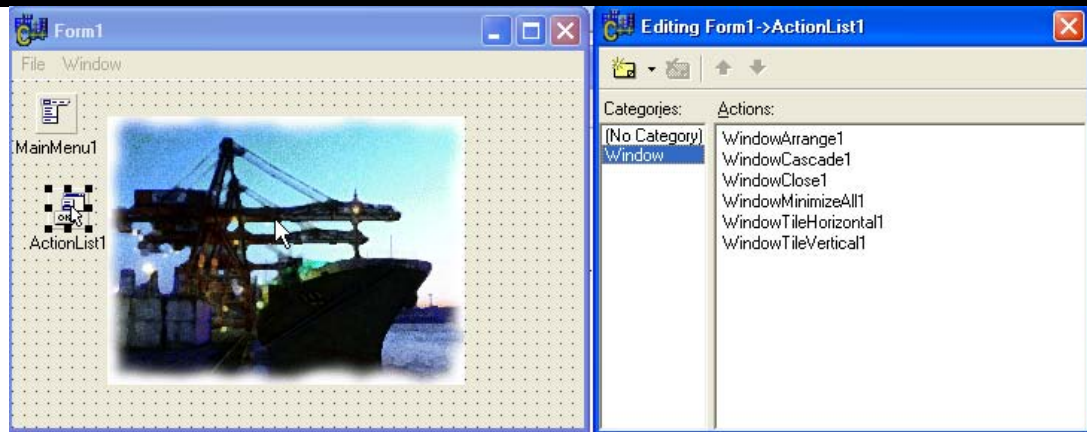
- MDIChildren là thuộc tính chứa một mảng các form con. Chúng ta có thể sử dụng thuộc tính MDIChildCount để tính xem có bao nhiêu cửa sổ con.

Bây giờ chúng ta tạo ứng dụng MDI theo từng bước sau:

- Bước 1: Tạo một dự án mới.
- Bước 2: Đặt tên cho Form1 thành MainForm.
- Bước 3: Lưu tên tập tin Unit1 thành mdlMain và tên Project thành ProMDI.
- Bước 4: Điều chỉnh và thêm các thành phần như trong cửa sổ sau:

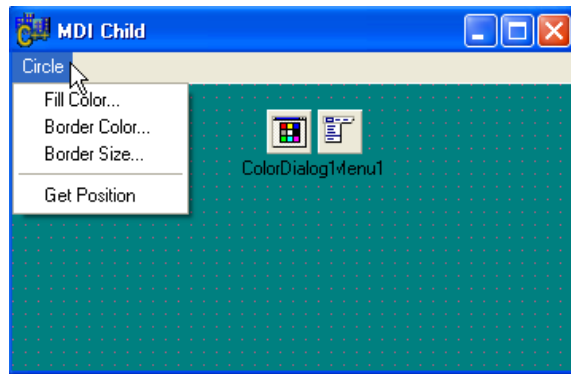
Đối tượng	Thuộc tính	Giá trị
TMainMenu	Các mục con của menu File Menu File có thuộc tính GroupIndex=0	New &Circle New &Bouncing Square Close Close &All - &Exit
	Các mục con của menu Window gắn kết với hành động tạo trong ActionList1 thông qua các thuộc tính Action như sau: Menu Windows có thuộc tính GroupIndex=0	WindowArrange1 WindowCascade1 WindowClose1 WindowMinimizeAll1 WindowTileHorizontal1 WindowTileVertical1
	Tạo thêm mục con cho menu Windows bằng cách thêm một mục chọn Menu Windows có thuộc tính GroupIndex=0	Count
MainForm	FormStyle	fsMDIForm
ActionList1	Tạo các sự kiện chuẩn của Windows.	

Đồng thời, chúng ta thêm một đối tượng Image (C:\Program Files\Common Files\CodeGear Shared\Images\Splash\256Color\Shipping.BMP) để đặt được form như sau:



Hình 73- Thiết kế cửa sổ chính

Bước 5: Thêm một Form mới, cài đặt thuộc tính Name thành CircleChildForm, thuộc tính FormStyle thành fsMDIChild, thuộc tính Color thành màu clTeal, Caption thành MDI Child. Lưu tập tin với tên mdlCircle đồng thời thêm một Menu (có GroupIndex = 1) và một hộp thoại ColorDialog để đặt được cửa sổ như sau:



Hình 74- Form con Circle

Bước 6: Khai báo thêm các biến như sau vào ngăn private của tập tin .h:

```
int    XCenter, YCenter;
int    BorderSize;
TColor  BorderColor, FillColor;
Tập tin .h như sau:
#ifndef mdlCircleH
#define mdlCircleH
// -----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Dialogs.hpp>
#include <Menus.hpp>

// -----
class TCircleChildForm : public TForm {
__published: // IDE-managed Components
    TMainMenu *MainMenu1;
    TColorDialog *ColorDialog1;
    TMenuItem *Circle1;
    TMenuItem *FillColor1;
    TMenuItem *BorderColor1;
    TMenuItem *BorderSize1;
    TMenuItem *N1;
```

```

TMenuItem *GetPosition1;

void __fastcall FillColor1Click(TObject *Sender);
void __fastcall BorderColor1Click(TObject *Sender);
void __fastcall BorderSize1Click(TObject *Sender);
void __fastcall GetPosition1Click(TObject *Sender);
void __fastcall FormCreate(TObject *Sender);
void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button, TShiftState
Shift,
    int X, int Y);
void __fastcall FormPaint(TObject *Sender);
void __fastcall FormClose(TObject *Sender, TCloseAction &Action);private: // User
declarations
    int XCenter, YCenter;
    int BorderSize;
    TColor BorderColor, FillColor;
public: // User declarations
    __fastcall TCircleChildForm(TComponent* Owner);
};

// -----
extern PACKAGE TCircleChildForm *CircleChildForm;
// -----
#endif

```

Bước 7: Viết mã lệnh cho các chức năng trên Menu như sau:

- Fill Color:

```

void __fastcall TCircleChildForm::FillColor1Click(TObject *Sender)
{
    ColorDialog1->Color = FillColor;
    if(ColorDialog1->Execute() )
    {
        FillColor = ColorDialog1->Color;
        this->Repaint();
    }
}

```

- Border Color:

```

void __fastcall TCircleChildForm::BorderColor1Click(TObject *Sender)
{
    ColorDialog1->Color = BorderColor;
    if(ColorDialog1->Execute() )
    {
        BorderColor = ColorDialog1->Color;
        this->Repaint();
    }
}

```

- BorderSize:

```

void __fastcall TCircleChildForm::BorderSize1Click(TObject *Sender)
{
    UnicodeString InputString;
    InputString = IntToStr (BorderSize);
    if(InputQuery ("Border", "Insert width", InputString) )

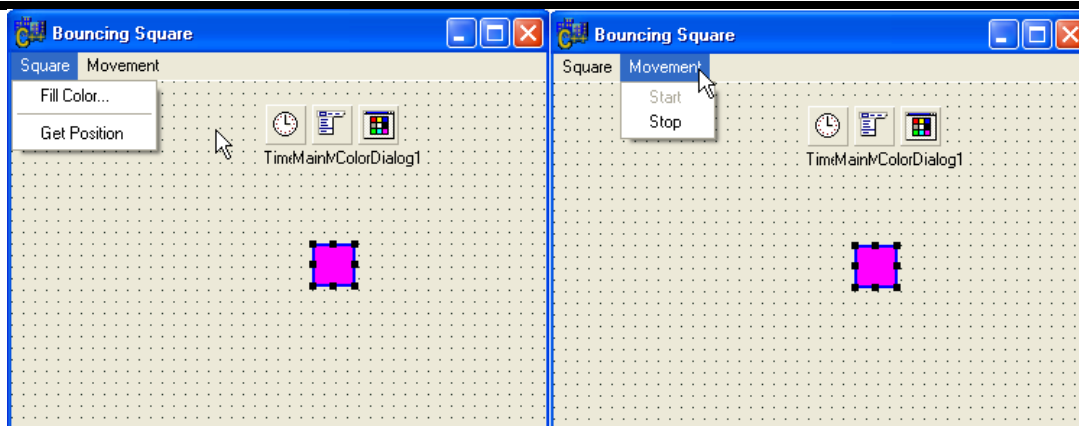
```

```

    {
        BorderSize = StrToIntDef (InputString, BorderSize);
        Repaint();
    }
}
- GetPosition:
MessageDlg ("The center of the circle is in the position (" +
    IntToStr (XCenter) + ", " + IntToStr (YCenter) + ").",
    mtInformation, TMsgDlgButtons() << mbOK, 0);
- Sự kiện Form Create:
void __fastcall TCircleChildForm::FormCreate(TObject *Sender)
{
    XCenter = - 200;
    YCenter = - 200;
    BorderSize = 1;
    BorderColor = clBlack;
    FillColor = this->Color;
}
- Sự kiện Mouse Down của Form:
void __fastcall TCircleChildForm::FormMouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    XCenter = X;
    YCenter = Y;
    Refresh();
}
- Sự kiện OnPaint của Form:
void __fastcall TCircleChildForm::FormPaint(TObject *Sender)
{
    Canvas->Pen->Width = BorderSize;
    Canvas->Pen->Color = BorderColor;
    Canvas->Brush->Color = FillColor;
    //Vẽ hình Ellipse tại vị trí nhấn chuột
    Canvas->Ellipse (XCenter-30, YCenter-30, XCenter+30, YCenter+30);
}
- Sự kiện OnClose của Form:
void __fastcall TCircleChildForm::FormClose(TObject *Sender,
    TCloseAction &Action)
{
    Action = caFree;
}

```

Bước 6: Thêm một Form mới, cài đặt thuộc tính Name thành BounceChildForm, thuộc tính FormStyle thành fsMDIChild, thuộc tính Color thành màu clTeal, Caption thành Bouncing Square. Lưu tập tin với tên mdlBounce đồng thời thêm một Menu (có GroupIndex = 2), một Timer (ngăn Win32) với thuộc tính Interval là 50, một Shape và một ColorDialog để đạt được cửa sổ như sau:



Hình 75-Thiết kế cửa sổ con thứ 2

Viết mã lệnh cho các sự kiện của các đối tượng trên form như sau:

- Khai báo để đạt được đoạn mã lệnh của tập tin .h như sau:

```
#ifndef mdlBounceH
#define mdlBounceH
// -----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>

// -----
enum directions {
    up_right, down_right, down_left, up_left
};
class TBounceChildForm : public TForm {
__published: // IDE-managed Components
    TMainMenu *MainMenu1;
    TMenuItem *Square1;
    TMenuItem *FillColor1;
    TMenuItem *N1;
    TMenuItem *GetPosition1;
    TMenuItem *Movement1;
    TMenuItem *Start1;
    TMenuItem *Stop1;
    TColorDialog *ColorDialog1;
    TTimer *Timer1;
    TShape *Shape1;

private: // User declarations
    directions dir;

public: // User declarations
    __fastcall TBounceChildForm(TComponent* Owner);
};
extern PACKAGE TBounceChildForm *BounceChildForm;
// -----
#endif
```

- Lệnh FillColor:

```
void __fastcall TBounceChildForm::FillColor1Click(TObject *Sender)
{
    if(ColorDialog1->Execute() )
        Shape1->Brush->Color = ColorDialog1->Color;
}
```

- Lệnh Get Position:

```
void __fastcall TBounceChildForm::GetPosition1Click(TObject *Sender)
{
    MessageDlg ("The top-left corner of the square was in the position (" +
        IntToStr (Shape1->Left) + ", " + IntToStr (Shape1->Top) + ")->",
        mtInformation, TMsgDlgButtons() << mbOK, 0);
}
```

- Lệnh Start:

```
void __fastcall TBounceChildForm::Start1Click(TObject *Sender)
{
    Timer1->Enabled = true;
    Start1->Enabled = false;
    Stop1->Enabled = true;
}
```

- Lệnh Stop:

```
void __fastcall TBounceChildForm::Stop1Click(TObject *Sender)
{
    Timer1->Enabled = false;
    Start1->Enabled = true;
    Stop1->Enabled = false;
}
```

- Sự kiện OnTimer của Timer1:

```
void __fastcall TBounceChildForm::Timer1Timer(TObject *Sender) {
    switch(dir) {
        case up_right: {
            Shape1->Left = Shape1->Left + 3;
            Shape1->Top = Shape1->Top - 3;
            if (Shape1->Top <= 0)
                dir = down_right;
            if (Shape1->Left + Shape1->Width >= ClientWidth)
                dir = up_left;
            break;
        }
        case down_right: {
            Shape1->Left = Shape1->Left + 3;
            Shape1->Top = Shape1->Top + 3;
            if (Shape1->Top + Shape1->Height >= ClientHeight)
                dir = up_right;
            if (Shape1->Left + Shape1->Width >= ClientWidth)
                dir = down_left;
            break;
        }
        case down_left: {
            Shape1->Left = Shape1->Left - 3;
            Shape1->Top = Shape1->Top + 3;
            if (Shape1->Top + Shape1->Height >= ClientHeight)
                dir = up_left;
```

```

        if (Shape1->Left <= 0)
            dir = down_right;
        break;
    }
    case up_left: {
        Shape1->Left = Shape1->Left - 3;
        Shape1->Top = Shape1->Top - 3;
        if (Shape1->Top <= 0)
            dir = down_left;
        if (Shape1->Left <= 0)
            dir = up_right;
    }
}

```

- Mã lệnh cho sự kiện OnCreate của Form:

```

void __fastcall TBounceChildForm::FormCreate(TObject *Sender)
{
    ColorDialog1->Color = Shape1->Brush->Color;
    dir = down_left;
}

```

- Mã lệnh cho sự kiện OnClose của Form:

```

void __fastcall TBounceChildForm::FormClose(TObject *Sender,
    TCloseAction &Action)
{
    Action = caFree;
}

```

Bước 7: Thêm các khai báo thư viện để Form chính (MainForm) nhận ra các form con bằng cách mở Form chính và chọn File/Use Unit..., chọn mdlCircle và mdlBounce, nhấn OK.

Bước 8: Viết mã lệnh cho các sự kiện của các đối tượng và sự kiện trên Form chính.

- Khai báo các biến thành viên của MainForm như sau:

```

int Count;
TCanvas *OutCanvas;
WNDPROC OldWinProc;
void *NewWinProc;
void __fastcall NewWinProcedure(TMessage &Msg);

```

Khi đó tập tin thư viện của Form sẽ là:

```

#ifndef mdlMainH
#define mdlMainH
// -----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <ActnList.hpp>
#include <StdActns.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include "mdlCircle.h"
#include "mdlBounce.h"

// -----
class TMainForm : public TForm {

```

```

__published: // IDE-managed Components
    TMainMenu *MainMenu1;
    TMenuItem *File1;
    TMenuItem *NewCircle1;
    TMenuItem *NewBouncingSquare1;
    TMenuItem *Close1;
    TMenuItem *CloseAll1;
    TMenuItem *N1;
    TMenuItem *Exit1;
    TActionList *ActionList1;
    TWindowClose *WindowClose1;
    TWindowCascade *WindowCascade1;
    TWindowTileHorizontal *WindowTileHorizontal1;
    TWindowTileVertical *WindowTileVertical1;
    TWindowMinimizeAll *WindowMinimizeAll1;
    TWindowArrange *WindowArrange1;
    TMenuItem *Windows1;
    TMenuItem *Arrange1;
    TMenuItem *Cascade1;
    TMenuItem *Close2;
    TMenuItem *ileHorizontally1;
    TMenuItem *ileVertically1;
    TMenuItem *Close3;
    TMenuItem *Count1;
    TImage *Image1;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall NewCircle1Click(TObject *Sender);

private: // User declarations
    int Count;
    TCanvas *OutCanvas;
    WNDPROC OldWinProc;
    void *NewWinProc;
    void __fastcall NewWinProcedure(TMessage &Msg);

public: // User declarations
    __fastcall TMainForm(TComponent* Owner);
};

// -----
extern PACKAGE TMainForm *MainForm;
// -----
#endif
- Thêm phương thức: NewWinProcedure để quản lý mỗi khi nền của Form được vẽ lại sẽ hiển
thị và vẽ hình trong Image1 làm hình nền:
// -----
void __fastcall TMainForm::NewWinProcedure(TMessage & Msg) {
    // TODO: Add your source code here
    int BmpWidth, BmpHeight;
    int I, J;

    // gọi hàm xử lý mặc định

```

```
Msg.Result = CallWindowProc(OldWinProc, ClientHandle, Msg.Msg, Msg.WParam,
Msg.LParam);
```

```
// Xử lý thông điệp vẽ nền
if (Msg.Msg == WM_ERASEBKGD) {
    BmpWidth = MainForm->Image1->Width;
    BmpHeight = MainForm->Image1->Height;
    if ((BmpWidth != 0) && (BmpHeight != 0)) {
        // vẽ ảnh làm nền
        OutCanvas->Handle = (HDC)Msg.WParam;
        for (I = 0; I <= MainForm->ClientWidth / BmpWidth; I++)
            for (J = 0; J <= MainForm->ClientHeight / BmpHeight; J++)
                OutCanvas->Draw(I * BmpWidth, J * BmpHeight,
                MainForm->Image1->Picture->Graphic);
    }
}
```

- Viết mã lệnh cho sự kiện OnCreate của Form:

```
void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    // tạo cài đặt mới cho hàm vẽ hình nền
    NewWinProc = MakeObjectInstance((TWndMethod) & NewWinProcedure);
    // lưu trữ lại hàm xử lý cũ
    OldWinProc = (WNDPROC)SetWindowLong(ClientHandle, GWL_WNDPROC, (long)
        NewWinProc);
    OutCanvas = new TCanvas();
    Count = 0;
}
```

- Viết mã lệnh cho lệnh New Circle:

```
void __fastcall TMainForm::NewCircle1Click(TObject *Sender) {
    TCircleChildForm *ChildForm;
    Count++;
    ChildForm = new TCircleChildForm(Application);
    ChildForm->Caption = ChildForm->Caption + " " + IntToStr(Count);
}
```

- Viết mã lệnh cho lệnh New Bouncing Square:

```
void __fastcall TMainForm::NewBouncingSquare1Click(TObject *Sender) {
    TBounceChildForm *ChildForm;
    Count++;
    ChildForm = new TBounceChildForm(Application);
    ChildForm->Caption = ChildForm->Caption + " " + IntToStr(Count);
}
```

- Lệnh Close gắn kết với hành động WindowClose1.

- Viết mã lệnh cho lệnh Close All:

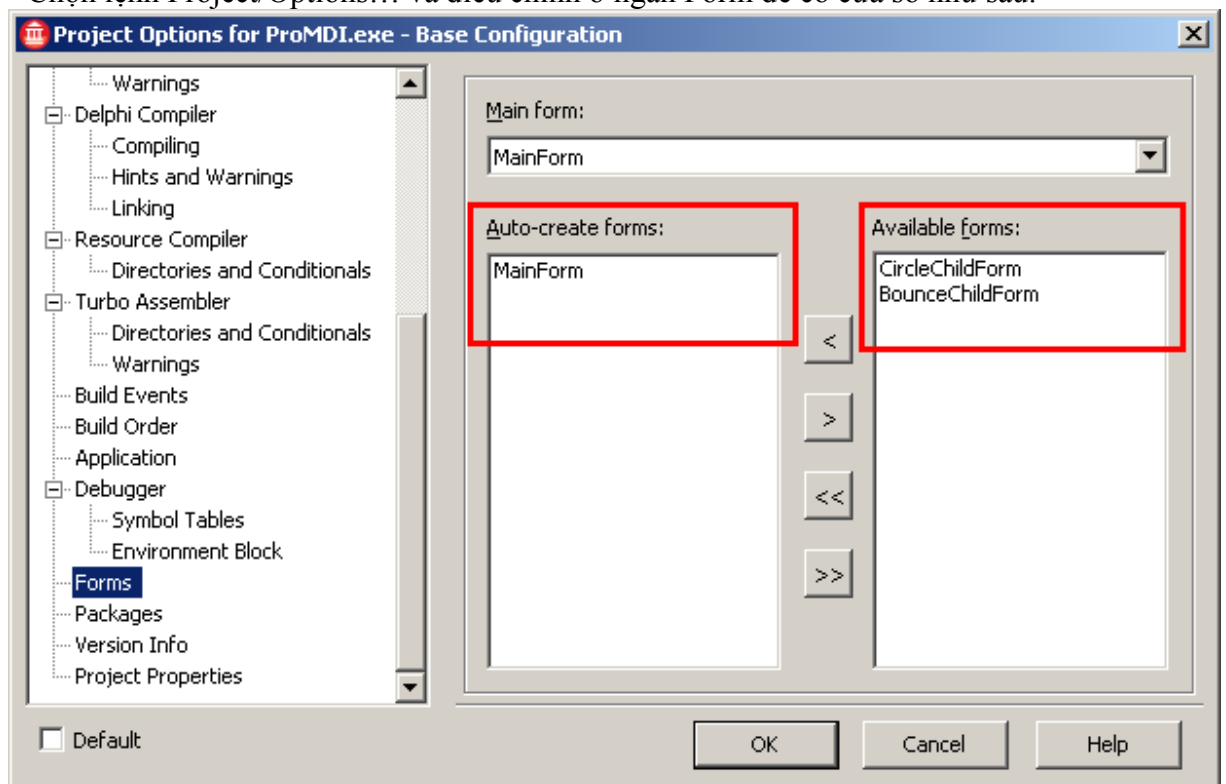
```
void __fastcall TMainForm::CloseAll1Click(TObject *Sender)
{
    int I;
    for( I = 0; I <= MDIChildCount - 1; I++)
        this->MDIChildren[I]->Close();
}
```

- Viết mã lệnh cho lệnh Exit:

```
void __fastcall TMainForm::Exit1Click(TObject *Sender)
{
    // ...
}
```



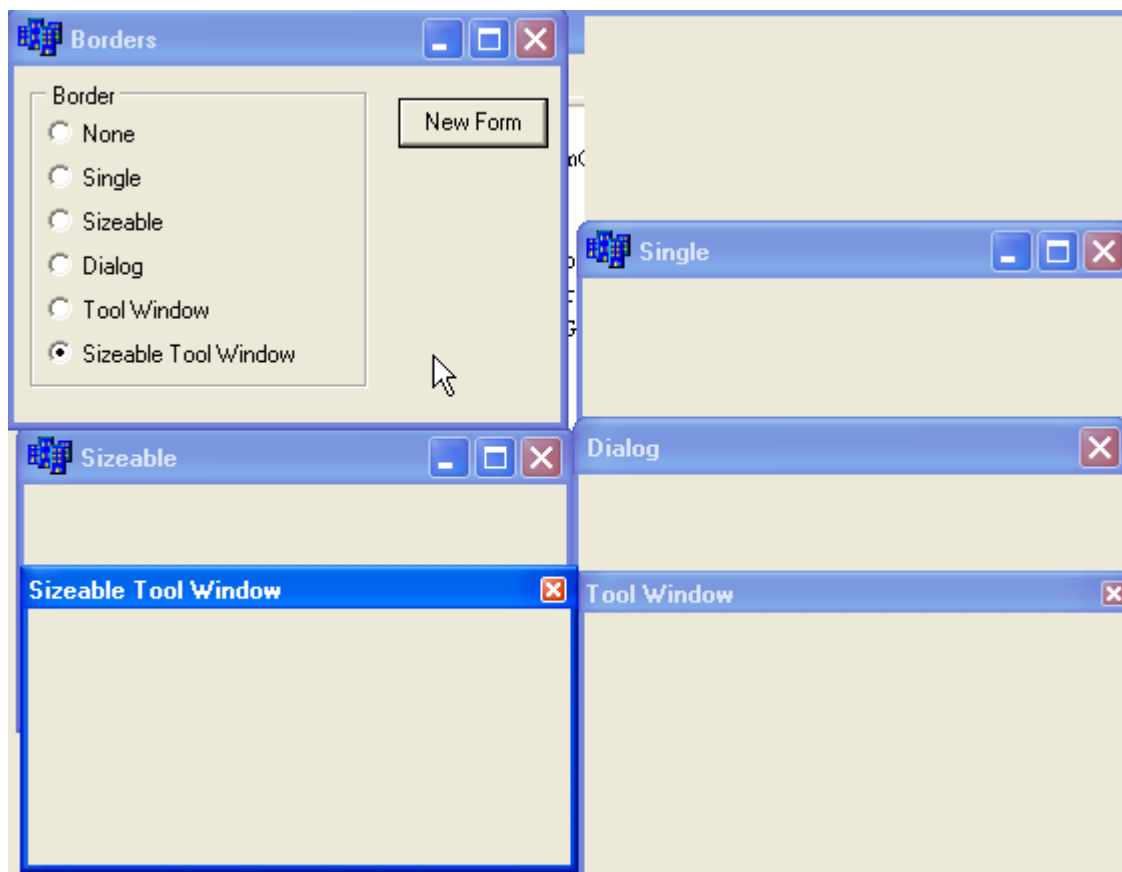
```
this->Close(); //đóng Form chính
//hoặc có thể sử dụng lệnh đóng ứng dụng: Application->Terminate();
}
- Viết mã lệnh cho lệnh Count:
void __fastcall TMainForm::Count1Click(TObject *Sender)
{
    int    NBounce, NCircle, I;
    UnicodeString ClassName;
    NBounce = 0;
    NCircle = 0;
    for( I = 0; I <=MDIChildCount - 1; I++)
    {
        ClassName=AnsiString(MDIChildren [I]->ClassName());
        if(ClassName=="TBounceChildForm" )
            NBounce++;
        else
            NCircle++;
    }
    MessageDlg ("There are " + IntToStr(MDIChildCount) + " child forms." +
    IntToStr(NCicle)+ " are Circle child windows and "+ IntToStr(NBounce) +" are Bouncing child
    windows", mtInformation, TMsgDlgButtons() << mbOK, 0);
}
- Các chức năng khác trong menu window được gắn kết với hành động tương ứng.
- Chọn lệnh Project/Options... và điều chỉnh ở ngăn Form để có cửa sổ như sau:
```



Hình 76- Điều tác các Form

Nhấn nút Ok để lưu lại các thay đổi. Việc làm này khai báo cho hệ thống rằng MainForm sẽ được tạo ra tự động còn hai Form CircleChildForm và BounceChildForm sẽ được tạo ra bằng mã lệnh.

Một thuộc tính quan trọng khác của một form đó là thuộc tính BorderStyle. Thuộc tính này định dạng kiểu hiển thị của Form. Chúng ta sẽ hiểu các mà thuộc tính này tác động lên Form như trong ví dụ sau:



Hình 77- Kết quả chương trình FormStyle

Chương trình trên khi chúng ta nhấn nút New Form sẽ áp dụng kiểu Form sẽ được tạo ra đã chọn ở nhóm ô chọn.

Chúng ta sẽ tạo chương trình với các tên sau:

Bước 1: Điều chỉnh các thuộc tính cho Form chính theo bảng và hình minh họa sau:



Hình 78- Cửa sổ chính

Các đối tượng được liệt kê như sau:

Đối tượng	Thuộc tính	Giá trị
TForm	Caption	Borders
RadioGroup	Name	BorderRadioGroup
	Items...	None
		Single Sizeable Dialog

		Tool Window Sizeable Tool Window
	ItemIndex	2
Button	Name	BtnNewForm
	Caption	New Form

Bước 2: Tạo Form hai với tên Form2 và để nguyên Form trống.

Bước 3: Lưu lại tập tin của hai Form: Form1 với tên First, Form2 với Second.

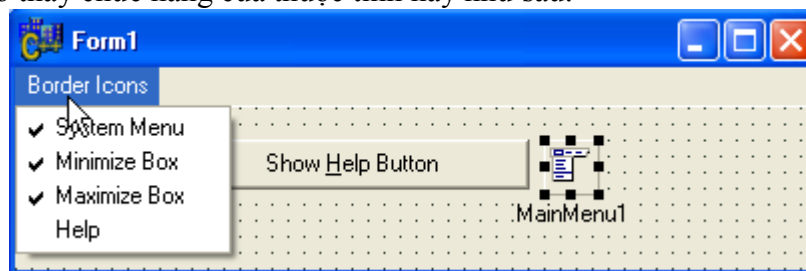
Bước 4: Từ cửa sổ thiết kế của Form1, chúng ta chọn File/Include Unit Hdr... để khai báo thêm thư viện của Form2 (tức là thư viện Second.h).

Bước 5: Viết mã lệnh cho sự kiện OnClick của nút BtnNewForm:

```
void __fastcall TForm1::BtnNewFormClick(TObject *Sender)
{
    TForm2 *NewForm;
    NewForm = new TForm2(Application);
    NewForm->BorderStyle = TFormBorderStyle (BorderRadioGroup->ItemIndex);
    NewForm->Caption      =      BorderRadioGroup->Items->Strings[BorderRadioGroup-
>ItemIndex];
    NewForm->Show();
}
```

Bước 6: Nhấn phím F9 để chạy chương trình.

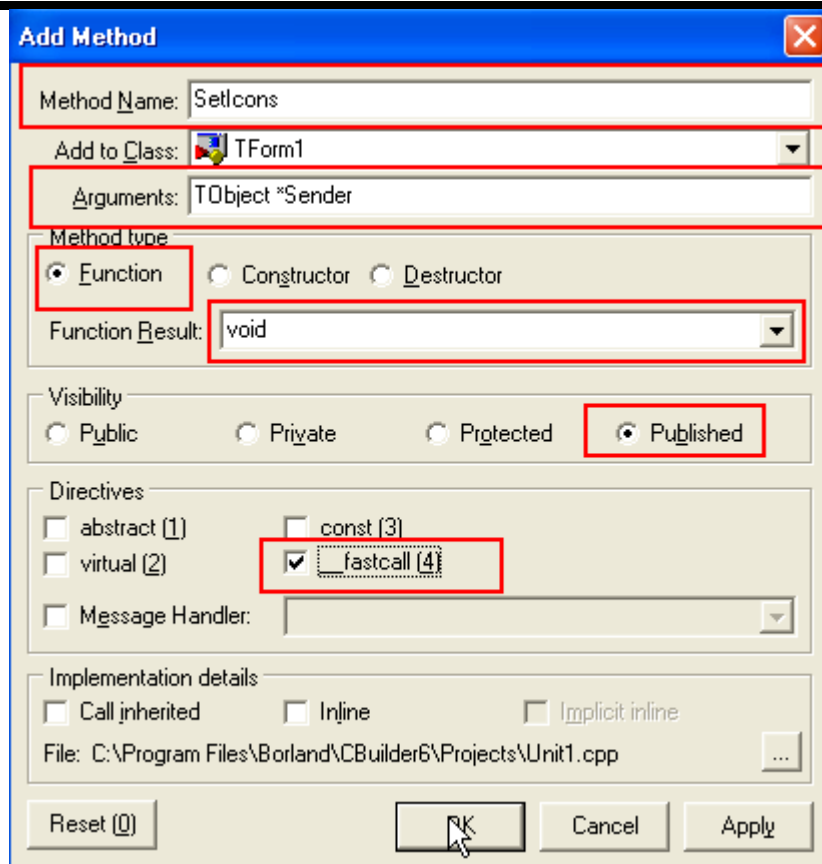
Một thuộc tính quan trọng khác của Form là BorderIcons, thuộc tính này cho phép hiển thị một biểu tượng trên tiêu đề của Form để hiển thị các nút nhấn theo các chức năng của menu hệ thống, chúng ta có thể gán các giá trị biSystemMenu, biMinimize, biMaximize, biHelp. Chương trình ví dụ sau cho thấy chức năng của thuộc tính này như sau:



Hình 79- Thiết kế chương trình BorderIcons

Bước 1: Thiết kế Form giống như hình minh họa.

Bước 2: Thêm phương thức SetIcons như hình minh họa và mã nguồn sau:



Hình 80- Thêm phương thức SetIcons

Mã lệnh:

```
void __fastcall TForm1::SetIcons(TObject * Sender)
{
```

```
    //TODO: Add your source code here
```

```
    AnsiString ClassName;
```

```
    TMenuItem *item;
```

```
    TBorderIcons BorIco= TBorderIcons();
```

```
    item = (TMenuItem *)Sender;
```

```
    item->Checked = ! item->Checked;
```

```
    if(SystemMenu1->Checked )
```

```
        BorIco = TBorderIcons()<< biSystemMenu ;
```

```
    if(MaximizeBox1->Checked)
```

```
        BorIco = BorIco<< biMaximize;
```

```
    if(MinimizeBox1->Checked )
```

```
        BorIco = BorIco<< biMinimize;
```

```
    if(Help1->Checked )
```

```
        BorIco = BorIco<< biHelp;
```

```
    BorderIcons = BorIco;
```

```
}
```

-Bước 3: Viết mã lệnh cho nút nhấn:

```
void __fastcall TForm1::btnHelpClick(TObject *Sender)
```

```
{
```

```
    BorderIcons = TBorderIcons()<< biSystemMenu << biHelp;
```

```
    SystemMenu1->Checked = true;
```

```
    MinimizeBox1->Checked = false;
```

```
    MaximizeBox1->Checked = false;
```

```
    Help1->Checked = true;
```

```
}

```

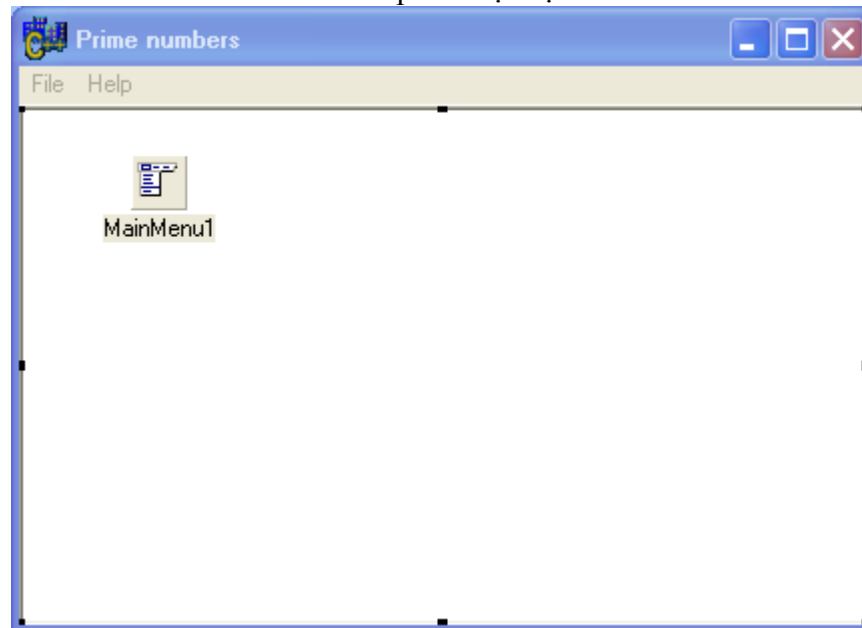
- Bước 4: Gắn kết phương thức SetIcons cho sự kiện OnClick của các phần tử lệnh của Menu.
- Bước 5: Chạy chương trình chúng ta sẽ thấy kiểu dáng Form sẽ bị tác động như thế nào khi chúng ta cho ẩn hiện các nút trên thanh tiêu đề như đóng, thu nhỏ,

Để kết thúc nội dung phần, chúng ta sẽ tạo một chương trình với màn hình Splash.

Đây là một kỹ thuật mà các chương trình thường hay sử dụng để hiển thị một màn hình đầu tiên trước khi form chính được hiển thị. Chương trình sau sử dụng một Form chính kiểm tra và in các giá trị là số nguyên tố từ 2 đến 100000, đoạn mã lệnh này xử lý rất lâu nhưng chúng ta sẽ tạo một Form dạng About nhằm hiển thị cửa sổ này như cửa sổ cài đặt trước khi thêm hết tất cả các số nguyên tố.

Chúng ta tiến hành theo từng bước sau:

Bước 1: Thiết kế Form chính với các thành phần mặc định như sau:



Hình 81-Thiết kế chương trình Splash

Bảng giá trị thuộc tính được liệt kê như sau:

Đối tượng	Thuộc tính	Giá trị
Form	Caption	Prime Numbers
ListBox	Columns	5
	Items	""
	anchors	akLeft=true akRight=true akTop=true akBottom=true

Menu chính của Form chỉ có hai hai mục chọn: File/Exit và Help/About.

Bước 2: Thêm phương thức xử lý kiểm tra giá trị số nguyên có phải là số nguyên tố hay không. Phương thức này trả về giá trị true hoặc false tương ứng với hàm ý là số nguyên tố và không là số nguyên tố.

```
bool TForm1::isPrime(long n)
{
    //TODO: Add your source code here
    int i;
    int k;
    bool kt=true;
```

```

k = Ceil(sqrt(n));
for (i=2;i<=k;i++)
    if (n % i ==0)
    {
        kt=false;
        break;
    }
return kt;
}

```

Để sử dụng được câu lệnh Ceil (làm tròn) và hàm sqrt (căn bậc 2) chúng ta phải khai báo thư viện math.hpp và math.h bằng cách nhấn Ctrl+Home, thêm câu lệnh sau:

```

#include <math.hpp>
#include <math.h>

```

Bước 3: Chọn File/New/Other..., chọn AboutBox trong ngăn Forms. Thêm 1 thanh ProgressBar và Image với hình minh họa sau:



Hình 82- Thiết kế cửa sổ dùng làm Splash

Trong hình minh họa trên, Image1 có thuộc tính Align là alClient (điền đầy Form) và Bốn thuộc tính của ProgressBar1 có giá trị min = 2 , Max = 100000, Visible = false, Align = alBottom (điền góc dưới của Form).

Bước 3: Thêm phương thức MakeSplash (với dạng public) của AboutBox

```

void TAboutBox::MakeSplash()
{
    //TODO: Add your source code here
    this->BorderStyle = bsNone;
    this->OKButton->Visible = false;
    this->ProgressBar1->Visible = true;
    Show();
    Update();
}

```

Bước 4: Chuyển sang Form1 và khai báo thư viện của C++ Builder bằng cách chọn Include Unit Hdr...

Bước 5: Viết mã lệnh cho sự kiện OnCreate của Form1 như sau:

```

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    int i;
    TAboutBox *splash;
    splash=new TAboutBox(Application);
    splash->MakeSplash();
    for( i = 2; i <=100000; i++)

```

```
{
    splash->ProgressBar1->Position = i;
    if(isPrime(i) )
        ListBox1->Items->Add (IntToStr (i));
}
delete splash;
}
```

Bước 6: Viết mã lệnh cho lệnh File/Exit:

```
void __fastcall TForm1::Exit1Click(TObject *Sender)
{
    this->Close();
}
```

Bước 7: Viết mã lệnh cho lệnh File/About:

```
AboutBox->ShowModal();
```

Bước 8: Nhấn phím F9 để chạy chương trình.

Như vậy, chúng ta đã hiểu phương pháp tạo và quản lý các Form trong chương trình. Ngoài các thuộc tính được đề cập ở trên, chúng ta còn một số thuộc tính chưa được đề cập vì ít khi sử dụng chẳng hạn thuộc tính Alpha là giá trị cho phép chúng ta tạo độ đậm cho Form (nếu AlphaBlendValue=0 và AlphaBlend = true, form sẽ trong suốt).

5. Xây dựng chương trình TextEditor

Trong nội dung bài này, chúng ta sẽ gắng xây dựng một chương trình soạn thảo văn bản đơn giản. Khi xây dựng chương trình này, chúng ta sẽ gặp một số đối tượng chưa được nghiên cứu. Tuy nhiên, khó khăn này sẽ không phải là vấn đề lớn, chúng ta sẽ tiến hành ứng dụng theo từng bước.

5.1. Tạo dự án

Chúng ta tiến hành tạo một dự án để bắt đầu thực hiện chương trình của chúng ta theo các bước sau:

Bước 1: Tạo một thư mục có tên TextEditor tại vị trí lưu trữ thích hợp.

Bước 2: Tạo một dự án mới bằng cách chọn thực đơn File/New/Application.

Bước 3: Chọn File/Save All để lưu các tập tin vào đĩa. Khi hộp thoại Save As hiển thị ra, chúng ta tìm đến thư mục TextEditor đã tạo ở bước 1. Tiến hành:

- Lưu tập tin Unit1 với tên mặc định Unit1.cpp.

- Lưu tập tin dự án với tên TextEditor.

Bước 4: Đặt thuộc tính Caption cho biểu mẫu thành Text Editor.

5.2. Thêm đối tượng soạn thảo

Ở nội dung này, chúng ta tiến hành thêm các đối tượng vào biểu mẫu để tạo giao diện chính cho chương trình:

Bước 1: Để thêm vùng soạn thảo văn bản, trước tiên, chúng ta thêm đối tượng RichEdit (ngăn công cụ Win32). Điều chỉnh thuộc tính Lines để nó trở thành giá trị rỗng.

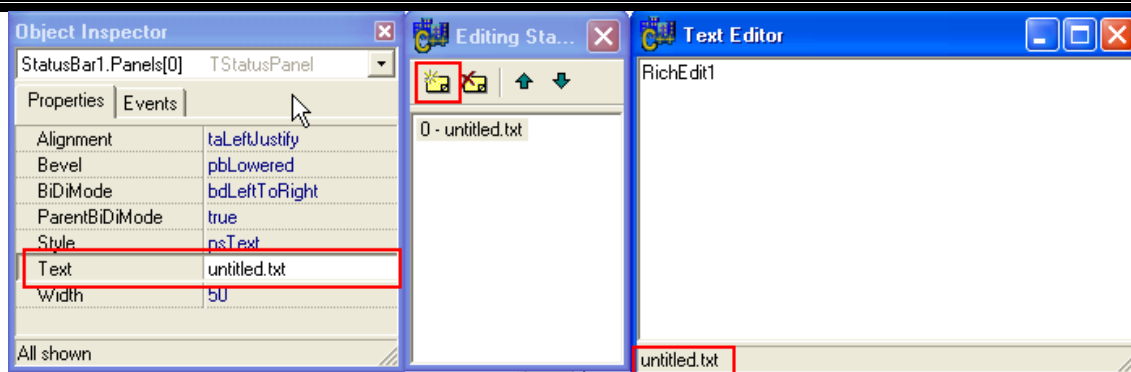
Bước 2: Điều chỉnh thuộc tính Align của đối tượng RichEdit ở trên thành alClient. Khi điều chỉnh thuộc tính này, đối tượng RichEdit sẽ lấp đầy trên biểu mẫu, kích thước của đối tượng RichEdit cũng sẽ thay đổi mỗi khi biểu mẫu thay đổi kích thước.

Bước 3: Thêm đối tượng StatusBar (thanh trạng thái - ngăn Win32), đối tượng này sẽ tự động được thêm ở cạnh dưới của biểu mẫu (Thuộc tính Align = alBottom).

Bước 4: Nhấn đối chuột lên thanh trạng thái để mở hộp thoại thiết kế thanh trạng thái. (Nhấn trên nút ... cạnh thuộc tính Panels).

Bước 5: Nhấn nút Add để thêm một phần tử cho đối tượng thanh trạng thái. Phần tử này dùng để hiển thị tên tập tin.

Bước 6: Điều chỉnh thuộc tính Text của phần tử vừa thêm vào thanh trạng thái thành untitled.txt.



Hình 83-Soạn thảo phần tử trên thanh trạng thái

Bước 7: Đóng hộp thoại soạn thảo thanh trạng thái.

5.3. Hoạch định chức năng

Một ứng dụng làm gì đi nữa thì thường nó cần một thực đơn và thanh công cụ. Chúng ta sử dụng đối tượng ActionList để xây dựng các đoạn mã lệnh dùng chung cho các chức năng nằm trên thanh thực đơn và thanh công cụ.

Bảng sau liệt kê các lệnh có được hoạch định:

Thực đơn	Lệnh	Nằm trên thanh công cụ	Ý nghĩa
File	New	Có	Tạo một tập tin mới
File	Open	Có	Mở một tập tin trên đĩa
File	Save	Có	Lưu tập tin đang soạn thảo vào đĩa
File	Save as...	Không	Lưu tập tin với tên khác
File	Exit	Có	Thoát khỏi chương trình
Edit	Cut	Có	Cắt văn bản vào vùng clipboard
Edit	Copy	Có	Sao chép văn bản vào vùng clipboard
Edit	Paste	Có	Dán văn bản từ clipboard
Help	Contents	Không	Hiển thị nội dung hướng dẫn chính
Help	Index	Không	Hiển thị ngăn Index của phần hướng dẫn
Help	About	Không	Hiển thị thông tin bản quyền

5.4. Thiết kế danh sách hành động

Trong nội dung này chúng ta sẽ dùng đối tượng ActionList để chia sẻ các đoạn mã lệnh dùng chung cho lệnh nằm trên thực đơn và trên thanh công cụ. Trước tiên chúng ta chuẩn bị hình ảnh để gắn kết với thực đơn và thanh công cụ theo các bước sau:

Bước 1: Thêm đối tượng ImageList (ngăn Win32) vào biểu mẫu.

Bước 2: Nhấn đôi chuột lên đối tượng ImageList để mở hộp thoại soạn thảo hình ảnh.

Bước 3: Nhấn nút Add.

Bước 4: Hộp thoại chọn hình ảnh hiển thị lên màn hình, chúng ta tìm thư mục chứa hình ảnh, mặc định chứa trong thư mục C:\Program Files\Common Files\Borland Shared\Images\Buttons.

Bước 5: Nhấn đôi chuột lên tập tin filenew.bmp.

Khi cửa sổ hiển thị ra hỏi rằng bạn có muốn chia thành 2 tập tin hay không? Chọn Yes. Khi đó, chúng ta sẽ nhìn thấy hai hình ảnh được thêm vào. Chúng ta xóa hình ảnh xám đi (Hình thứ 2).

Bước 6: Tương tự bước 5, thêm các hình ảnh sau:

fileopen.bmp; filesave.bmp; doorshut.bmp; cut.bmp; copy.bmp; paste.bmp

Như vậy khi kết thúc bước này, các hình ảnh sẽ có ImageIndex liệt kê như sau:

Tên tập tin	ImageIndex
Filenew.bmp	0
Fileopen.bmp	1
Filesave.bmp	2
Doorshut.bmp	3
Cut.bmp	4
Copy.bmp	5
Paste.bmp	6
Help.bmp	7

Chúng ta tiến hành gắn kết các hình ảnh lên các hành động và thêm các hành động vào ActionList theo các bước sau:

Bước 7: Nhấn đôi chuột lên đối tượng ActionList trên ngăn Standard. Khi đó đối tượng được tạo ra mang tên ActionList1.

Bước 8: Cài đặt thuộc tính Images của đối tượng ActionList1 thành ImageList1.

Bước 9: Nhấn đôi chuột lên đối tượng ActionList1, chúng ta mở được cửa sổ soạn thảo Action List.

Bước 10: Nhấn chuột lên nút New Action (hoặc phím Insert). Đối tượng mới tạo ra mang tên Action1.

Bước 11: Điều chỉnh thuộc tính cho đối tượng Action1 theo bảng sau:

Thuộc tính	Giá trị	Ghi chú
Caption	&New	Ký tự N là phím nóng
Category	File	Lệnh này thuộc nhóm lệnh File
Hint	Create file	Dòng văn bản chú thích ngắn gọn về chức năng
ImageIndex	0	
Name	FileNew	

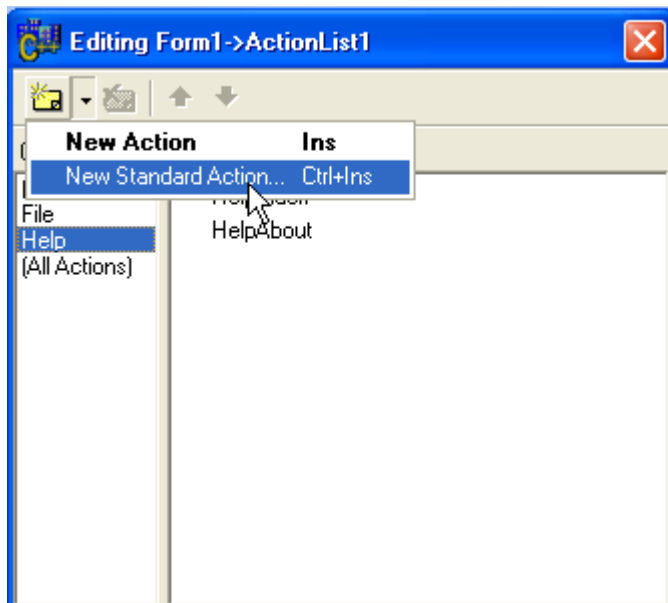
Bước 12: Chúng ta tiến hành các bước tương tự như bước 11 để có được bảng đối tượng liệt kê như sau:

Đối tượng	Thuộc tính	Giá trị
Action	Caption	&Save
	Category	File
	Hint	Save file
	ImageIndex	2
	Name	FileSave
Action	Caption	&Index
	Category	Help
	ImageIndex	-1
	Name	HelpIndex
Action	Caption	&About
	Category	Help
	ImageIndex	7

	Name	HelpAbout
--	------	-----------

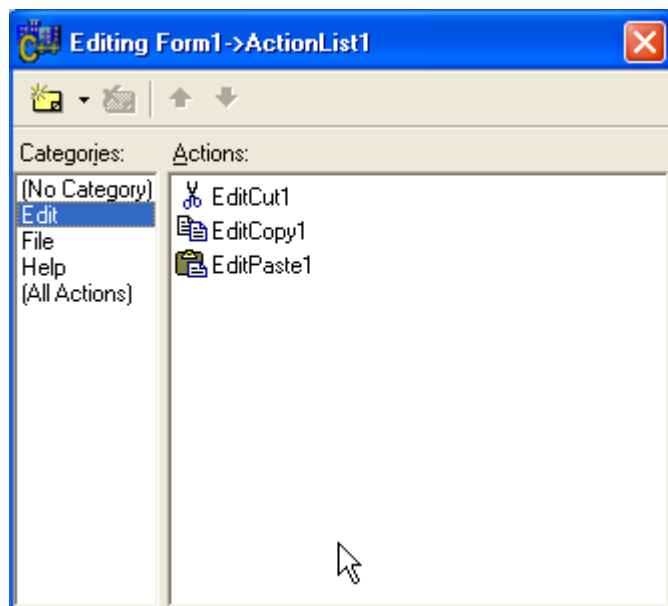
C++ Builder có dựng sẵn cho chúng ta các hành động chuẩn như Copy, Cut, ... Chúng ta sẽ vận dụng các hành động chuẩn này để tạo các lệnh còn lại theo các bước sau:

Bước 13: Chọn New Standard Action... như hình sau:



Hình 84-Tạo hành động chuẩn

Bước 14: Trong hộp thoại Standard Action Classes, dùng phím Ctrl và chọn các phần tử TEditCut, TEditCopy, and TEditPaste của chức năng Edit. Nhấn Ok. Cửa sổ soạn thảo ActionList sẽ như sau:



Hình 85-Soạn thảo ActionList

Bước 15: Thực hiện tương tự bước 14 để tạo các hành động của chức năng File: Scroll TFileOpen, TFileSaveAs, và TFileExit. THelpContents của chức năng Help.

Bước 16: Các hành động được gán hình ảnh mặc định, chúng ta có thể giữ nguyên hoặc có thể thay đổi các hình ảnh cho đúng với các hình ảnh ta đã lưu trữ trong ImageList1.

Tên đối tượng	Thuộc tính ImageIndex
EditCut1	4
EditCopy1	5
EditPasted1	5

FileOpen1	1
FileExit	3

Bước 17: Đóng cửa sổ soạn thảo ActionList và chọn File/Save All để lưu tất cả những gì đã sửa đổi.

2.3.5. Tạo thực đơn cho chương trình

Trong phần này, chúng ta sẽ thêm thực đơn chính với ba thực đơn đổ xuống gồm: File, Edit, Help với mỗi mục chọn con trong mỗi mục chọn File, Edit, Help được liên kết với các hành động đã tạo ở trên.

Các bước thực hiện thực đơn chính như sau:

Bước 1: Thêm đối tượng MainMenu trong ngăn Standard của bảng công cụ. (Đối tượng sẽ mang tên MainMenu1)

Bước 2: Cài đặt giá trị cho Images thành ImageList1 để thêm các hình ảnh cho MainMenu.

Bước 3: Nhấp đôi chuột lên đối tượng MainMenu1 để mở cửa sổ Menu Designer (cửa sổ thiết kế thực đơn).

Bước 4: Phần tử đầu tiên sẽ được đặt thuộc tính Caption thành &File và nhấn Enter.

Bước 5: Chọn phần tử trống phía dưới lệnh File vừa tạo. Cài đặt thuộc tính Action thành FileNew

Bước 6: Tương tự như thế chúng ta tiến hành gắn kết các đối tượng. Theo thứ tự được liệt kê dưới đây:

- Chọn phần tử dưới phần tử New và cài đặt thuộc tính Action thành FileOpen1.
- Chọn phần tử dưới phần tử Open và cài đặt thuộc tính Action thành FileSave.
- Chọn phần tử dưới phần tử Save và cài đặt thuộc tính Action thành FileSaveAs1.
- Chọn phần tử dưới phần tử Save As, điều chỉnh thuộc tính Caption thành "-" (không có dấu nháy) để tạo dòng phân cách..
- Chọn phần tử dưới dòng phân cách và cài đặt thuộc tính Action thành FileExit1.

Bước 7: Tạo thực đơn Edit:

- Chọn phần tử bên phải của lệnh File, cài đặt thuộc tính Action thành &Edit, và nhấn phím Enter.
- Chọn phần tử dưới phần tử Edit và cài đặt thuộc tính Action thành EditCut1.
- Chọn phần tử dưới phần tử Cut và cài đặt thuộc tính Action thành EditCopy1.
- Chọn phần tử dưới phần tử Copy và cài đặt thuộc tính Action thành EditPaste1.

Bước 8: Tạo thực đơn Help

- Chọn phần tử bên phải của lệnh Edit command, cài đặt thuộc tính Action thành &Edit, và nhấn phím Enter.
- Chọn phần tử dưới phần tử Help and và cài đặt thuộc tính Action thành HelpContents.
- Chọn phần tử dưới phần tử Contents và cài đặt thuộc tính Action thành HelpIndex.
- Chọn phần tử dưới phần tử Index, và cài đặt thuộc tính Caption thành "-" (không có dấu nháy) để tạo dòng ngăn cách.
- Chọn phần tử dưới dòng ngăn cách và cài đặt thuộc tính Action thành HelpAbout.

Bước 9: Đóng cửa sổ thiết kế thực đơn và chọn File/Save All để lưu lại những thay đổi.

5.6. Thêm thanh công cụ

Chúng ta tiến hành thêm thanh công cụ bằng các bước sau:

Bước 1: Thêm đối tượng Toolbar trong ngăn Win32 vào biểu mẫu.

Bước 2: Cài đặt giá trị cho thuộc tính Images thành giá trị ImageList1. Cài đặt thuộc tính Indent thành 4 (Giá trị này canh lề trái cho hình ảnh cách lề trái 4 điểm ảnh). Cài đặt giá trị ShowHint thành true.

Bước 3: Để thêm nút nhấn vào thanh công cụ, chúng ta nhấn chuột phải và chọn New Button. Chúng ta thêm bốn nút nhấn vào công cụ.

Bước 4: Nhấn chuột phải lên thanh công cụ, chọn New Separator để thêm một dòng ngăn cách.

Bước 5: Thêm ba nút nhấn nữa.

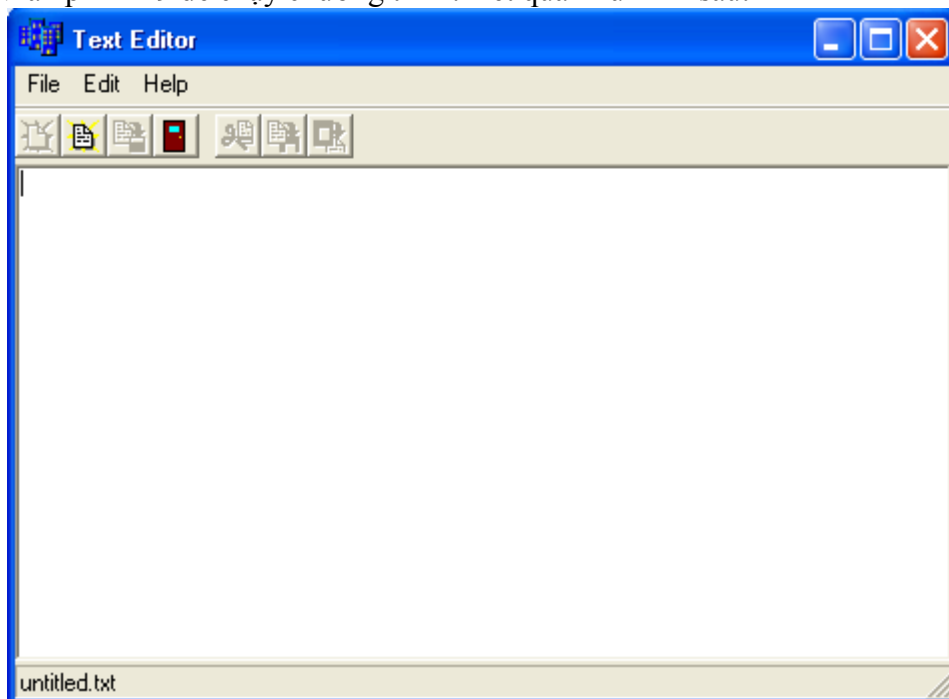
Bước 6: Gắn kết nối các hành động với các nút nhấn:

Điều chỉnh thuộc tính Action thành các giá trị như sau:

Nút đầu tiên thành FileNew; nút thứ 2 thành FileOpen1; nút thứ 3 thành FileSave; nút thứ 4 thành FileExit. nút nhấn thứ 5 thành EditCut1; nút nhấn thứ 6 thành EditCopy1; nút nhấn cuối cùng thành EditPaste1.

Bước 7: Chọn File/Save All.

Bước 8: Nhấn phím F9 để chạy chương trình. Kết quả như hình sau:



Hình 86-Cửa sổ chương trình

5.7. Hoàn chỉnh sự kiện

Trong phần này, chúng ta sẽ tiến hành hoàn chỉnh mã lệnh cho các hành động không chuẩn và bổ sung các giá trị để hoàn chỉnh các hành động chuẩn.

a) Mã lệnh cho hành động FileNew

Bước 1: Khai báo thêm một biến mang tên FileName có phạm vi toàn cục bằng cách chọn View/Units..., chọn Unit1, nhấn nút OK. Nhấn chuột phải lên ngăn Unit1.cpp, chọn Open Source/Header file. Tìm vị trí public và chèn thêm đoạn lệnh để được như sau:

```
public:           // User declarations
    __fastcall TForm1(TComponent* Owner);
    AnsiString FileName;
```

Bước 2: Nhấn phím F12 để về cửa sổ thiết kế biểu mẫu.

Bước 3: Nhấn đôi chuột lên đối tượng ActionList1, nhấn đôi chuột lên hành động FileNew. Chèn đoạn mã lệnh để đạt được đoạn mã lệnh sau:

```
void __fastcall TForm1::FileNewExecute(TObject *Sender)
{
    RichEdit1->Clear();
    FileName = "untitled.txt"; //khởi tạo tên tập tin
    StatusBar1->Panels->Items[0]->Text = FileName;
}
```

Bước 4: Chọn File/Save All.

b) Viết mã lệnh cho hành động FileOpen1

Bước 1: Nhấn phím F12, nhấn đôi chuột lên đối tượng ActionList1.

Bước 2: Chọn hành động FileOpen1.

Bước 3: Trong cửa sổ thuộc tính, nhấn chuột vào dấu cộng bên trái thuộc tính Dialog để mở các thuộc tính của Dialog. Dialog tham chiếu đến hộp thoại mở tập tin với tên mặc định là FileOpen1->OpenDialog. Khi phương thức Execute được thực thi nó sẽ mở một hộp thoại để cho phép chúng ta chọn tập tin để mở.

Bước 4: Cài đặt thuộc tính DefaultExt thành txt.

Bước 5: Nhấn đôi chuột lên thuộc tính Filter để hiển thị cửa sổ soạn thảo Filter (Tạo bộ lọc tập tin theo phần mở rộng).

Trong dòng đầu tiên, chúng ta gõ Text Files (*.txt) vào cột Filter Name và gõ .txt vào cột Filter.

Ở dòng thứ hai, chúng ta gõ All files (*.*) vào cột Filter Name và giá trị *.* vào cột Filter. Nhấn nút Ok.

Bước 6: Cài đặt giá trị thuộc tính Title thành Open file. Như vậy hộp thoại này sẽ có tiêu đề là Open file.

Bước 7: Chọn ngăn Events, chúng ta nhấn đôi chuột lên sự kiện OnAccept để viết mã lệnh cho sự kiện này. Chèn đoạn mã lệnh để đạt được kết quả như sau:

```
void __fastcall TForm1::FileOpen1Accept(TObject *Sender)
{
    RichEdit1->Lines->LoadFromFile (FileOpen1->Dialog->FileName);
    FileName = FileOpen1->Dialog->FileName;
    StatusBar1->Panels->Items[0]->Text = FileName;
}
```

Bước 8: Chọn File/Save All

c) Viết mã lệnh cho hành động FileSave

Bước 1: Nhấn F12, nhấn đôi chuột trái lên đối tượng ActionList1.

Bước 2: Nhấn đôi chuột lên hành động FileSave và chèn đoạn mã lệnh để đạt được đoạn mã lệnh như sau:

```
void __fastcall TForm1::FileSaveExecute(TObject *Sender)
{
    if (FileName == "untitled.txt")
        FileSaveAs1->Execute(); //lưu lần đầu gọi FileSaveAs1
    else
        RichEdit1->Lines->SaveToFile(FileName);
}
```

Bước 3: Chọn File/Save All

d) Viết mã lệnh cho hành động FileSaveAs1

Bước 1: Nhấn F12 và nhấn đôi chuột lên đối tượng ActionList1.

Bước 2: Chọn hành động FileSaveAs1.

Bước 3: Trong cửa sổ thuộc tính mở rộng thuộc tính Dialog để điều chỉnh các thuộc tính cho hộp thoại lưu tập tin. .

Bước 4: Cài đặt giá trị của thuộc tính DefaultExt thành txt.

Bước 5: Thực hiện theo bước 5 của phần b ở trên.

Bước 6: Cài đặt thuộc tính Title thành Save as

Bước 7: Viết mã lệnh cho sự kiện BeforeExecute như sau:

```
void __fastcall TForm1::FileSaveAs1BeforeExecute(TObject *Sender)
{
```

```
    FileSaveAs1->Dialog->InitialDir = ExtractFilePath (FileName);
}
```

Bước 8: Viết mã lệnh cho sự kiện OnAccept như sau:

```
void __fastcall TForm1::FileSaveAs1Accept(TObject *Sender)
```

{

```

FileName = FileSaveAs1->Dialog->FileName;
RichEdit1->Lines->SaveToFile(FileName);
StatusBar1->Panels->Items[0]->Text = FileName;
}

```

Bước 9: Chọn File/Save all

d) Tạo tập tin hướng dẫn

Chúng ta có thể sử dụng chương trình hcw.exe trong thư mục C:\Program Files\Borland\CBuilder6\Help\Tool để tạo tập tin hướng dẫn. Trong mục này, chúng tôi sử dụng ba tập tin hướng dẫn mẫu TextEditor.rtf (rich text file)s, tập tin hướng dẫn (TextEditor.hlp) và tập tin nội dung hướng dẫn (TextEditor.cnt) được giải nén từ tập tin C:\Program Files\Borland\CBuilder6\Help\B6X1.zip vào thư mục TextEditor.

Chúng ta mở cửa sổ thiết kế biểu mẫu, nhấn đôi đối tượng ActionList1, nhấn đôi hành động HelpContents1 và chèn đoạn mã lệnh để được đoạn mã lệnh sau:

```

void __fastcall TForm1::HelpContents1Execute(TObject *Sender)
{
    const static int HELP_TAB = 15;
    const static int CONTENTS_ACTIVE = -3;
    Application->HelpCommand(HELP_TAB, CONTENTS_ACTIVE);
}

```

Đoạn mã lệnh trên dùng để mở cửa sổ Help và kích hoạt ngăn contents.

Tương tự, chúng ta viết mã lệnh cho hành động HelpIndex như sau:

```

void __fastcall TForm1::HelpIndexExecute(TObject *Sender)
{
    const static int HELP_TAB = 15;
    const static int INDEX_ACTIVE = -2;
    Application->HelpCommand(HELP_TAB, INDEX_ACTIVE);
}

```

e) Viết mã lệnh cho hành động HelpAbout

Chúng ta theo các bước sau:

Bước 1: Thêm hộp thoại AboutBox bằng cách chọn File/New/Other/Forms.

Bước 2: Nhấn đôi chuột lên biểu tượng About Box.

Bước 3: Đổi các thuộc tính Caption như sau:

Product Name thành Text Editor.

Version thành Version 1.0.

Copyright thành Copyright by Cao Dang Nghe Da Lat.

Comment thành This program written by Le Xuan Thach

Bước 4: Lưu biểu mẫu trên bằng cách chọn File|Save As và lưu với tên About.cpp.

Trong cửa sổ soạn thảo mã lệnh chúng ta thấy có những ngăn sau: Unit1.cpp, Unit1.h, About.cpp

Bước 5: Nhấn chuột lên ngăn Unit1.cpp, nhấn tổ hợp phím Ctrl + Home. Chọn File|Include Unit Hdr, sau đó chọn About và nhấn OK. Chú ý rằng có câu lệnh #include được thêm vào đầu tập tin Unit1.cpp .

Bước 6: Nhấn F12, nhấn đôi chuột lên đối tượng ActionList1.

Bước 7: Nhấn đôi chuột lên hành động HelpAbout và viết đoạn mã lệnh để có đoạn mã lệnh sau:

```

void __fastcall TForm1::HelpAboutExecute(TObject *Sender)
{
    AboutBox->ShowModal();
}

```

Sự kiện ShowModal() sẽ mở cửa sổ ở chế độ Modal, chế độ này yêu cầu người sử dụng phải đóng biểu mẫu này lại nếu muốn tiếp tục thực hiện một tác vụ khác.

Bước 8: Chọn File/Save All

g) Khởi tạo các giá trị đầu tiên

Viết đoạn mã lệnh cho sự kiện OnCreate của biểu mẫu như sau:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Application->HelpFile = ExtractFilePath(Application->ExeName) + "TextEditor.hlp";
    FileName = "untitled.txt";
    StatusBar1->Panels->Items[0]->Text = FileName;
}
```

Trong đoạn mã lệnh này, chúng ta tiến hành gán tên tập tin hướng dẫn là TextEditor.hlp cho chương trình và khởi tạo tên tập tin cũng như thanh trạng thái.

Chúng ta đã xây dựng thành công chương trình trên.

BÀI 5- ĐỒ HỌA VÀ MULTIMEDIA

1. Tổng quan về lập trình đồ họa

Trong những chương trên, chúng ta đã từng đặt các thành phần VCL như nút nhấn, nhãn, ... lên cửa sổ Form để tạo ra ứng dụng. Khi chúng ta đặt một nhãn hay một nút nhấn lên Form, hệ điều hành vẽ lên cửa sổ của Form các đối tượng này.

Theo cách lập trình của Windows, chúng ta sử dụng hệ thống Windows API để vẽ thông qua DC của hệ điều hành. Ví dụ, để vẽ một hình chữ nhật, chúng ta có thể sử dụng các lệnh Windows API bằng đoạn mã lệnh tương tự như sau:

```
voidfastcall TForm1::ButtonDrawGDIClick(TObject *Sender)
{
    HDC hDC = GetDC(Handle);
    Rectangle(hDC,10,10,100,100);
    ReleaseDC(Handle, hDC);
}
```

Trong ví dụ này, chúng ta sử dụng hàm GetDC để lấy DC của cửa sổ, sau đó vẽ hình chữ nhật lên form sau đó giải phóng DC bằng cách ReleaseDC.

Trong nội dung của bài học này, chúng ta không nghiên cứu phương pháp lập trình trong Windows API. Để có thể nghiên cứu kỹ hơn về nội dung lập trình Windows API, chúng ta có thể tìm đọc sách Lập trình Windows (Programming Windows) để nghiên cứu kỹ hơn về nội dung lập trình này. Do đó, trong nội dung này chúng ta chỉ nghiên cứu phương pháp lập trình do chính C++ Builder cung cấp, đó là sử dụng đối tượng Canvas sẽ được đề cập trong mục 7.3.

2. Các thành phần đồ họa VCL

VCL cung cấp các đối tượng đồ họa có phương thức để vẽ trên canvas, bằng cách sử dụng Canvas vẽ các đối tượng đồ họa và có thể tải và lưu các tập tin đồ họa.

Đối tượng	Diễn giải
Picture	Được sử dụng để giữ bất kỳ hình ảnh đồ họa nào. Để thêm một định dạng tập tin mới, chúng ta sử dụng phương thức Picture Register. Sử dụng phương thức này để xử lý độc quyền các tập tin như hiển thị hình ảnh trong một đối tượng hình ảnh.
Bitmap	Một đối tượng đồ họa mạnh mẽ được sử dụng để tạo, quản lý (co giãn, cuộn, quay và vẽ), lưu trữ các hình ảnh như tập tin trên đĩa. Tạo bản sao của một ảnh bitmap một cách nhanh chóng, đó là bản sao, không phải hình ảnh.
Clipboard	Đại diện cho côngtenơ cho bất kỳ văn bản hay đồ họa nào mà bị cắt, được sao chép, hay được dán từ hoặc tới một ứng dụng. Với clipboard, bạn có thể có và khôi phục dữ liệu theo khuôn dạng thích hợp; quản lý tham chiếu đếm, và sự mở và đóng Clipboard; quản lý và thao tác những định dạng cho những đối tượng trong clipboard.
Icon	Án định giá trị tải từ một tập tin tin biểu tượng (.ICO file).
Metafile	Chứa đựng một tập tin bao gồm các phép toán yêu cầu để tạo hình ảnh chứ không phải chứa các điểm ảnh thực của hình ảnh.

3. Sử dụng các thuộc tính và các phương thức của đối tượng Canvas

Đối tượng Canvas không tìm thấy trên bảng công cụ của C++ Builder. Canvas là một thuộc tính con, gắn liền với hầu hết các đối tượng đồ họa. Canvas đóng vai trò như một tấm vải vẽ, chúng ta có thể sử dụng bút vẽ (pen), màu (color), cọ vẽ (brush), điểm ảnh (pixel), chế độ vẽ

(pen mode) cùng các phương thức đơn giản như Line, Ellipse, Circle, ... để vẽ lên những hình, từ đơn giản đến phức tạp.

Đối với Form, Canvas chính là bề mặt của cửa sổ. Đối với ảnh Bitmap, Canvas chính là bề mặt mà các điểm ảnh (pixel) thể hiện trên đó. Ví dụ, để vẽ hình Ellipse lên cửa sổ Form ta gọi lệnh sau:

```
Form1->Canvas->Ellipse(10,10,50,100);
```

Chúng ta cũng có thể sao chép nội dung ảnh vẽ từ một Canvas này sang Canvas khác. Ví dụ dưới đây sẽ tạo ra đối tượng TBitmap, nạp ảnh .bmp từ đĩa vào Canvas của TBitmap, sau đó chép ảnh sang Canvas của Form:

```
void TForm1::Button1Click(TObject *Sender)
{
    TRect r;
    Graphics::TBitmap *bmp;
    bmp = new Graphics::TBitmap();
    bmp->LoadFromFile("C:\\Windows\\Athen.bmp");
    r= Rect(0,0,bmp->Width,bmp->Height);
    this->Canvas->CopyRect(this->ClientRect,bmp->Canvas,r);
    delete bmp;
}
```

Sau đây sẽ liệt kê các thuộc tính và phương thức của đối tượng Canvas.

Bảng thuộc tính thông dụng của đối tượng Canvas.

Thuộc tính	Diễn giải
Font	Chỉ định font được sử dụng khi ghi văn bản trên hình ảnh. Tập hợp các thuộc tính của đối tượng TFont chỉ định bề mặt font, màu, kích cỡ và kiểu dáng của font chữ.
Brush	Chỉ định màu và mẫu của Canvas sử dụng để điền các hình vẽ đồ họa và nền. Tập các thuộc tính của đối tượng TBrush chỉ định màu và mẫu hay hình ảnh sử dụng khi điền trong không gian của Canvas.
Pen	Chỉ định dạng của bút vẽ sử dụng để vẽ đường và đường viền.
PenPos	Chỉ định vị trí của bút vẽ.
Pixels	Chỉ định mã của một vùng của các điểm trong ClipRect hiện tại.

Một số phương thức thông dụng của Canvas.

Phương thức	Diễn giải
Arc	Vẽ một cung tròn.
Chord	Một cung tròn khép kín bằng cách nối điểm đầu và điểm cuối của cung tròn.
CopyRect	Sao chép một phần của bức ảnh từ canvas khác vào canvas hiện tại.
Draw	Vẽ một đối tượng hình ảnh chỉ định bởi các tham số đồ họa.
Ellipse	Vẽ hình ellipse định nghĩa bởi một hình chữ nhật làm biên.
FillRect	Điền một hình chữ nhật chỉ định trên canvas sử dụng cọ vẽ (brush) hiện tại.
FloodFill	Điền một vùng của canvas sử dụng cọ vẽ hiện tại.
FrameRect	Vẽ một hình chữ nhật sử dụng cọ vẽ của canvas để vẽ đường viền.
LineTo	Vẽ một đường thẳng trên canvas từ PenPos đến vị trí X và Y chỉ định; đưa vị trí bút vẽ đến (X,Y).
MoveTo	Thay đổi vị trí vẽ hiện tại đến điểm (X,Y).

Pie	Vẽ một hình bánh là của một hình ellipse được định biên bởi hình chữ nhật (X1,Y1,X2,Y2).
Polygon	Vẽ một loạt các đường thẳng trên canvas kết nối các điểm và đóng bằng cách vẽ đường thẳng nối điểm đầu với điểm cuối.
Polyline	Vẽ một loạt các đường thẳng trên canvas với bút vẽ hiện tại, nối chúng lại với nhau bằng cách chuyển các điểm vào Points.
Rectangle	Vẽ một hình chữ nhật trên canvas với đỉnh trên (X1,Y1) và góc dưới (X2,Y2) sử dụng bút vẽ và điền nó sử dụng cọ vẽ.
RoundRect	Vẽ một hình chữ nhật với góc tròn.
StretchDraw	Vẽ một hình ảnh trên Canvas so cho nó khít với hình chữ nhật chỉ định.
TextHeight, TextWidth	Trả về giá trị chiều cao và chiều rộng của một chuỗi trong font hiện tại.
TextOut	Ghi một chuỗi trên Canvas.
TextRect	Ghi một chuỗi bên trong một vùng.

Để minh họa, chúng ta xem xét một số ví dụ về đối tượng Canvas và các đối tượng liên quan:
void __fastcall TForm1::FormPaint(TObject *Sender)

```
{
  Canvas->MoveTo(0,0);
  Canvas->LineTo(ClientWidth, ClientHeight);
  Canvas->MoveTo(0, ClientHeight);
  Canvas->LineTo(ClientWidth, 0);
}
```

- Đoạn lệnh sau đọc một tập tin hình ảnh và sử dụng nó để làm cọ vẽ:

```
Graphics::TBitmap *BrushBmp = new Graphics::TBitmap;
```

```
try
{
  BrushBmp->LoadFromFile("MyBitmap.bmp");
  Form1->Canvas->Brush->Bitmap = BrushBmp;
  Form1->Canvas->FillRect(Rect(0,0,100,100));
}
__finally
```

```
{
  Form1->Canvas->Brush->Bitmap = NULL;
  delete BrushBmp;
}
```

- Đoạn lệnh sau thay đổi cọ vẽ bằng cách dùng màu đỏ và tô bằng đường chéo ca rô:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{

  TCanvas *pCanvas = Image1->Canvas;

  pCanvas->Brush->Color = clRed;

  pCanvas->Brush->Style = bsDiagCross;
  pCanvas->Ellipse(0, 0, Image1->Width, Image1->Height);
}
```

- Trong ví dụ sau sẽ dùng một Timer để vẽ các hình chữ nhật ngẫu nhiên với các đường vẽ được sinh ra ngẫu nhiên:

```
int x, y;
//-----

void __fastcall TForm1::FormActivate(TObject *Sender)
{
    WindowState = wsMaximized;
    Timer1->Interval = 50;
    randomize();
}
//-----

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    x = random(Screen->Width - 10);
    y = random(Screen->Height - 10);
    Canvas->Pen->Color = (Graphics::TColor) random(65535);
    switch (random(5))
    {
        case 0: Canvas->Pen->Style = psSolid; break;
        case 1: Canvas->Pen->Style = psDash; break;
        case 2: Canvas->Pen->Style = psDot; break;
        case 3: Canvas->Pen->Style = psDashDot; break;
        case 4: Canvas->Pen->Style = psDashDotDot; break;
    }
    Canvas->Rectangle(x, y, x + random(400), y + random(400));
}
- Dùng Canvas để vẽ một dòng văn bản lên bề mặt form:
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
{
    Canvas->Pen->Color = clBlue;
    Canvas->MoveTo( 10, 10 );
    Canvas->LineTo( 100, 100 );
    Canvas->Brush->Color = clBtnFace;
    Canvas->Font->Name = "Arial";
    Canvas->TextOut( Canvas->PenPos.x, Canvas->PenPos.y, "This is the end of the line" );
}
```

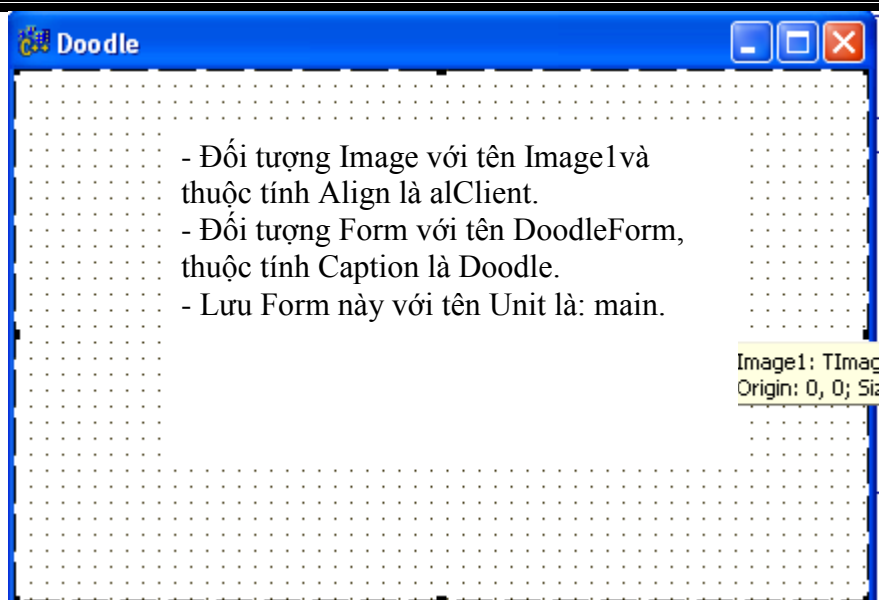
4. Tạo và quản lý các bức vẽ

Trong nội dung phần này, chúng ta sẽ vận dụng các kiến thức đã học để tiến hành tạo các ứng dụng bằng cách sử dụng Canvas và các công cụ hỗ trợ khác.

Chương trình đầu tiên là chương trình Dooble trong bộ ví dụ kèm theo của C++ Builder 6.

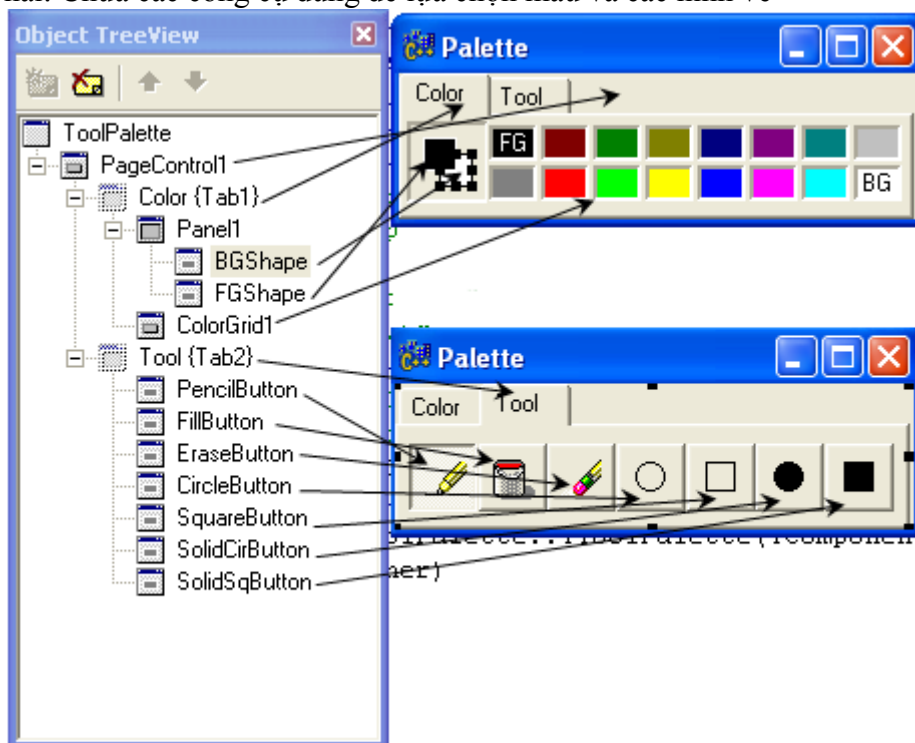
Đầu tiên chúng ta tạo một dự án với giao diện sau:

-Form đầu tiên: Là form sẽ dùng để vẽ.



Hình 87- Cửa sổ dùng làm cửa sổ vẽ

- Form thứ hai: Chứa các công cụ dùng để lựa chọn màu và các hình vẽ



Hình 88- Cửa sổ lựa chọn màu và hình vẽ

Form này có tên TollPalatte và được lưu vào unit Palette.

Các đối tượng trong khung bên trái có thể được liệt kê từ trên xuống dưới như sau:

- Page Control: Trong ngăn công cụ Win32:

+ Thuộc tính Style: tsTabs.

+ Nhấn chuột phải chọn New page, đặt Caption là Color.

+ Nhấn chuột phải chọn New Page, đặt Caption là Tool

- Chọn ngăn Color đặt các đối tượng như hình vẽ và đổi tên sơ đồ mũi tên:

+ Panel1: Đối tượng Panel từ ngăn công cụ Standard.

+ BGShape và FGShape: Đối tượng Shape từ ngăn công cụ Additional và tiến hành điều chỉnh các thuộc tính trong Brush để có màu đen và màu trắng như trên hình.

+ ColorGrid1: Đối tượng CColorGrid trong ngăn công cụ Samples.

- Chọn ngăn Tool để đặt các đối tượng SpeedButton và tiến hành điều chỉnh các hình ảnh (thuộc tính Glyph), thuộc tính Group =1, thuộc tính Down để đạt được kết quả như trên hình trên.

Tiến hành viết mã lệnh để đạt được các tập tin như sau:

Tập tin main.cpp:

```
#include <vcl.h>
#pragma hdrstop

#include "main.h"
#include "palette.h"
//-----
#pragma resource "*.dfm"
#pragma resource "extrares.res"
TDoodleForm *DoodleForm;
//-----
__fastcall TDoodleForm::TDoodleForm(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TDoodleForm::FormCreate(TObject *Sender)
{
    HINSTANCE HInst;
    HInst = reinterpret_cast<HINSTANCE>(HInstance);
    // Load custom cursors for tools from extrares.res
    Screen->Cursors[crFill] = LoadCursor(HInst, "FILL");
    Screen->Cursors[crPlus] = LoadCursor(HInst, "PLUS");
    Screen->Cursors[crDraw] = LoadCursor(HInst, "DRAW");
    Screen->Cursors[crErase] = LoadCursor(HInst, "ERASE");
    Cursor = TCursor(crDraw);
}
//-----
void __fastcall TDoodleForm::Image1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    // The Doodle form contains an Image control on which all of the painting
    // will occur. Another option would have been to draw directly on the
    // form's canvas. The benefit of drawing on the Image control is that
    // it knows how to repaint itself.

    if (ToolPalette->FillButton->Down)
    {
        // The fill tool is the only one that makes use of the right mouse
        // button. The right mouse button is used to fill with the BG color.
        if (Button == mbLeft)
            Image1->Canvas->Brush->Color = ToolPalette->FGShape->Brush->Color;
        else
            Image1->Canvas->Brush->Color = ToolPalette->BGShape->Brush->Color;
        Image1->Canvas->FloodFill(X, Y, Image1->Canvas->Pixels[X][Y], fsSurface);
        return;
    }
}
```

```
// Otherwise, only the left mouse button is of interest.
if (Button != mbLeft)
    return;

if (ToolPalette->EraseButton->Down)
{
    Image1->Canvas->Pen->Color = ToolPalette->BGShape->Brush->Color;
    Image1->Canvas->Brush->Color = ToolPalette->BGShape->Brush->Color;
    Image1->Canvas->Pen->Width = 13;
    Image1->Canvas->Rectangle(X-1, Y-1, X, Y);
    Image1->Canvas->MoveTo(X,Y);
    return;
}

if (ToolPalette->PencilButton->Down)
{
    Image1->Canvas->Pen->Color = ToolPalette->FGShape->Brush->Color;
    Image1->Canvas->Brush->Color = ToolPalette->BGShape->Brush->Color;
    Image1->Canvas->MoveTo(X,Y);
    return;
}

// At this point, we know we are dealing with a circle, solid circle,
// square, or solid square

InitialX = X;
InitialY = Y;

// TmpImage will hold the image as it existed when the mouse went down.
// This will be retained until the mouse is released. This image will be
// used to restore the image during sizing of squares and circles.
TmpImage = new TImage(this);
TmpImage->Picture = Image1->Picture;
}
//-----
void __fastcall TDoodleForm::Image1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
    // Only the left mouse button is of interest
    if (!Shift.Contains(ssLeft))
        return;

    if (ToolPalette->FillButton->Down)
        return;

    if (ToolPalette->PencilButton->Down)
    {
        Image1->Canvas->LineTo(X,Y);
        return;
    }

    if (ToolPalette->EraseButton->Down)
    {
```

```
        Image1->Canvas->LineTo(X,Y);
        return;
    }

    // At this point, we know we are dealing with an ellipse or rectangle,
    // filled or not filled.
    DrawShape(X, Y);
}
//-----
void __fastcall TDoodleForm::Image1MouseUp(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    // Only the left mouse button is of interest
    if (Button != mbLeft)
        return;

    if ((ToolPalette->FillButton->Down) || (ToolPalette->PencilButton->Down))
        return;

    if (ToolPalette->EraseButton->Down)
    {
        Image1->Canvas->Pen->Width = 1;
        return;
    }

    // At this point, we know we are dealing with an ellipse or rectangle,
    // filled or not filled.
    DrawShape(X, Y);

    delete TmpImage;
}
//-----
void __fastcall TDoodleForm::FormActivate(TObject *Sender)
{
    // Contains the code for positioning and showing the tool palette.
    // The call to ToolPalette->Show() could have gone in DoodleForm's
    // Show method if not for the desire to position the window.

    if (ToolPalette->Visible)
        return;

    // Initial position of the palette relative to the main form
    ToolPalette->Top = DoodleForm->Top + (DoodleForm->Height / 6);
    ToolPalette->Left = DoodleForm->Left - ((DoodleForm->Width / 5));
    ToolPalette->Show();
}
//-----
void __fastcall TDoodleForm::DrawShape(int X, int Y)
{
    TRect bounds; // for graphics functions which require a rect

    // Start with the original image that we saved when the button
    // first went down. This blows away any previous rectangles or ellipses
```

```

// that were drawn while the user dragged.
Image1->Picture = TmpImage->Picture;

// The above line blew away the brush. The line below restores it.
Image1->Canvas->Brush->Color = ToolPalette->FGShape->Brush->Color;
Image1->Canvas->Pen->Color = ToolPalette->FGShape->Brush->Color;

// Some graphics functions do not allow a rectangle with its bottom above
// its top, or right to the left of its left side. This code is to deal
// with the case that the user started dragging from the bottom or right.
if (X < InitialX)
{
    bounds.Left = X;
    bounds.Right = InitialX;
}
else
{
    bounds.Right = X;
    bounds.Left = InitialX;
}

if (Y < InitialY)
{
    bounds.Top = Y;
    bounds.Bottom = InitialY;
}
else
{
    bounds.Bottom = Y;
    bounds.Top = InitialY;
}

// Draw the circle or square using the corresponding function.
if (ToolPalette->CircleButton->Down)
    Image1->Canvas->Arc(InitialX, InitialY, X, Y, X, Y, X, Y);
else if (ToolPalette->SolidCirButton->Down)
    Image1->Canvas->Ellipse(InitialX, InitialY, X, Y);
else if (ToolPalette->SquareButton->Down)
    Image1->Canvas->FrameRect(bounds);
else if (ToolPalette->SolidSqButton->Down)
    Image1->Canvas->FillRect(bounds);
}
//-----
void __fastcall TDoodleForm::FormShow(TObject *Sender)
{
    SetFocus();
}
//-----

void __fastcall TDoodleForm::FormClose(TObject *Sender,
    TCloseAction &Action)
{
    Application->Terminate();
}

```

```

}
//-----

Tập tin palette.cpp:
#include <vcl.h>
#pragma hdrstop

#include "palette.h"
#include "main.h"
//-----
#pragma link "CGRID"
#pragma resource "*.dfm"
TToolPalette *ToolPalette;
//-----
__fastcall TToolPalette::TToolPalette(TComponent* Owner)
: TForm(Owner)
{
}
//-----
void __fastcall TToolPalette::ColorGrid1Change(TObject *Sender)
{
    // Set the colors of the glyphs that show foreground and background color.
    // Since the ColorGrid already displays FG and BG, the glyphs aren't really
    // needed. Note that in the case of the color being black, the color of
    // the boarder of the glyph (aka the Pen) is changed to white. This is
    // just so that the frame can still be seen against the fill color.
    FGShape->Brush->Color = ColorGrid1->ForegroundColor;
    if (FGShape->Brush->Color == TColor(clBlack))
        FGShape->Pen->Color = TColor(clWhite);
    else
        FGShape->Pen->Color = TColor(clBlack);

    BGShape->Brush->Color = ColorGrid1->BackgroundColor;
    if (BGShape->Brush->Color == TColor(clBlack))
        BGShape->Pen->Color = TColor(clWhite);
    else
        BGShape->Pen->Color = TColor(clBlack);
}
//-----
void __fastcall TToolPalette::ShapeButtonClick(TObject *Sender)
{
    DoodleForm->Cursor = TCursor(crPlus);
}
//-----
void __fastcall TToolPalette::FillButtonClick(TObject *Sender)
{
    DoodleForm->Cursor = TCursor(crFill);
}
//-----
void __fastcall TToolPalette::PencilButtonClick(TObject *Sender)
{
    DoodleForm->Cursor = TCursor(crDraw);
}

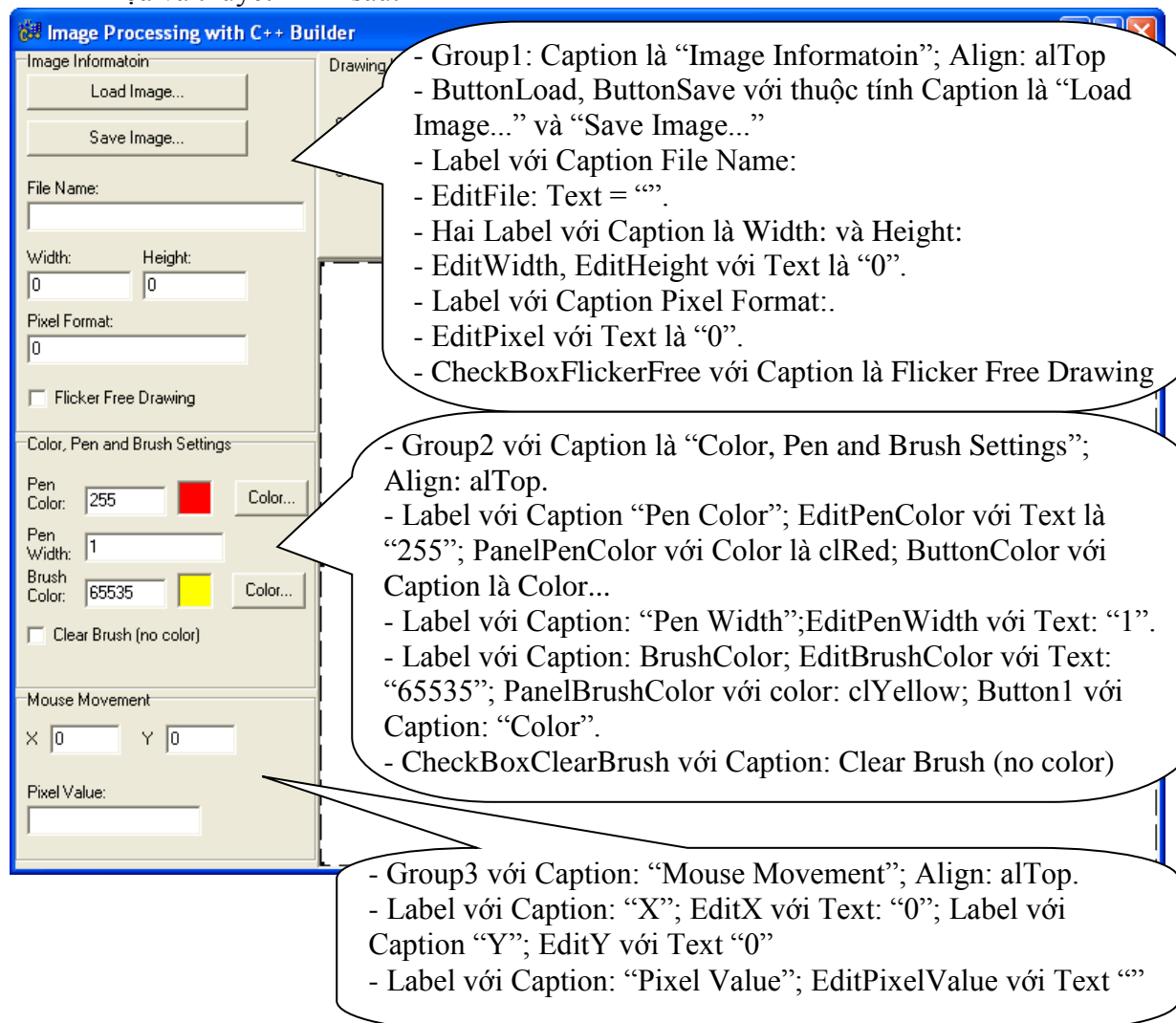
```

```
//-----
void __fastcall TToolPalette::EraseButtonClick(TObject *Sender)
{
    DoodleForm->Cursor = TCursor(crErase);
}
//-----
```

Chương trình thứ 2 phức tạp hơn, một chương trình xử lý trực tiếp trên hình ảnh của cửa hình vẽ.

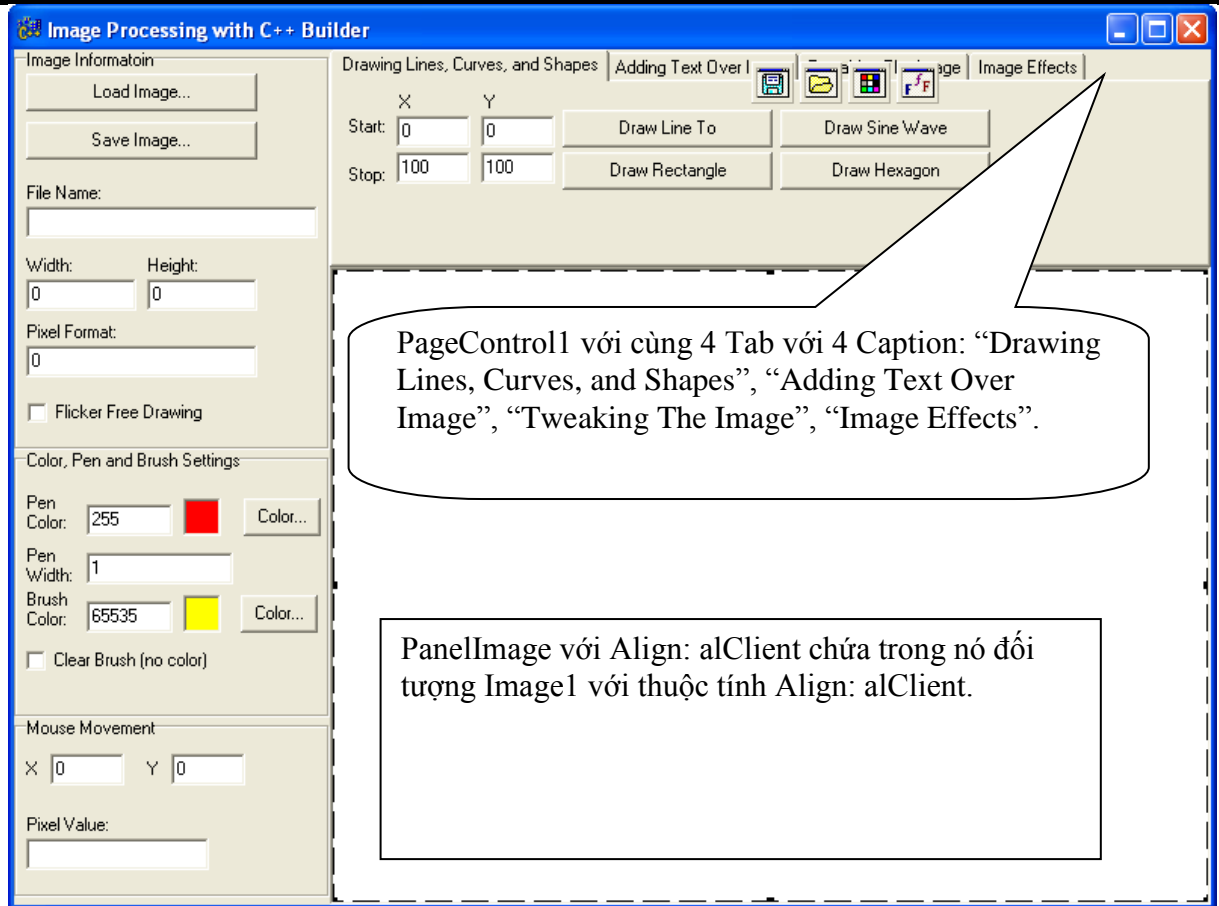
Bước 1: Đầu tiên, chúng ta thêm vào form bốn đối tượng ColorDialog1, FontDialog1, OpenFileDialog1, SaveDialog1.

Bước 2: Đặt đối tượng Panel1 với thuộc tính Align là alLeft. Tiến hành đặt các đối tượng theo hình minh họa và thuyết minh sau:



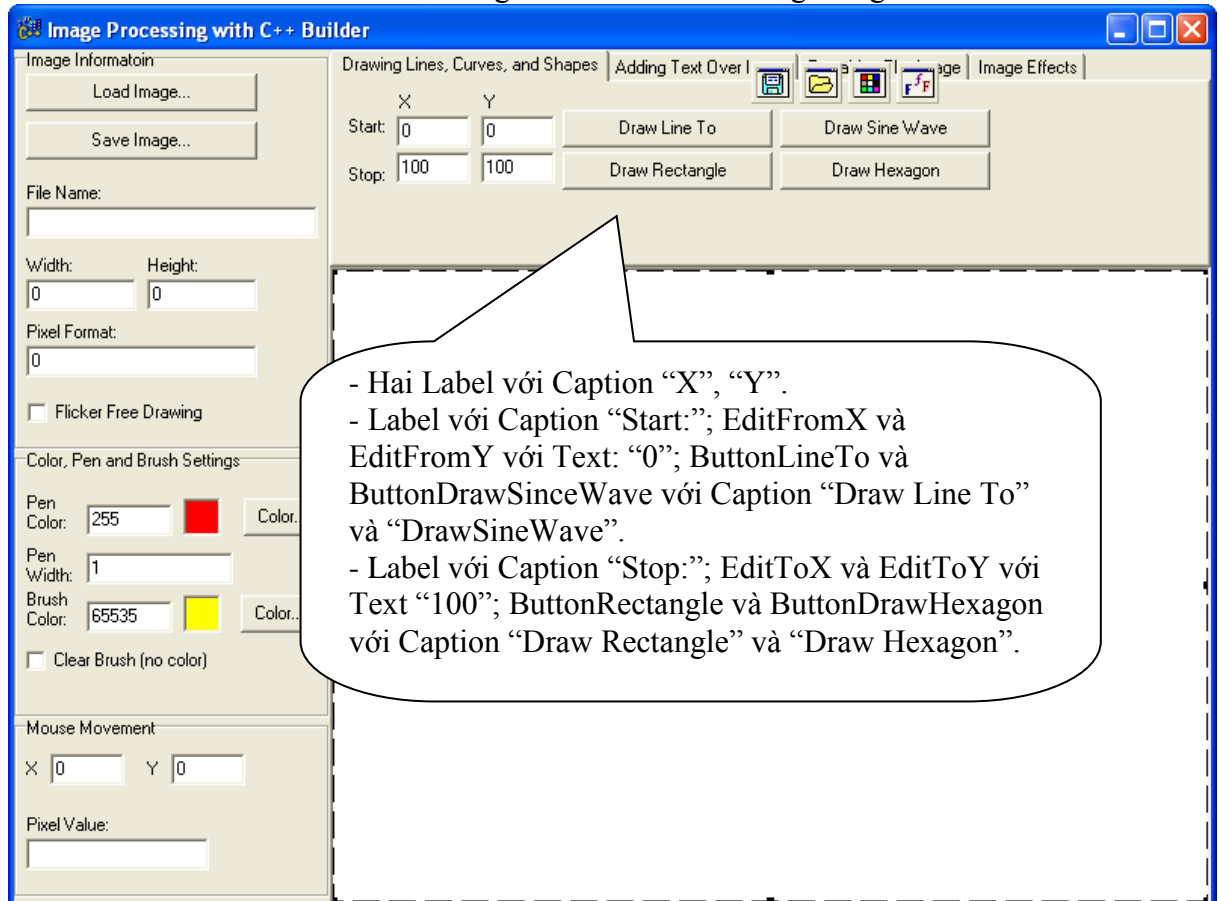
Hình 89- Thiết kế PanelLeft

Bước 3: Thêm đối tượng PanelRight với Align: alClient.



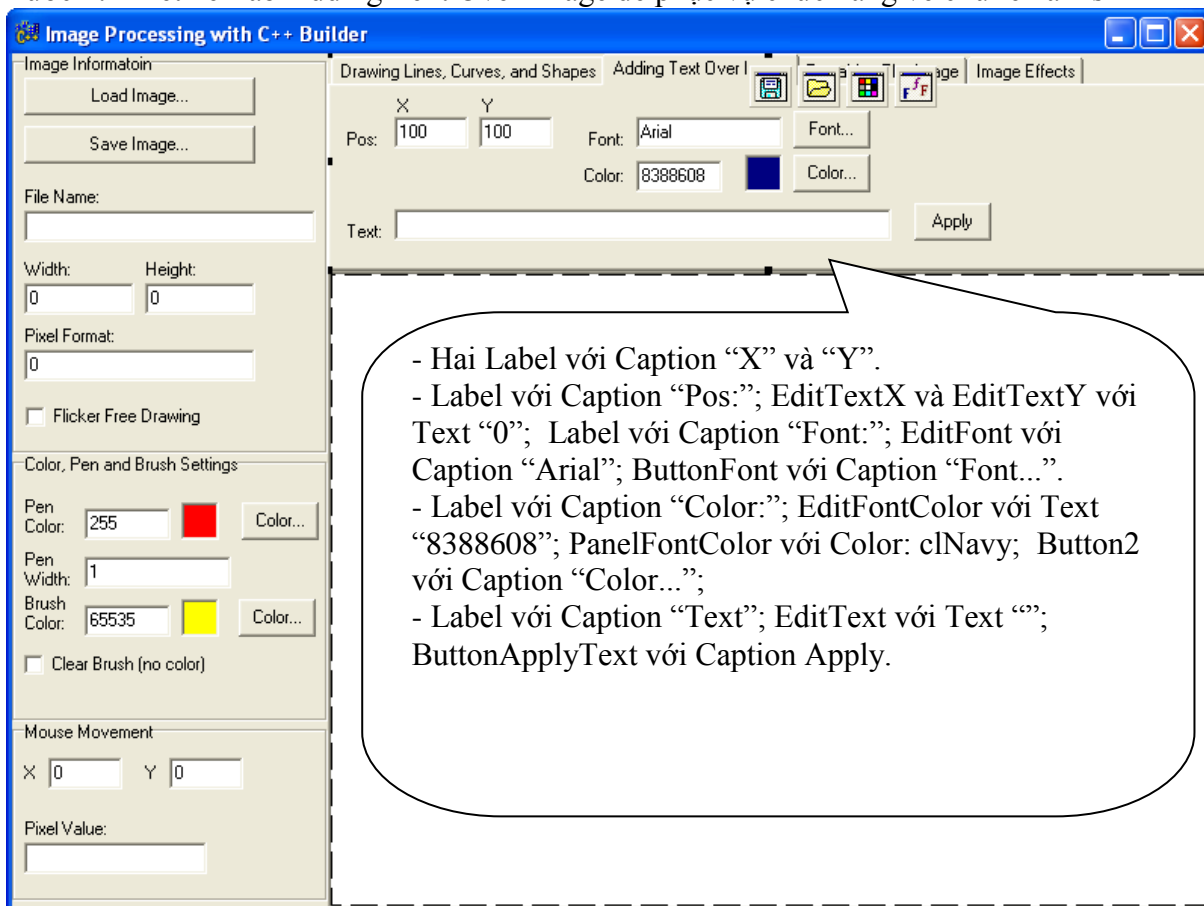
Hình 90- Thiết kế PanelRight

Bước 3: Thiết kế Tab đầu tiên Drawing Lines để vẽ các đường thẳng lên ảnh.



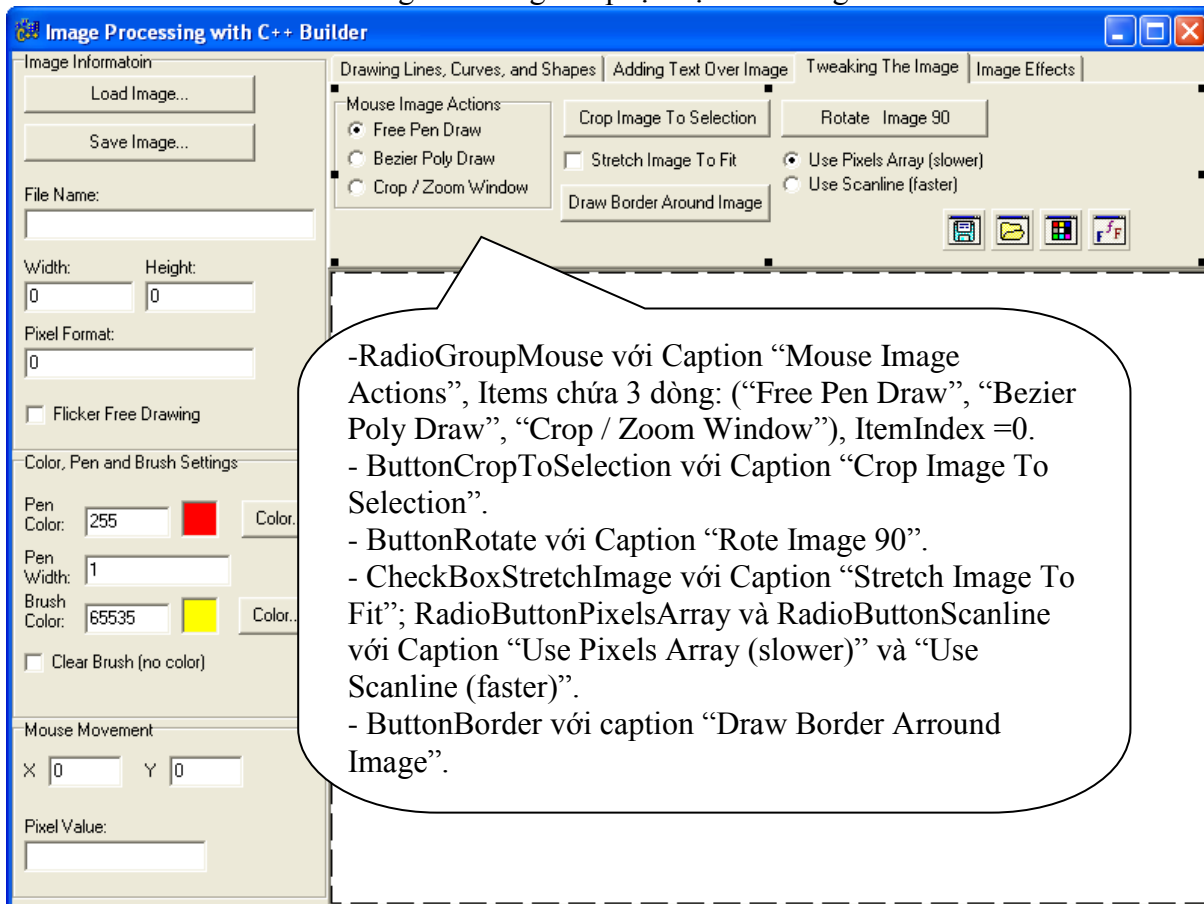
Hình 91- Thiết kế Tab Drawing Lines

Bước 4: Thiết kế Tab Adding Text Over Image để phục vụ chức năng vẽ chữ lên ảnh



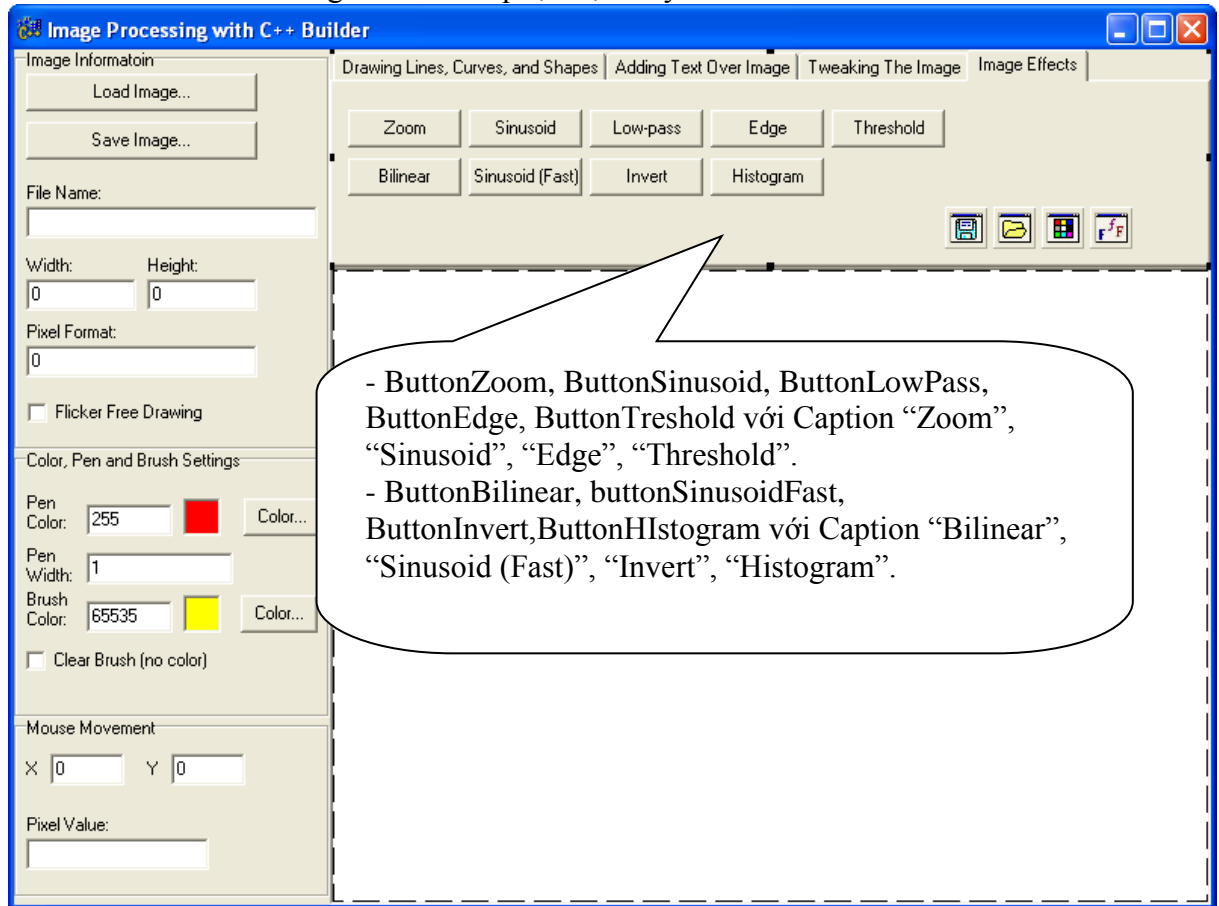
Hình 92- Thiết kế Tab Adding Text Over Image

Bước 5: Thiết kế Tab Tweaking The Image để phục vụ chức năng cắt xén ảnh.



Hình 93- Thiết kế Tab Tweaking The Image

Bước 6: Thiết kế tab Image Effects để phục vụ xử lý ảnh



Hình 94- Thiết kế Tab Image Effects

Bước 7: Sau khi thiết kế xong Form chúng ta viết lệnh cho form với mã lệnh được liệt kê sau đây (lưu ý tập tin unit: ImageProcessForm:

```
#include <vcl.h>
#include <math.h>
#include <jpeg.hpp>
#pragma hdrstop
```

```
#include "ImageProcessForm.h"
```

```
//-----
```

```
#pragma package(smart_init)
```

```
#pragma resource "*.dfm"
```

```
int Bilinear(BYTE Image[][256], float i, float j)
```

```
{
    int x1, y1, x2, y2;
    float Gray;

    x1 = (int)floor(i);
    x2 = x1+1;
    y1 = (int)floor(j);
    y2 = y1+1;
    Gray = (y2-j)*(x2-i)*Image[x1][y1] + (y2-j)*(i-x1)*Image[x2][y1]
          + (j-y1)*(x2-i)*Image[x1][y2] + (j-y1)*(i-x1)*Image[x2][y2];
    return INT(Gray);
}
```

```

int Lowpass(BYTE Image[][256], int x, int y)
{
    int Mask[3][3] = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}};
    int i, j, sum=0;

    for (j=-1; j<=1; j++)
        for (i=-1; i<=1; i++)
            sum += Image[x+i][y+j]*Mask[i+1][j+1];
    return INT(sum/9.0);
}

int Edge(BYTE Image[][256], int x, int y)
{
    int Mask[3][3] = {{-1, -1, -1}, {-1, 9, -1}, {-1, -1, -1}};
    int i, j, sum=0;

    for (j=-1; j<=1; j++)
        for (i=-1; i<=1; i++)
            sum += Image[x+i][y+j]*Mask[i+1][j+1];
    return INT(sum/9.0);
}

TForm1 *Form1;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    MouseDown = false;
    BezierSequence = 0;
    NumCurves    = 0;
    CurrentPoint  = 0;

    oX = -1;
    oY = -1;
    lX = -1;
    lY = -1;
}
//-----

void __fastcall TForm1::WMEraseBkgnd(TMessage Msg)
{
    //    if (CheckBoxEraseBackground->Checked)
    //        TForm::
    //            WndProc(Msg); //inherited; //  Erase the backGround
}

//-----
void __fastcall TForm1::ButtonLoadClick(TObject *Sender)
{
    //This code requires "jpeg.hpp" to be included in the source file
    OpenFileDialog->Filter =
        "Bmp files (*.bmp)|*.BMP| JPEG images (*.jpg) | *.jpg; ";
    if (OpenDialog1->Execute())

```

```

{
    if (!FileExists(OpenDialog1->FileName))
        return; // make sure it exists, else get out.
    AnsiString temp2 = ExtractFileName(OpenDialog1->FileName);
    AnsiString temp = ExtractFileExt(OpenDialog1->FileName);
    AnsiString Ext = temp.LowerCase();

    if (Ext.AnsiPos(".jpg") > 0) // it's a jpg
    {
        //-- Decompress the jpeg image into a bitmap.
        TJPEGImage *myjpeg = new TJPEGImage();
        myjpeg->LoadFromFile(OpenDialog1->FileName);
        myjpeg->DIBNeeded(); // used when jpeg image needs bitmap rep
        Image1->Picture->Bitmap->Assign(myjpeg);
        delete myjpeg;
    }
    else if (Ext.AnsiPos(".bmp") > 0)
    {
        Image1->Picture->Bitmap->LoadFromFile(OpenDialog1->FileName);
    }
    EditFile->Text      = ExtractFileName(OpenDialog1->FileName);
    EditWidth->Text     = Image1->Width;
    EditHeight->Text    = Image1->Height;
    EditPixelFormat->Text = Image1->Picture->Bitmap->PixelFormat;
}
}
//-----

void __fastcall TForm1::ButtonSinusoidClick(TObject *Sender)
{
    int x, y;
    int Amplitude, Gray;
    float Period;
    TColor RGB;

    Image1->Picture->Bitmap = new Graphics::TBitmap;
    Image1->Picture->Bitmap->PixelFormat = pf24bit;
    Image1->Picture->Bitmap->Width = 256;
    Image1->Picture->Bitmap->Height = 256;
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            Amplitude = 64*(255-y)/255;
            Period = 100*sqrt(1/(1+(exp(0.013*x)*exp(0.027*x)/400)));
            Gray = Amplitude*sin(2*M_PI/Period*x)+128;
            RGB = (Graphics::TColor)((Gray << 16) | (Gray << 8) | Gray);
            Image1->Canvas->Pixels[x][y] = RGB;
        }
}
//-----

void __fastcall TForm1::ButtonSinusoidFastClick(TObject *Sender)
{

```

```

int x, y;
int Amplitude;
float Period;
BYTE ImageData[256][256], *LinePtr;

Image1->Picture->Bitmap = new Graphics::TBitmap;
Image1->Picture->Bitmap->PixelFormat = pf24bit;
Image1->Picture->Bitmap->Width = 256;
Image1->Picture->Bitmap->Height = 256;
for (y=0; y<=255; y++)
    for (x=0; x<=255; x++)
    {
        Amplitude = 64*(255-y)/255;
        Period = 100*sqrt(1/(1+(exp(0.013*x)*exp(0.027*x)/400)));
        ImageData[x][y] = Amplitude*sin(2*M_PI/Period*x)+128;
    }
// Copy the image data to TBitmap
for (y=0; y<=255; y++)
{
    LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
    for (x=0; x<=255; x++)
    {
        LinePtr[x*3] = ImageData[x][y];    // Red
        LinePtr[x*3+1] = ImageData[x][y];  // Green
        LinePtr[x*3+2] = ImageData[x][y];  // Blue
    }
}
Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonInvertClick(TObject *Sender)
{
    int x, y;
    BYTE* LinePtr;

    // image inverting
    for (y=0; y<=255; y++)
    {
        LinePtr=(BYTE*)Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            LinePtr[x] = 255 - LinePtr[x];
    }
    Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonSaveClick(TObject *Sender)
{
    //This code requires "jpeg.hpp" to be included in the source file
    // SaveDialog1->Filter =
    // "Bmp files (*.bmp)|*.BMP| JPEG images (*.jpg; *.jpeg) | *.jpg; *.jpeg" ;

```



```

SaveDialog1->Title = "Save Image";
SaveDialog1->DefaultExt = ".jpg";
// SaveDialog1->Filter = "JPG|*.jpg";
SaveDialog1->Filter =
    "JPEG images (*.jpg) | *.jpg; | Bmp files (*.bmp)|*.BMP" ;
SaveDialog1->FilterIndex = 1;
if (SaveDialog1->Execute())
{
    AnsiString temp2 = ExtractFileName(SaveDialog1->FileName);
    AnsiString temp = ExtractFileExt(SaveDialog1->FileName);
    AnsiString Ext = temp.LowerCase();

    if (Ext.AnsiPos(".jpg") > 0) // it's a jpg
    {
        //-- Decompress the jpeg image into a bitmap.
        TJPEGImage *jp = new TJPEGImage();
        try
        {
            jp->Assign(Image1->Picture->Bitmap);
            jp->SaveToFile(SaveDialog1->FileName);
        }
        __finally
        {
            delete jp;
        }
    }
    else if (Ext.AnsiPos(".bmp") > 0)
    {
        Image1->Picture->Bitmap->SaveToFile(SaveDialog1->FileName);
    }
}

}
//-----

void __fastcall TForm1::ButtonZoomClick(TObject *Sender)
{
    BYTE ImageData[256][256], *LinePtr;
    BYTE Output[256][256];
    int x, y, i, j;
    int zoom=4, x0=127, y0=127;

    // Copy the image data to TBitmap
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            ImageData[x][y] = LinePtr[x];
    }
    // Calculate the output image
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            i = INT((x+(zoom-1)*x0) / zoom );

```

```

        j = INT((y+(zoom-1)*y0) / zoom );
        Output[x][y] = ImageData[i][j];
    }
    // copy the image back for display
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            LinePtr[x] = Output[x][y];
    }
    Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonBilinearClick(TObject *Sender)
{
    BYTE ImageData[256][256], *LinePtr;
    BYTE Output[256][256];
    int x, y;
    float i, j, zoom=4.0;
    int x0=127, y0=127;

    // Copy the image data to TBitmap
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            ImageData[x][y] = LinePtr[x];
    }
    // Calculate the out image
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
        {
            i = (x+(zoom-1)*x0) / zoom;
            j = (y+(zoom-1)*y0) / zoom;
            Output[x][y] = Bilinear(ImageData, i, j);
        }
    // copy the image back for display
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            LinePtr[x] = Output[x][y];
    }
    Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonLowPassClick(TObject *Sender)
{
    BYTE ImageData[256][256], *LinePtr;
    BYTE Output[256][256];
    int x, y;

```

```

// Copy the image data to TBitmap
for (y=0; y<=255; y++)
{
    LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
    for (x=0; x<=255; x++)
        ImageData[x][y] = LinePtr[x];
}
// Calculate the low-pass image
for (y=1; y<=254; y++)
    for (x=1; x<=254; x++)
        Output[x][y] = Lowpass(ImageData, x, y);
// copy the image back for display
for (y=0; y<=255; y++)
{
    LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
    for (x=0; x<=255; x++)
        LinePtr[x] = Output[x][y];
}
Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonEdgeClick(TObject *Sender)
{
    BYTE ImageData[256][256], *LinePtr;
    BYTE Output[256][256];
    int x, y;

    // Copy the image to ImageData
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            ImageData[x][y] = LinePtr[x];
    }
    // Calculate the edge image
    for (y=1; y<=254; y++)
        for (x=1; x<=254; x++)
            Output[x][y] = Edge(ImageData, x, y);
    // copy the image back for display
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            LinePtr[x] = Output[x][y];
    }
    Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonHistogramClick(TObject *Sender)
{

```

```

    unsigned int Histogram[256], Transform[256];
    BYTE ImageData[256][256], *LinePtr;
    int x, y, i, j, sum;

    // Copy the image to ImageData
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            ImageData[x][y] = LinePtr[x];
    }
    // initialize the histogram
    for (i=0; i<=255; i++)
        Histogram[i] = 0;
    // Construct the histogram
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
            Histogram[ImageData[x][y]]++;
    // Compute the transformation
    for (i=0; i<=255; i++)
    {
        sum = 0;
        for (j=0; j<=i; j++)
            sum += Histogram[j];
        Transform[i] = INT(255.0*sum/(256*256));
    }
    // Transform the image
    for (y=0; y<=255; y++)
        for (x=0; x<=255; x++)
            ImageData[x][y] = Transform[ImageData[x][y]];
    // copy the image back for display
    for (y=0; y<=255; y++)
    {
        LinePtr = (BYTE *) Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
            LinePtr[x] = ImageData[x][y];
    }
    Image1->Refresh();
}
//-----

```

```

void __fastcall TForm1::ButtonThresholdClick(TObject *Sender)
{
    int x, y;
    BYTE* LinePtr;

    // image thresholding
    for (y=0; y<=255; y++)
    {
        LinePtr=(BYTE*)Image1->Picture->Bitmap->ScanLine[y];
        for (x=0; x<=255; x++)
        {
            if (LinePtr[x] > 128)

```

```
        LinePtr[x] = 255;
    else
        LinePtr[x] = 0;
    }
}
Image1->Refresh();
}

//-----

void __fastcall TForm1::FreeZoomWindow()
{
    if (oX != -1)
    {
        TRect MyRect = TRect(oX,oY,lX,lY);
        Image1->Canvas->Rectangle(MyRect);
    }
    oX = -1;
    oY = -1;
    lX = -1;
    lY = -1;
}

//-----

void __fastcall TForm1::Image1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    if (Button == mbLeft)
    {
        Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0));
        Image1->Canvas->Pen->Style = psSolid; // solid line
        Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);
        if (RadioGroupMouse->Items->Strings[RadioGroupMouse->ItemIndex] ==
            "Bezier Poly Draw")
        {
            points[CurrentPoint].x = X;
            points[CurrentPoint].y = Y;

            if ((CurrentPoint == 0) || (BezierSequence == 2))
            {
                if (CurrentPoint > 0) NumCurves++;
                Image1->Canvas->Ellipse(X-2,Y-2, X+2, Y+2); // temporary
            }
            else // control point
                Image1->Canvas->Ellipse(X-1,Y-1, X+1, Y+1); // temporary

            BezierSequence++;
            CurrentPoint++;
            if (BezierSequence >= 3)
                BezierSequence = 0;
        }
    }
}
```

```

else if (RadioGroupMouse->Items->Strings[RadioGroupMouse->ItemIndex] ==
    "Free Pen Draw")// watch mouse move
{
    if (CheckBoxFlickerFree->Checked)
        PanelImage->DoubleBuffered = true;
    Image1->Canvas->MoveTo(X,Y);
    MouseDown = true;
    Image1->Canvas->Pen->Mode = pmCopy;
}
else if (RadioGroupMouse->Items->Strings[RadioGroupMouse->ItemIndex] ==
    "Crop / Zoom Window")// watch mouse move
{
    Image1->Canvas->Pen->Color = clBlack;
    Image1->Canvas->Pen->Style = psDash; // dashed line
    Image1->Canvas->Pen->Width = 1;
    Image1->Canvas->Pen->Mode = pmNotXor;
    Image1->Canvas->Brush->Style = bsClear;
    FreeZoomWindow();
    oX = X;
    oY = Y;
    lX = X;
    lY = Y;
    DrawSelectRegion = true;
    if (CheckBoxFlickerFree->Checked)
        PanelImage->DoubleBuffered = true;
}
EditFromX->Text = X; // this is useful for marking positions to draw
EditFromY->Text = Y;
}
else if (Button == mbRight)
{
    if (RadioGroupMouse->Items->Strings[RadioGroupMouse->ItemIndex] ==
        "Bezier Poly Draw") // release and draw Bezier
    {
        Image1->Canvas->MoveTo(points[0].x, points[0].y);
        int size = ((NumCurves)*3);
        Image1->Canvas->PolyBezierTo(points, size-1); //BezierSequence-1);
        BezierSequence = 0;
        NumCurves = 0;
        CurrentPoint = 0;
    }
    else // mark the position in the Font Edit boxes
    {
        EditTextX->Text = X;
        EditTextY->Text = Y;
        EditToX->Text = X;
        EditToY->Text = Y;
    }
}
}

//-----

```

```

void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
    EditX->Text = X;
    EditY->Text = Y;
    EditPixelValue->Text = Image1->Picture->Bitmap->Canvas->Pixels[X][Y];
    if (MouseDown)    // follow movement of the mouse
    {
        Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0));
        Image1->Canvas->LineTo(X,Y);
    //    Image1->Canvas->Pixels[X][Y] = clWhite;
    }
    else if (DrawSelectRegion)
    {
        TRect MyRect = TRect(oX,oY,lX,lY);
        Image1->Canvas->Rectangle(MyRect);
        lX = X;
        lY = Y;
        MyRect = TRect(oX,oY,lX,lY);
        Image1->Canvas->Rectangle(MyRect);
    }
}

//-----

void __fastcall TForm1::Image1MouseUp(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    // if (DrawSelectRegion)
    // {
    //     TRect MyRect = TRect(oX,oY,lX,lY);
    //     Image1->Canvas->Rectangle(MyRect);
    // }
    DrawSelectRegion = false;
    MouseDown = false;
    PanelImage->DoubleBuffered = false;
    EditToX->Text = X;
    EditToY->Text = Y;
}
//-----

void __fastcall TForm1::ButtonBorderClick(TObject *Sender)
{
    // let's draw a border around the image
    TRect MyRect(0,0,Image1->Width,Image1->Height);
    Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0)); //clBlue;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);
    Image1->Canvas->Brush->Style = bsClear;
    Image1->Canvas->Rectangle(MyRect);
}
//-----

```

```
void __fastcall TForm1::ButtonColorClick(TObject *Sender)
{
    ColorDialog1->Color = (TColor)EditPenColor->Text.ToIntDef(0);
    if (ColorDialog1->Execute())
    {
        EditPenColor->Text = ColorDialog1->Color;
        PanelPenColor->Color = ColorDialog1->Color;
    }
}
```

```
//-----
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    ColorDialog1->Color = (TColor)EditBrushColor->Text.ToIntDef(0);
    if (ColorDialog1->Execute())
    {
        EditBrushColor->Text = ColorDialog1->Color;
        PanelBrushColor->Color = ColorDialog1->Color;
    }
}
//-----
```

```
void __fastcall TForm1::ButtonLineToClick(TObject *Sender)
{
    Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0)); //clBlue;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);

    int x1 = EditFromX->Text.ToIntDef(0);
    int y1 = EditFromY->Text.ToIntDef(0);
    int x2 = EditToX->Text.ToIntDef(0);
    int y2 = EditToY->Text.ToIntDef(0);
    Image1->Canvas->MoveTo(x1,y1);
    Image1->Canvas->LineTo(x2,y2);
}
//-----
```

```
void __fastcall TForm1::ButtonRectangleClick(TObject *Sender)
{
    Image1->Canvas->Brush->Color = TColor(EditBrushColor->Text.ToIntDef(0)); //clBlue;
    if (CheckBoxClearBrush->Checked)
        Image1->Canvas->Brush->Style = bsClear; // clear rectangle
    else
        Image1->Canvas->Brush->Style = bsSolid;
    Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0)); //clBlue;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);
```

```
    int x1 = EditFromX->Text.ToIntDef(0);
```



```

    int y1 = EditFromY->Text.ToIntDef(0);
    int x2 = EditToX->Text.ToIntDef(0);
    int y2 = EditToY->Text.ToIntDef(0);
    TRect rect(x1,y1,x2,y2);

    if (CheckBoxFlickerFree->Checked)
        PanelImage->DoubleBuffered = true;
    Image1->Canvas->Rectangle(rect);
    PanelImage->DoubleBuffered = false;
}

//-----

void __fastcall TForm1::CheckBoxBezierClick(TObject *Sender)
{
    BezierSequence = 0;
    NumCurves      = 0;
    CurrentPoint = 0;
}
//-----

void __fastcall TForm1::ButtonDrawCurveShapeClick(TObject *Sender)
{
    Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0)); //clBlue;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);

    // let's draw a sine wave...

    TPoint point[30];
    double pi = 3.1514927;
    point[0].x = 0;
    point[0].y = 100;
    point[1].x = 0;
    point[1].y = 100;
    for (int i = 1; i<10; i++)
    {
        int j = i*3 -1;
        point[j].x = i*10;
        point[j].y = (sin(i*(pi/2))*100) + 100;
        point[j+1].x = i*10;
        if (point[j].y > 0)
            point[j+1].y = 300;
        else if (point[j].y == 0)
            point[j+1].y = 0;
        else // < 0
            point[j+1].y = 100;

        point[j+2].x = i*10;
        if (point[j].y > 0)
            point[j+2].y = 300;
        else if (point[j].y == 0)

```

```

        point[j+2].y = 0;
    else // < 0
        point[j+2].y = 100;

    }
// Image1->Canvas->MoveTo(point[0].x,point[1].y);
Image1->Canvas->PolyBezier(point, 29); //BezierSequence-1);
}
//-----

void __fastcall TForm1::ButtonDrawHexagonClick(TObject *Sender)
{
    Image1->Canvas->Brush->Color = TColor(EditBrushColor->Text.ToIntDef(0));
    if (CheckBoxClearBrush->Checked)
        Image1->Canvas->Brush->Style = bsClear; // clear shape
    else
        Image1->Canvas->Brush->Style = bsSolid;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);

    int startx = EditFromX->Text.ToIntDef(0);
    int starty = EditFromY->Text.ToIntDef(0);
    int endx   = EditToX->Text.ToIntDef(0);
    int endy   = EditToY->Text.ToIntDef(0);

    int lengthx = abs(endx - startx);
    int lengthy = abs(endy - starty);

    TPoint points[5];
    points[0] = Point(startx, starty + (lengthy*0.4));
    points[1] = Point(startx + (lengthx*0.5), starty);
    points[2] = Point(startx + (lengthx), starty + (lengthy*0.4));
    points[3] = Point(startx + (lengthx*0.7), starty + lengthy);
    points[4] = Point(startx + (lengthx*0.3), starty + lengthy);

    if (CheckBoxFlickerFree->Checked)
        PanelImage->DoubleBuffered = true;
    Image1->Canvas->Polygon(points, 4);
    PanelImage->DoubleBuffered = false;
}
//-----

void __fastcall TForm1::ButtonDrawSineWaveClick(TObject *Sender)
{
    Image1->Canvas->Pen->Color = TColor(EditPenColor->Text.ToIntDef(0)); //clBlue;
    Image1->Canvas->Pen->Style = psSolid; // solid line
    Image1->Canvas->Pen->Width = EditPenWidth->Text.ToIntDef(1);

    const int maxpts = 19;
    TPoint pts[maxpts];
    double pi_3 = 3.1415926535897932384626433832795 / 3.0;

```

```

int startx = EditFromX->Text.ToIntDef(0);
int starty = EditFromY->Text.ToIntDef(0);

for (int i = 0; i<maxpts; i++)
{
    pts[i].x = (i+1)*10 + startx;
    pts[i].y = (sin(i*pi_3)*200) + starty; //300; //get back to axis (pi) every 3rd one
}

Image1->Canvas->PolyBezier(EXISTINGARRAY(pts));

/* Show the points as well as the curve */
for (int i = 0; i<maxpts; i++)
{
    Image1->Canvas->Rectangle(pts[i].x-3,pts[i].y-3,pts[i].x+3,pts[i].y+3);
}
}
//-----

void __fastcall TForm1::ButtonFontClick(TObject *Sender)
{
    FontDialog1->Font->Color = TColor(EditFontColor->Text.ToIntDef(0));
    if (FontDialog1->Execute())
    {
        EditFont->Text = FontDialog1->Font->Name;
        Image1->Canvas->Font->Name = FontDialog1->Font->Name;
        Image1->Canvas->Font->Style = FontDialog1->Font->Style;
        Image1->Canvas->Font->Size = FontDialog1->Font->Size;
        Image1->Canvas->Font->Color = FontDialog1->Font->Color;
    }
}
//-----

void __fastcall TForm1::ButtonApplyTextClick(TObject *Sender)
{
    Image1->Canvas->Brush->Color = TColor(EditBrushColor->Text.ToIntDef(0)); //clBlue;
    if (CheckBoxClearBrush->Checked)
        Image1->Canvas->Brush->Style = bsClear; // clear rectangle
    else
        Image1->Canvas->Brush->Style = bsSolid;
    Image1->Canvas->Font->Name = EditFont->Text;
    Image1->Canvas->Font->Color = TColor(EditFontColor->Text.ToIntDef(0));

    int x = EditTextX->Text.ToIntDef(0);
    int y = EditTextY->Text.ToIntDef(0);
    Image1->Canvas->TextOut(x,y,EditText->Text);
}
//-----

void __fastcall TForm1::ButtonRotateClick(TObject *Sender)
{
    //create and setup a temporary source bitmap

```

```

Graphics::TBitmap *source = new Graphics::TBitmap;
source->Assign(Image1->Picture->Bitmap);
source->PixelFormat = Image1->Picture->Bitmap->PixelFormat;

// create and setup a temporary destination bitmap
Graphics::TBitmap *dest = new Graphics::TBitmap;
dest->Width = source->Height;
dest->Height = source->Width;
dest->PixelFormat = source->PixelFormat;

if (RadioButtonPixelsArray->Checked) //Rotate one pixel at a time
{
    for (int x=0;x<source->Width;x++)
        for(int y=0;y<source->Height;y++)
            dest->Canvas->Pixels[y][source->Width-1-x] =
                source->Canvas->Pixels[x][y];
}
else // uses the faster scanline method
{
    RGBTRIPLE* pixels;
    TColor color;
    for(int y=0;y<source->Height;y++)
    {
        pixels = (RGBTRIPLE*)source->ScanLine[y];
        for (int x=0;x<source->Width;x++)
            dest->Canvas->Pixels[y][source->Width-1-x] =
                TColor(
                    RGB(
                        pixels[x].rgbtRed,
                        pixels[x].rgbtGreen,
                        pixels[x].rgbtBlue));
    }
}

//now assign destination bitmap back to Image1 & cleanup
Image1->Picture->Bitmap = dest;
delete dest;
delete source;
}
//-----

void __fastcall TForm1::ButtonCropToSelectionClick(TObject *Sender)
{
    // let's crop

    if (IX == -1) return; // no zoom window to work with

    // create and setup a temporary destination bitmap
    Graphics::TBitmap *dest = new Graphics::TBitmap;
    dest->Width = abs(IX - oX);
    dest->Height = abs(IY - oY);

    TRect OldOne = Rect(oX,oY,IX,IY);
    TRect NewOne = Rect(0,0,dest->Width, dest->Height);

```

```

FreeZoomWindow();

//create and setup a temporary source bitmap
Graphics::TBitmap *source = new Graphics::TBitmap;
source->Assign(Image1->Picture->Bitmap);
source->PixelFormat = Image1->Picture->Bitmap->PixelFormat;

dest->PixelFormat = source->PixelFormat;
dest->Canvas->CopyRect(NewOne,source->Canvas,OldOne);

//now assign destination bitmap back to Image1 & cleanup
Image1->Picture->Bitmap->FreeImage();
Image1->Picture->Bitmap->Assign(dest);
delete dest;
delete source;

}
//-----

void __fastcall TForm1::CheckBoxStretchImageClick(TObject *Sender)
{
    Image1->Stretch = CheckBoxStretchImage->Checked;
}

//-----

void __fastcall TForm1::Button2Click(TObject *Sender)
{
    ColorDialog1->Color = (TColor)EditFontColor->Text.ToIntDef(0);
    if (ColorDialog1->Execute())
    {
        EditFontColor->Text = ColorDialog1->Color;
        PanelFontColor->Color = ColorDialog1->Color;
    }
}

//-----

```

Bước 8: Nhấn phím F9 để kiểm thử chương trình.

5. Lập trình multimedia.

Trong phần này, chúng ta sẽ nghiên cứu các chức năng cơ bản của lập trình multimedia trong C++ Builder. Đầu tiên, chúng ta nghiên cứu phương pháp đưa vào các đoạn video một cách âm thầm vào ứng dụng của chúng ta.

Đối tượng hoạt hình trong C++ Builder cho phép chúng ta thêm các trích đoạn video một cách âm thầm vào trong ứng dụng. Để thêm một trích đoạn từ vào một ứng dụng, theo các bước sau:

1. Nhấn đôi chuột lên biểu tượng animate trên trang Win32 của bảng thành phần. Biểu tượng của điều khiển hoạt hình lên form mà bạn muốn hiển thị trích đoạn video.
2. Sử dụng Object Inspector, sử dụng thuộc tính Name và nhập vào tên mới. Chúng ta sử dụng tên này để gọi đối tượng animate.
3. Thực hiện theo các hành động sau:

Sử dụng thuộc tính CommonAVI và chọn một AVI cho phép từ danh sách đồ xuống hoặc sử dụng thuộc tính FileName để chọn tập tin AVI.

Chọn tài nguyên của một AVI sử dụng thuộc tính ResName và ResID. Sử dụng ResHandle để chỉ định mô đun chứa chỉ định tài nguyên bởi ResName và ResID.

Cái này sẽ tải tập tin AVI vào trong bộ nhớ. Nếu chúng ta muốn hiển thị khung đầu tiên của trích đoạn AVI trên màn hình cho đến khi người chơi sử dụng thuộc tính Active hay phương thức Play, sau đó cài đặt thuộc tính Open thành true.

4. Cài đặt thuộc tính Repetitions là số lần lặp lại khi chơi tập tin AVI này, nếu chỉ định là 0 thì có nghĩa lặp lại cho đến khi sử dụng phương thức Stop.

5. Tạo bất kỳ sự thay đổi nào khác cho các cài đặt của điều khiển animation. Ví dụ, nếu chúng ta muốn thay đổi khung hình đầu tiên sẽ hiển thị khi đối tượng hoạt hình được mở, hãy cài đặt giá trị StartFrame thành giá trị của khung hình.

6. Cài đặt giá trị thuộc tính Active thành true và viết mã lệnh thực thi cho sự kiện nếu cần.

Như đã thấy, sáu bước trên giúp chúng ta đưa các Video clip dạng AVI vào ứng dụng của chúng ta. Tiếp theo, chúng ta có thể sử dụng đối tượng TMediaPlayer để đưa âm thanh và video clip vào ứng dụng của chúng ta. Nó mở một thiết bị media và play, stop, pause, record, các âm thanh và video clip sử dụng thiết bị media. Thiết bị media có thể là phần cứng hay phần mềm:

Để thêm một âm thanh hay video clip vào một ứng dụng, chúng ta theo các bước sau:

1. Nhấn đôi lên biểu tượng MediaPlayer ở trang System của bảng thành phần.
2. Sử dụng cửa sổ Object Inspector để điều chỉnh tên thuộc tính.
3. Chọn thiết bị từ thuộc tính DeviceType.
4. Nếu thiết bị lưu trữ media của nó trong một tập tin. Hãy chỉ định tập tin bằng cách sử dụng thuộc tính FileName

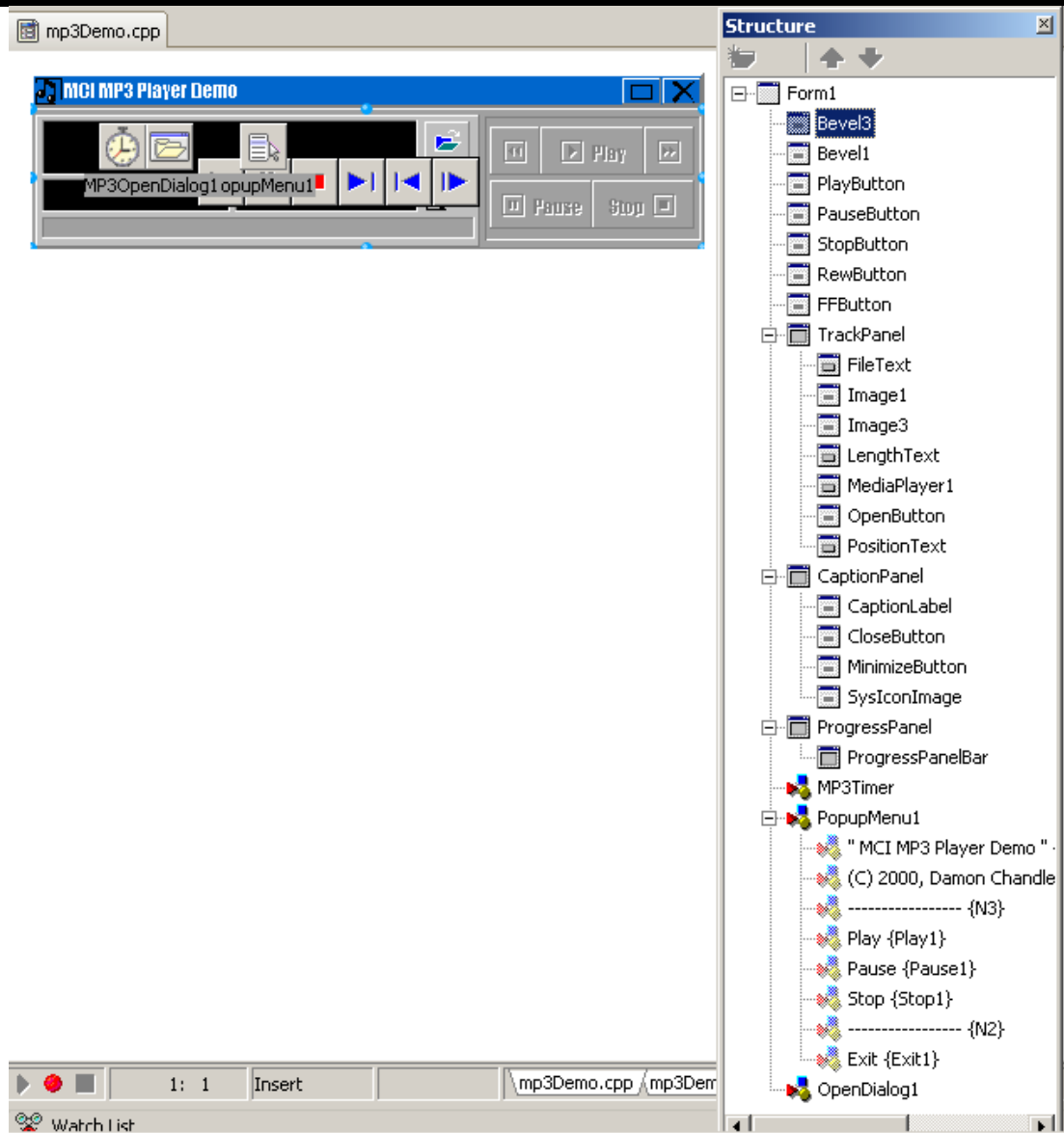
5. Cài đặt thuộc tính AutoOpen thành true.

6. Cài đặt thuộc tính AutoEnable property thành true.

7. Sử dụng thuộc tính EnabledButtons để cho phép hay không cho phép các nút nhấn hiển thị trên đối tượng. Khi đối tượng hoạt động có thể nhấn lên các nút nhấn hay viết mã lệnh để thực hiện chức năng tương ứng như Play, Pause, Stop, Next, Previous,

Để kết thúc nội dung phần này, chúng ta xem qua các bước thực hiện chương trình Mp3 Player

Bước 1: Tạo Form như hình sau với các liệt kê về cây đối tượng kèm theo dữ liệu của tập tin khai báo Form



Hình 95- Thiết kế chương trình Mp3 Player

Bước 2: Viết mã lệnh cho tập tin .cpp như sau:

Tập tin .h:

```
//-----
#ifndef mp3DemoH
#define mp3DemoH
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Buttons.hpp>
#include <Graphics.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <MPlayer.hpp>
//-----
```

```
class TForm1 : public TForm
```

```

{
__published:      // IDE-managed Components
    TBevel *Bevel3;
    TBevel *Bevel1;
    TSpeedButton *PlayButton;
    TSpeedButton *PauseButton;
    TSpeedButton *StopButton;
    TSpeedButton *RewButton;
    TSpeedButton *FFButton;
    TPanel *TrackPanel;
    TStaticText *FileText;
    TStaticText *LengthText;
    TStaticText *PositionText;
    TPanel *CaptionPanel;
    TSpeedButton *MinimizeButton;
    TSpeedButton *CloseButton;
    TLabel *CaptionLabel;
    TImage *SysIconImage;
    TTimer *MP3Timer;
    TPopupMenu *PopupMenu1;
    TMenuItem *MCICDPlayerDemo1;
    TMenuItem *C2000DamonChandler1;
    TMenuItem *N3;
    TMenuItem *Play1;
    TMenuItem *Pause1;
    TMenuItem *Stop1;
    TMenuItem *N2;
    TMenuItem *Exit1;
    TOpenDialog *OpenDialog1;
    TPanel *ProgressPanel;
    TPanel *ProgressPanelBar;
    TSpeedButton *OpenButton;
    TImage *Image1;
    TImage *Image3;
    TMediaPlayer *MediaPlayer1;
    void __fastcall OpenButtonClick(TObject *Sender);
    void __fastcall PlayButtonClick(TObject *Sender);
    void __fastcall CloseButtonClick(TObject *Sender);
    void __fastcall MinimizeButtonClick(TObject *Sender);
    void __fastcall CaptionPanelMouseDown(TObject *Sender,
        TMouseButton Button, TShiftState Shift, int X, int Y);
    void __fastcall MP3TimerTimer(TObject *Sender);
    void __fastcall StopButtonClick(TObject *Sender);
    void __fastcall PauseButtonClick(TObject *Sender);
    void __fastcall FFButtonClick(TObject *Sender);
    void __fastcall RewButtonClick(TObject *Sender);
    void __fastcall CaptionPanelDblClick(TObject *Sender);
    void __fastcall SysIconImageClick(TObject *Sender);
    void __fastcall PopupMenu1Popup(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);

private:      // User declarations
    long track_length_;

```

```

    long current_pos_;

    MESSAGE void __fastcall MMMciNotify(TMessage& AMsg);
    void __fastcall UpdateControls();
    void __fastcall UpdateLengthInfo();

    POINT PSnapped_;
    RECT RWork_;
    MESSAGE void __fastcall WMEnterSizeMove(TMessage& AMsg);
    MESSAGE void __fastcall WMMoving(TMessage& AMsg);

public:
    // User declarations
    __fastcall TForm1(TComponent* Owner);

BEGIN_MESSAGE_MAP
    MESSAGE_HANDLER(MM_MCINOTIFY, TMessage, MMMciNotify)
    MESSAGE_HANDLER(WM_ENTERSIZEMOVE, TMessage, WMEnterSizeMove)
    MESSAGE_HANDLER(WM_MOVING, TMessage, WMMoving)
END_MESSAGE_MAP(TForm)
};

//-----
extern PACKAGE TForm1 *Form1;
//-----
#endif

Tập tin .cpp:
#include <vcl.h>
#pragma hdrstop

#include <math.h>
#include "MP3Device.h"
#include "mp3Demo.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

static const int fixed_height = 105;
static const char fixed_caption[] = "MCI MP3 Player Demo";
//-----

// ***** BEGIN MCI-Related Code *****

__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner), current_pos_(0), track_length_(0)
{

}

//-----

void __fastcall TForm1::MMMciNotify(TMessage& AMsg)
{

```

```

    UpdateControls();
}
//-----

void __fastcall TForm1::UpdateLengthInfo()
{
    current_pos_ = 0;
    track_length_ = this->MediaPlayer1->Length/1000.0L;
    int mins = floor(track_length_ / 60);
    int secs = track_length_ - (mins * 60);

    LengthText->Caption = "[" + FormatFloat("00", mins) +
        ":" + FormatFloat("00", secs) +
        "]" total time";
}
//-----

void __fastcall TForm1::MP3TimerTimer(TObject *Sender)
{
    long pos = this->MediaPlayer1->Position;
    if (pos == this->MediaPlayer1->Length)
    {
        current_pos_ = 0;
        this->MediaPlayer1->Position = 0;
    }
    else current_pos_ = pos / 1000.0L;

    int mins = floor(current_pos_ / 60);
    int secs = current_pos_ - (mins * 60);

    if (track_length_ > 0)
    {
        float scale = static_cast<float>(current_pos_) / track_length_;
        ProgressPanelBar->Width = (ProgressPanel->Width - 3) * scale;
    }

    PositionText->Caption = "[" + FormatFloat("00", mins) +
        ":" + FormatFloat("00", secs) +
        "]" elapsed time";
    Application->Title = "[" + FileText->Caption.Trim() + "]" +
        PositionText->Caption;
    if (Height == CaptionPanel->Height)
    {
        CaptionLabel->Caption =
            fixed_caption + AnsiString(" -- ") + PositionText->Caption;
    }
    UpdateControls();
}
//-----

void __fastcall TForm1::UpdateControls()
{
    PlayButton->Enabled = (this->MediaPlayer1->Mode != mpPlaying);
}

```

```

        PauseButton->Enabled = (this->MediaPlayer1->Mode == mpPlaying);
        StopButton->Enabled = (this->MediaPlayer1->Mode == mpPlaying || this-
>MediaPlayer1->Mode == mpPaused);
        RewButton->Enabled = track_length_ > 0;
        FFButton->Enabled = track_length_ > 0;
    }
    //-----

void __fastcall TForm1::OpenButtonClick(TObject *Sender)
{
    if (OpenDialog1->Execute())
    {
        //this->MediaPlayer1->Stop();
        this->MediaPlayer1->FileName = OpenDialog1->FileName;
        this->MediaPlayer1->Open();
        this->MediaPlayer1->Play();
        UpdateLengthInfo();
        FileText->Caption = " " + ExtractFileName(OpenDialog1->FileName);
        MP3Timer->Enabled = true;
        /*bool was_playing = mp3_.playing();
        if (mp3_.open((char *)OpenDialog1->FileName.c_str()))
        {
            FileText->Caption =
                " " + ExtractFileName(OpenDialog1->FileName);
            UpdateLengthInfo();

            if (was_playing) mp3_.play();
        } */
    }
}
//-----

void __fastcall TForm1::PlayButtonClick(TObject *Sender)
{
    this->MediaPlayer1->Play();
}
//-----

void __fastcall TForm1::StopButtonClick(TObject *Sender)
{
    this->MediaPlayer1->Stop();
    this->MediaPlayer1->Position = 0;
}
//-----

void __fastcall TForm1::PauseButtonClick(TObject *Sender)
{
    this->MediaPlayer1->Pause();
}
//-----

void __fastcall TForm1::RewButtonClick(TObject *Sender)

```

```

{
    long pos = current_pos_ - 10;
    if (pos >= 0)
    {
        this->MediaPlayer1->Pause();
        this->MediaPlayer1->Position = (pos>0?pos*1000.0L:0);
        if (!PlayButton->Enabled) this->MediaPlayer1->Play(); //mp3_.play();
        current_pos_ = pos;
        float scale = static_cast<float>(current_pos_) / track_length_;
        ProgressPanelBar->Width = ProgressPanel->Width * scale;
    }
}
//-----

void __fastcall TForm1::FFButtonClick(TObject *Sender)
{
    long pos = current_pos_ + 10;
    if (pos < track_length_)
    {
        this->MediaPlayer1->Pause();
        this->MediaPlayer1->Position = (pos<this->MediaPlayer1->Length
?pos*1000.0L:this->MediaPlayer1->Length);
        if (!PlayButton->Enabled) this->MediaPlayer1->Play();

        current_pos_ = pos;
        float scale = static_cast<float>(current_pos_) / track_length_;
        ProgressPanelBar->Width = ProgressPanel->Width * scale;
    }
}
//-----

// ***** END MCI-Related Code *****

void __fastcall TForm1::CloseButtonClick(TObject *Sender)
{
    Close();
}
//-----

void __fastcall TForm1::MinimizeButtonClick(TObject *Sender)
{
    Application->Minimize();
}
//-----

void __fastcall TForm1::CaptionPanelMouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
    ReleaseCapture();
    SNDMSG(Handle, WM_NCLBUTTONDOWN, HTCAPTION, 0);
}
//-----

```

```

void __fastcall TForm1::CaptionPanelDbClick(TObject *Sender)
{
    if (Height == CaptionPanel->Height)
    {
        CaptionLabel->Caption = fixed_caption;
        Height = fixed_height;

        RECT RWork;
        if (SystemParametersInfo(SPI_GETWORKAREA, 0, &RWork, 0))
        {
            while (Top + Height > RWork.bottom)
            {
                Top = Top - 3;
                Application->ProcessMessages();
            }
        }
    }
    else
    {
        while (Height > CaptionPanel->Height - 5)
        {
            Height = Height - 10;
            Application->ProcessMessages();
        }
        Height = CaptionPanel->Height;

        RECT RWork;
        if (SystemParametersInfo(SPI_GETWORKAREA, 0, &RWork, 0))
        {
            if (abs(Top - RWork.bottom + fixed_height) < Height)
            {
                while (Top < RWork.bottom - Height)
                {
                    Top = Top + 2;
                    Application->ProcessMessages();
                }
            }
        }
    }
}
//-----

void __fastcall TForm1::WMEnterSizeMove(TMessage& AMsg)
{
    DWORD dwPos = GetMessagePos();
    PSnapped_ = Point(LOWORD(dwPos), HIWORD(dwPos));

    SystemParametersInfo(SPI_GETWORKAREA, 0, &RWork_, 0);
}
//-----

void __fastcall TForm1::WMMoving(TMessage& AMsg)

```

```

{
    DWORD dwPos = GetMessagePos();
    POINT PTrack = { LOWORD(dwPos), HIWORD(dwPos) };
    LPRECT lpRect = reinterpret_cast<LPRECT>(AMsg.LParam);

    static bool IsSnappedX = false;
    static bool IsSnappedY = false;

    const int snap_zone = 12;
    if (abs(lpRect->left - RWork_.left) < snap_zone)
    {
        if (!IsSnappedX) PSnapped_ = PTrack;
        IsSnappedX = true;
        lpRect->left = RWork_.left;
        lpRect->right = lpRect->left + Width;
    }
    if (abs(lpRect->top - RWork_.top) < snap_zone)
    {
        if (!IsSnappedY) PSnapped_ = PTrack;
        IsSnappedY = true;
        lpRect->top = RWork_.top;
        lpRect->bottom = lpRect->top + Height;
    }
    if (abs(lpRect->right - RWork_.right) < snap_zone)
    {
        if (!IsSnappedX) PSnapped_ = PTrack;
        IsSnappedX = true;
        lpRect->right = RWork_.right;
        lpRect->left = lpRect->right - Width;
    }
    if (abs(lpRect->bottom - RWork_.bottom) < snap_zone)
    {
        if (!IsSnappedY) PSnapped_ = PTrack;
        IsSnappedY = true;
        lpRect->bottom = RWork_.bottom;
        lpRect->top = lpRect->bottom - Height;
    }

    if (IsSnappedX)
    {
        if (abs(PSnapped_.x - PTrack.x) > snap_zone)
        {
            IsSnappedX = false;
            lpRect->left = lpRect->left + (PTrack.x - PSnapped_.x);
            lpRect->right = lpRect->left + Width;
        }
    }
    if (IsSnappedY)
    {
        if (abs(PSnapped_.y - PTrack.y) > snap_zone)
        {
            IsSnappedY = false;
            lpRect->top = lpRect->top + (PTrack.y - PSnapped_.y);

```

```

        lpRect->bottom = lpRect->top + Height;
    }
}

TForm::Dispatch(&AMsg);
}
//-----

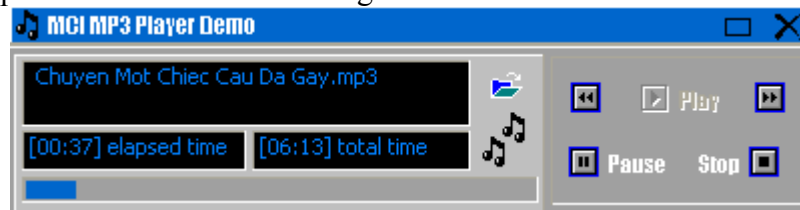
void __fastcall TForm1::SysIconImageClick(TObject *Sender)
{
    PopupMenu1->Popup(Mouse->CursorPos.x, Mouse->CursorPos.y);
}
//-----

void __fastcall TForm1::PopupMenu1Popup(TObject *Sender)
{
    Play1->Enabled = PlayButton->Enabled;
    Pause1->Enabled = PauseButton->Enabled;
    Stop1->Enabled = StopButton->Enabled;
}
//-----

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    this->MediaPlayer1->TimeFormat = tfMilliseconds;
}
//-----

```

Bước 3: Nhấn phím F9 để kiểm thử chương trình



Hình 96-Hình kết quả thực thi chương trình

BÀI 6- LIÊN KẾT CƠ SỞ DỮ LIỆU

1. Các kỹ thuật truy cập dữ liệu trong C++ Builder.

Trong nội dung phần này, chúng ta sẽ nghiên cứu sơ lược về các cách kết nối với cơ sở dữ liệu.

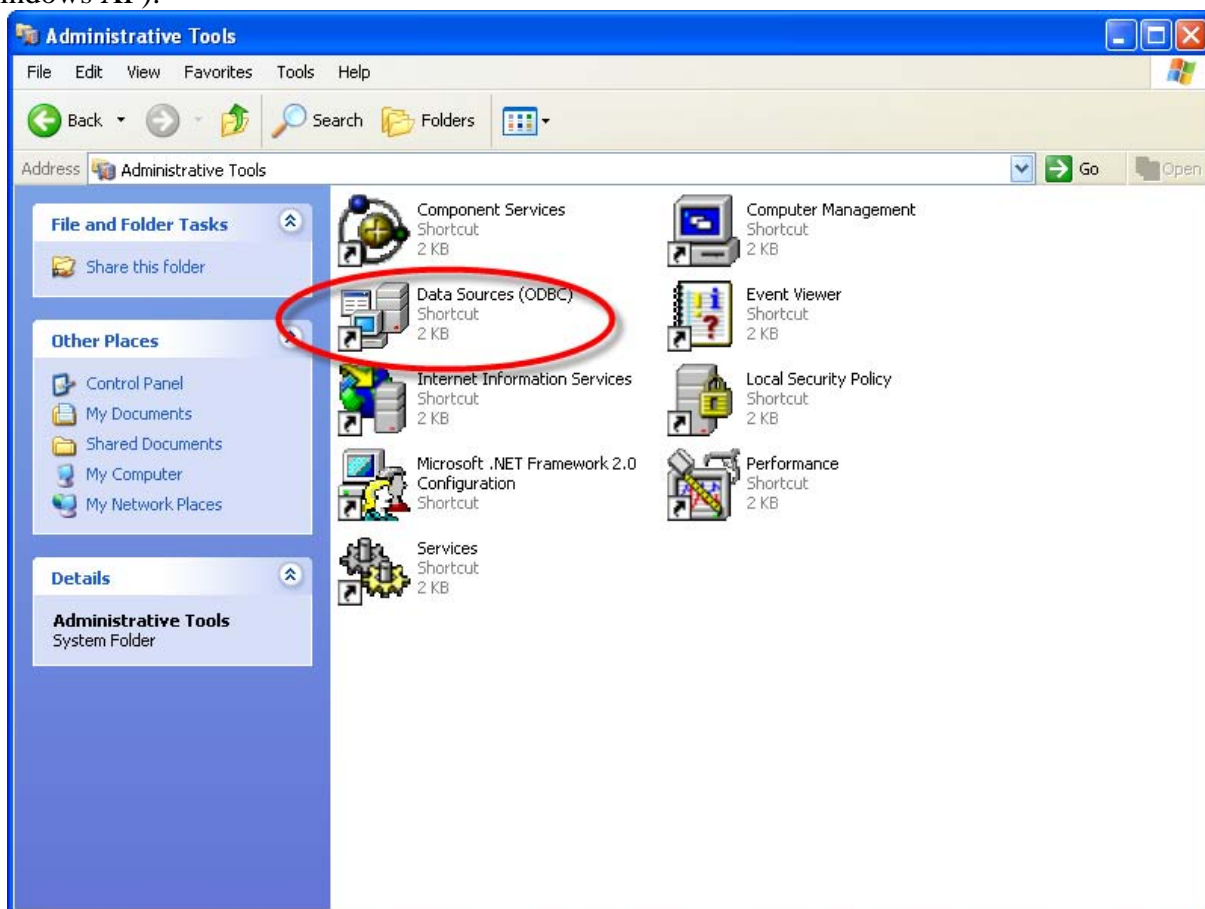
* Truy xuất dữ liệu thông qua ODBC:

Mỗi hệ cơ sở dữ liệu có cách lưu trữ dữ liệu và quản lý dữ liệu riêng. Để đọc được các tập tin hay cơ sở dữ liệu được tổ chức bởi các hệ quản trị dữ liệu này, phương pháp tốt nhất là đọc bởi chính sản phẩm tạo ra tập tin hay cơ sở dữ liệu này. Để sử dụng các ngôn ngữ lập trình truy cập dữ liệu mà không nhờ đến hệ quản trị cơ sở dữ liệu, người lập trình phải nghiên cứu cấu trúc lưu trữ và quản lý của từng hệ quản trị cơ sở dữ liệu, việc làm này hầu như không thể thực hiện nổi.

Giải pháp được đề ra, mỗi nhà phát triển tự phát triển một trình điều khiển với một số hàm thư viện tổng quát có thể đọc và truy cập cơ sở dữ liệu của họ. Nhưng cách này vẫn chưa lại hiệu quả cao, bởi vì, lập trình viên phải nghiên cứu tài liệu của từng trình điều khiển để biết được tên và cách gọi các hàm trong thư viện mà trình điều khiển cung cấp.

Microsoft Windows đã hướng đến giải pháp đơn giản hơn, đó là dùng cầu nối ODBC (Open Database Connectivity). Khi nhà cung cấp hệ quản trị cơ sở dữ liệu muốn lập trình viên có thể truy cập cơ sở dữ liệu của họ, họ phải viết trình điều khiển riêng theo chuẩn ODBC do Microsoft đưa ra. Microsoft và Windows sẽ lo giao tiếp với trình điều khiển. Cách thực hiện này mang lại sự đơn giản cho người lập trình vì người lập trình không cần quan tâm đến trình điều khiển.

Để cài đặt ODBC, bạn chọn Control Panel/Administrative/ODBC Data Sources (cho Windows XP):



Hình 97-Hình minh họa tạo kết nối ODBC

- Truy xuất dữ liệu thông qua OLE:

Phương thức này được xây dựng dựa trên công nghệ COM. Chúng ta không cần dùng công nghệ ODBC nhưng bạn phải sử dụng công nghệ COM. Tuy nhiên, COM gắn với Windows nên gọi đến đối tượng COM có thể xem là cách khả thi trong Windows.

- Truy xuất dữ liệu thông qua ADO:

Nhận thức được tầm quan trọng của COM và cơ sở dữ liệu, Microsoft đã viết ra phần lớn các đối tượng cho phép truy xuất đến hầu như mọi hệ cơ sở dữ liệu lớn nhỏ thông qua COM. Những đối tượng truy xuất dữ liệu này được gọi tắt với tên ADO (Active Data Object). Do ADO là một phần của hệ điều hành nên chúng ta có thể tin tưởng vào tốc độ và sự ổn định của nó.

- Truy xuất dữ liệu thông qua SQL Links:

Borland thay thế vai trò của ODBC bằng SQL Links. SQL Links của Borland đọc các trình điều khiển cơ sở dữ liệu và cho phép truy xuất dữ liệu như ADO, ODBC thông qua BDE (Borland Database Engine).

- Truy xuất không cần trình điều khiển:


C++ Builder cung cấp một loạt các công cụ hỗ trợ để kết nối trực tiếp cơ sở dữ liệu Interbase của Borland. Nhược điểm lớn nhất của cơ sở dữ liệu Interbase là đến thời điểm này vẫn chưa hỗ trợ Unicode.

2. Các thành phần truy cập cơ sở dữ liệu của C++ Builder.

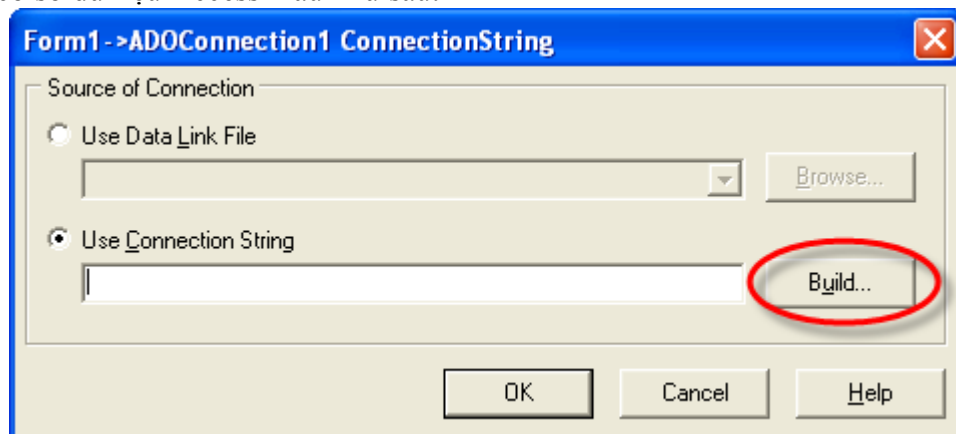
Trước khi đề cập đến phương pháp truy cập dữ liệu, chúng tôi đưa ra 3 cấp để chúng ta nắm rõ phương pháp kết nối dữ liệu:

- Cấp 1: Lựa chọn các trình điều khiển kết nối dữ liệu.
- Cấp 2: Chọn các thành phần truy cập dữ liệu tương ứng.
- Cấp 3: Các thành phần điều khiển và làm việc trực tiếp với cơ sở dữ liệu.

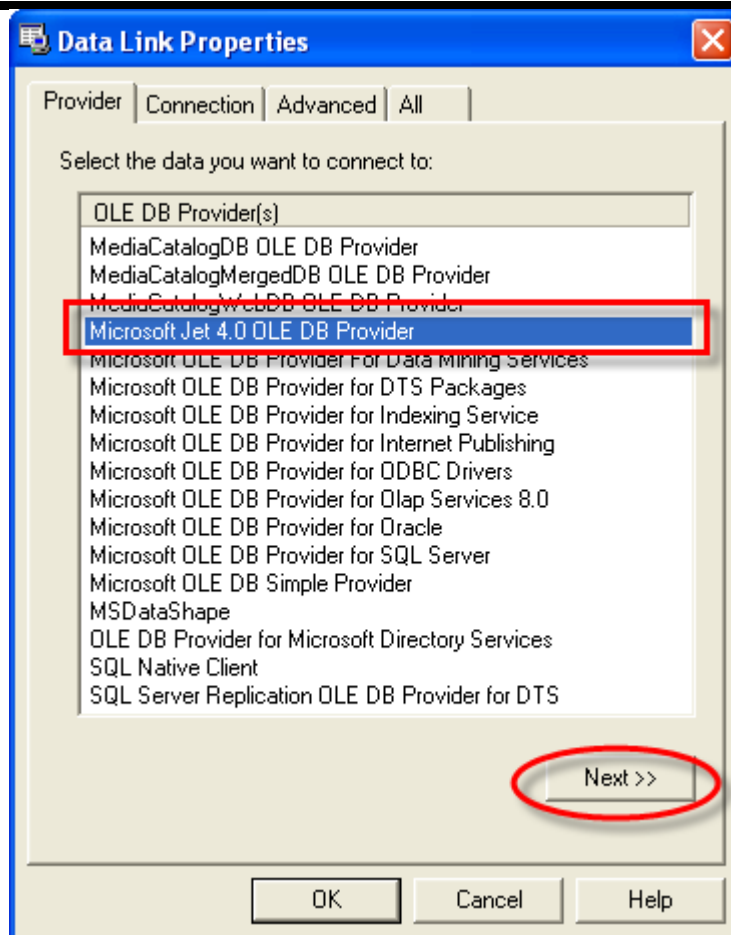
Trong mục này, chúng tôi chỉ trình bày các đối tượng và hướng dẫn từng bước để kết nối dữ liệu ở hai bước đầu theo các bước sau:

- Bước 1: Sử dụng đối tượng ADOConnection  trong ngăn công cụ ADO.

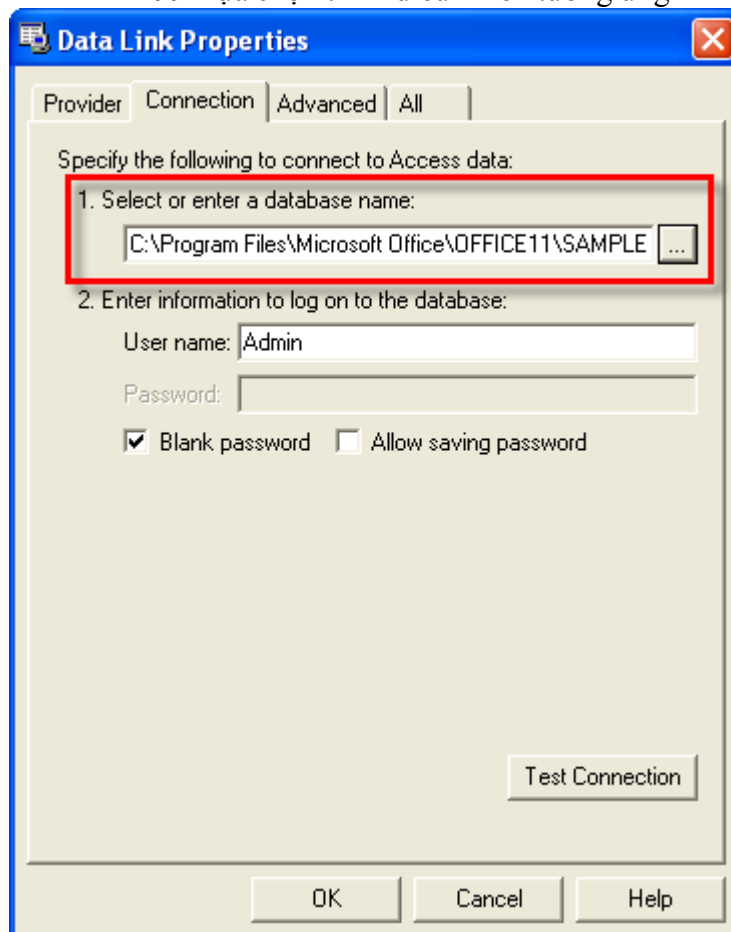
- Bước 2: Sử dụng thuộc tính ConnectionString và kết nối đến cơ sở dữ liệu theo hình minh họa kết nối cơ sở dữ liệu Access mẫu như sau:



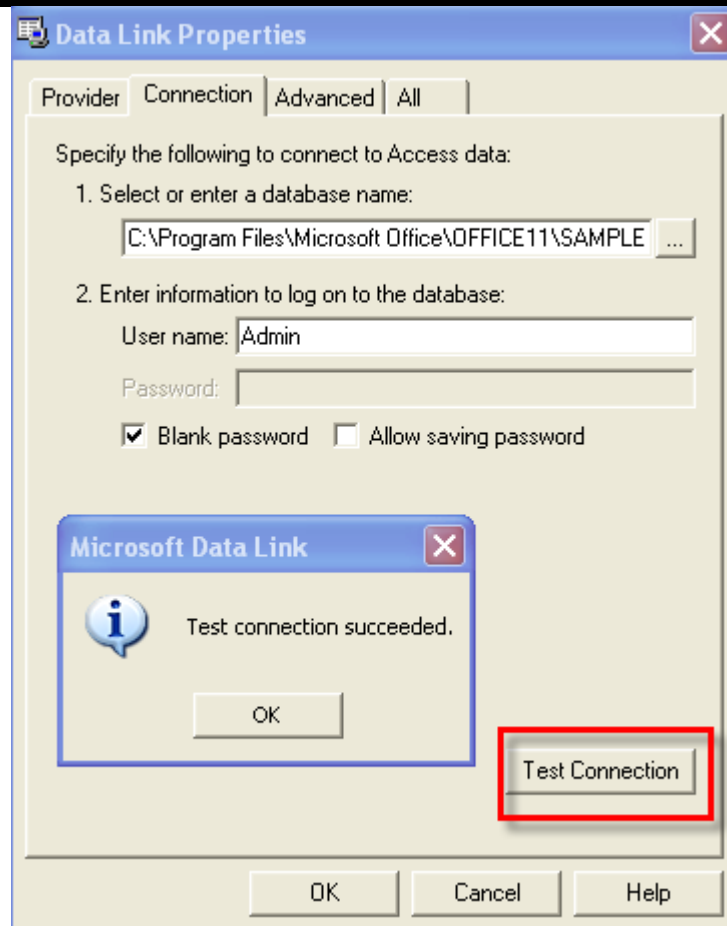
Hình 98-Hộp thoại đầu tiên tạo kết nối



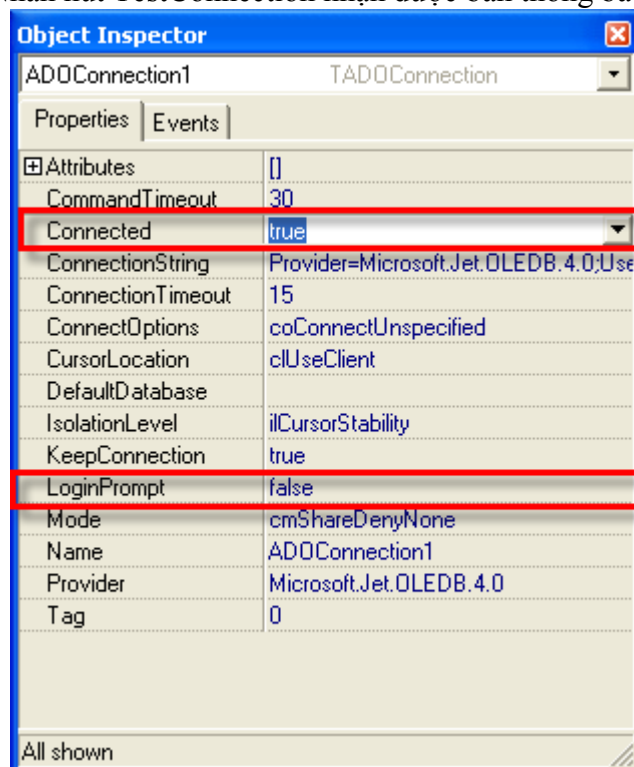
Hình 99-Lựa chọn trình điều khiển tương ứng




Hình 100-Hình minh họa chọn tập tin mẫu Access.

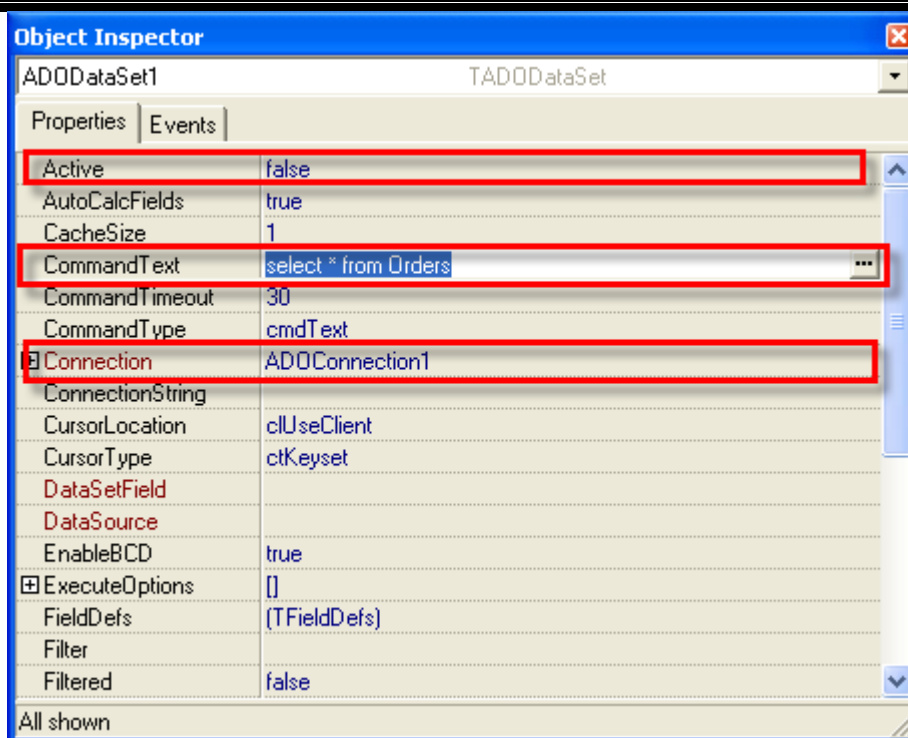


Hình 101-Nhấn nút TestConnection nhận được bản thông báo thành công




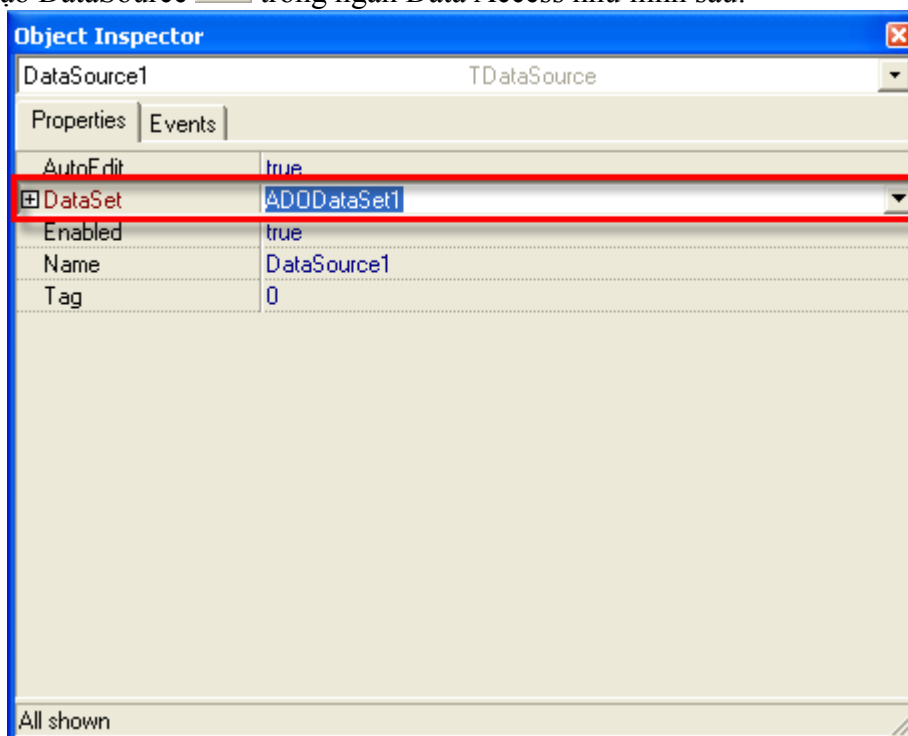
Hình 102-Đổi LoginPrompt = false và Connected=true để mở kết nối cơ sở dữ liệu.

- Bước 3: Sử dụng đối tượng ADODataset () trong ngăn ADO để kết nối dữ liệu theo các hình minh họa sau:



Hình 103-Đổi các thuộc tính Connection, CommandText bằng câu lệnh SQL, Active=true.

-Bước 4: Tạo DataSource  trong ngăn Data Access như hình sau:



Hình 104-Điều chỉnh các thuộc tính cho đối tượng DataSource.

Tất cả các hình trên minh họa việc tạo kết nối và sử dụng các đối tượng hiểu dữ liệu để kết nối dữ liệu. Trong phần tiếp theo, chúng ta sẽ nghiên cứu phương pháp tạo kết nối cho các đối tượng điều khiển trên Form.

3. Các điều khiển nhận biết dữ liệu của C++ Builder

Khi chúng ta xây dựng giao diện người sử dụng cho ứng dụng cơ sở dữ liệu của chúng ta, chúng ta tạo làm sao để tổ chức để hiển thị thông tin và các đối tượng điều khiển thông tin đó.

Một trong những quyết định đầu tiên mà chúng ta phải lựa chọn hoặc là hiển thị một bản ghi đơn hay nhiều bản ghi vào cùng một thời điểm. Thêm vào đó, chúng ta sẽ muốn thêm các điều

hiển thị dữ liệu hướng và quản lý bản ghi. Điều khiển TDBNavigator cung cấp rất nhiều chức năng cho chúng ta thực hiện điều này.

Các thành phần nhận biết dữ liệu như TDBEdit và TDBGrid sẽ tự động hiển thị các giá trị đính kèm với các thành phần trường. Nếu việc soạn thảo được cho phép cho dataset và các điều khiển, các đối tượng nhận biết dữ liệu cũng có thể gọi và thay đổi giá trị của cơ sở dữ liệu. Thông thường, các thuộc tính và phương thức dựng sẵn của đối tượng nhận biết dữ liệu cho phép chúng kết nối đến dataset, hiển thị các giá trị, và cập nhật mà không cần yêu cầu lập trình thêm. Sử dụng chúng khi nào có thể ứng dụng cơ sở dữ liệu của chúng ta.

Các điều khiển chuẩn cũng có thể hiển thị và sửa đổi giá trị cơ sở dữ liệu đính kèm với thành phần trường. Sử dụng các điều khiển chuẩn, tuy nhiên, có thể yêu cầu phải lập trình thêm. Ví dụ, khi sử dụng các điều khiển chuẩn, ứng dụng của chúng ta sẽ không đáp ứng để giám sát khi nào cập nhật dữ liệu khi dữ liệu trong trường thay đổi. Nếu dataset có một thành phần datasource, chúng ta có thể sử dụng sự kiện của chúng để xử lý vấn đề này. Trong đa số trường hợp, sự kiện OnDataChange cho phép chúng ta cập nhật dữ liệu của điều khiển và sự kiện OnStateChange có thể giúp kiểm tra khi nào đối tượng cho phép tương tác hay không cho phép tương tác (enable hay disable).

Ngăn công cụ Data Controls của bảng công cụ cung cấp một tập các điều khiển nhận biết dữ liệu bằng cách sử dụng các trường trong một bảng ghi cơ sở dữ liệu và nếu dataset cho phép, người sử dụng có thể sửa đổi và chuyển chúng ngược vào cơ sở dữ liệu. Bằng cách đặt điều khiển dữ liệu trên form trong ứng dụng cơ sở dữ liệu của chúng ta, chúng ta có thể xây dựng giao diện người sử dụng của ứng dụng cơ sở dữ liệu của chúng ta để có thể hiển thị thông tin tới người sử dụng.

Các điều khiển nhận biết dữ liệu chúng ta thêm vào giao diện của chúng ta phụ thuộc vào các nhân tố khác nhau, bao gồm:

Kiểu của dữ liệu mà chúng ta hiển thị. Chúng ta có thể chọn giữa các điều khiển thiết kế để hiển thị và sửa đổi dữ liệu văn bản thô, các điều khiển làm việc với văn bản định dạng, điều khiển cho đồ họa, các phần tử đa phương tiện, ... Các điều khiển này hiển thị các kiểu khác nhau của thông tin trong một bản ghi.

Chúng ta muốn tổ chức thông tin như thế nào. Chúng ta có thể muốn hiển thị thông tin từ một bản ghi lên màn hình hay danh sách các thông tin từ nhiều bản ghi trên một lưới.

Kiểu của dataset cung cấp dữ liệu cho các điều khiển. Các điều khiển phản xạ sự giới hạn của dataset nằm dưới. Ví dụ, chúng ta không thể sử dụng một lưới với một dataset theo một hướng duy nhất bởi vì các dataset này chỉ có thể cung cấp một bản ghi đơn vào một thời điểm.

Nếu chúng ta muốn người sử dụng dẫn hướng qua các bản ghi của dataset và thêm hay sửa dữ liệu. Chúng ta có thể muốn thêm các điều khiển riêng của chúng ta hay máy móc để dẫn hướng và soạn thảo hay chúng ta có thể muốn sử dụng một điều khiển xây dựng trong như điều khiển dẫn hướng dữ liệu.

Trong rất nhiều ứng dụng, chúng ta có thể chỉ muốn cung cấp thông tin về một bản ghi đơn tại một thời điểm. Ví dụ, một ứng dụng về buôn bán có thể hiển thị thông tin về một bản ghi mua bán mà không cần quan tâm đến các dòng khác. Những thông tin này chắc chắn từ một bản ghi trong dataset các đơn đặt hàng.

Những điều khiển nhận biết dữ liệu trong những giao diện người sử dụng trình bày một trường đơn lẻ từ một bản ghi cơ sở dữ liệu. Ngăn Data Controls của bảng công cụ cung cấp sự lựa chọn rộng lớn về các điều khiển để trình bày các kiểu khác nhau của các trường. Những điều khiển này thường là những phiên bản nhận biết dữ liệu của các điều khiển khác cho phép trên bản công cụ. Ví dụ, điều khiển TDBEdit là phiên bản nhận biết dữ liệu của điều khiển TEdit.

Những điều khiển mà chúng ta sử dụng phụ thuộc trên kiểu dữ liệu của dữ liệu (văn bản, văn bản được định dạng, những đồ họa, thông tin lô gic, ...) chứa trong trường.

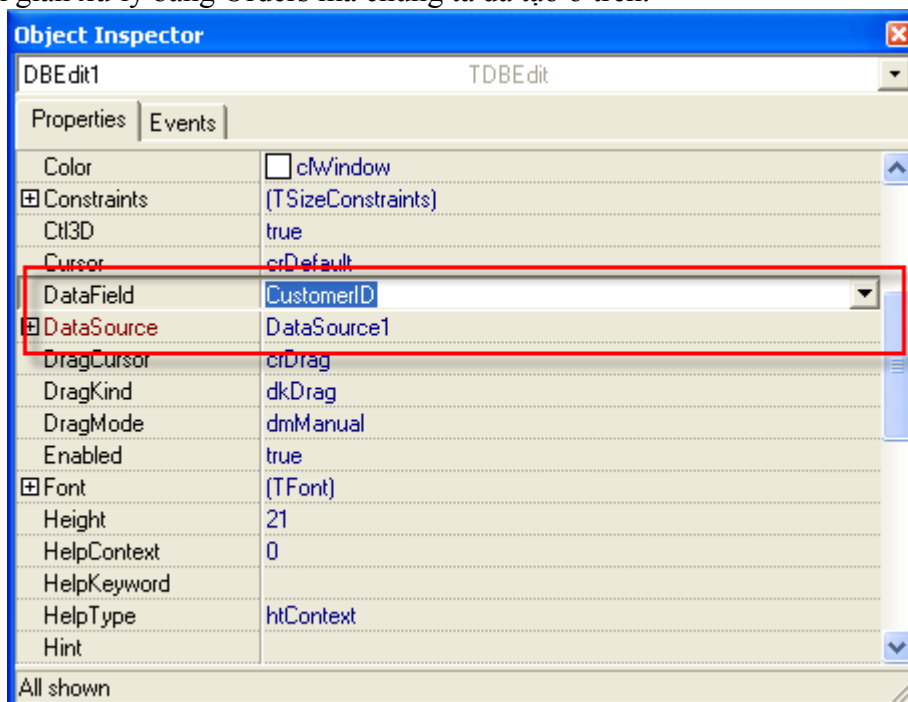
Đôi khi chúng ta muốn hiển thị rất nhiều bản ghi trong một form. Ví dụ, một ứng dụng lập hóa đơn có thể hiển thị tất cả các hóa đơn tạo bởi một người mua hàng trên một form.

Để hiển thị nhiều bản ghi, chúng ta sử dụng một điều khiển lưới. Các điều khiển lưới cung cấp một khung nhìn đa trường, đa bản ghi tạo giao diện người sử dụng hấp dẫn hơn và có hiệu quả hơn.

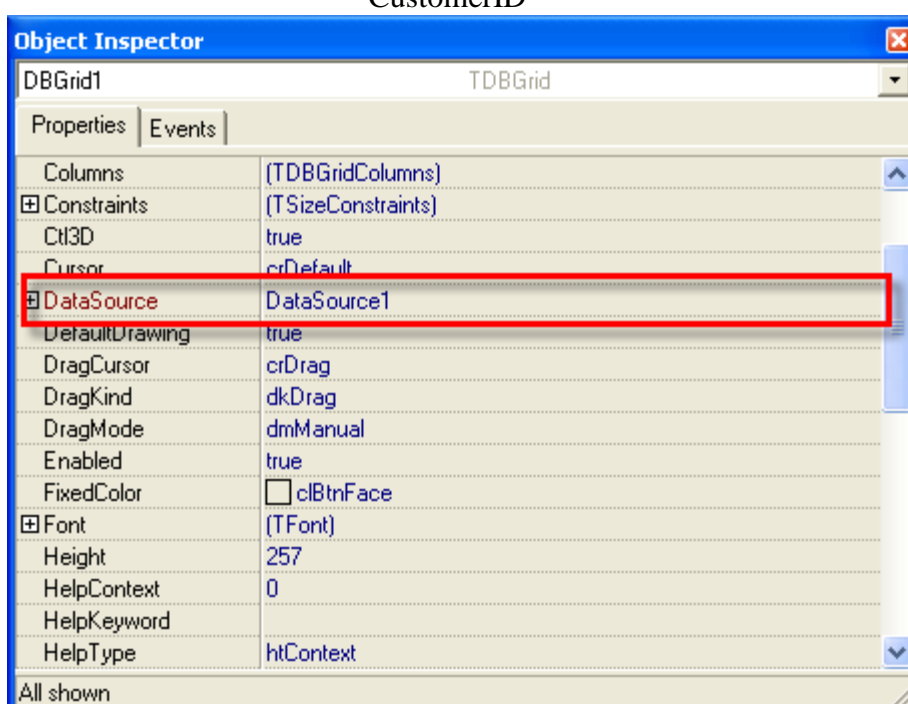
4. Các thành phần cơ sở dữ liệu cơ bản.

Chúng ta có một loạt các đối tượng truy cập dữ liệu trong C++ Builder, các đối tượng này được liệt kê trong ngăn Data Controls. Chức năng của các đối tượng trong ngăn công cụ này tương tự như ngăn công cụ Standard mà chúng tôi đã đề cập ở các bài trên, ngoại trừ hầu hết các đối tượng này được trang bị thêm hai thuộc tính DataSource dùng để liên kết với đối tượng DataSource chứa dữ liệu và thuộc tính DataField để liên kết với trường trong DataSource nhằm thao tác trực tiếp dữ liệu của trường này.

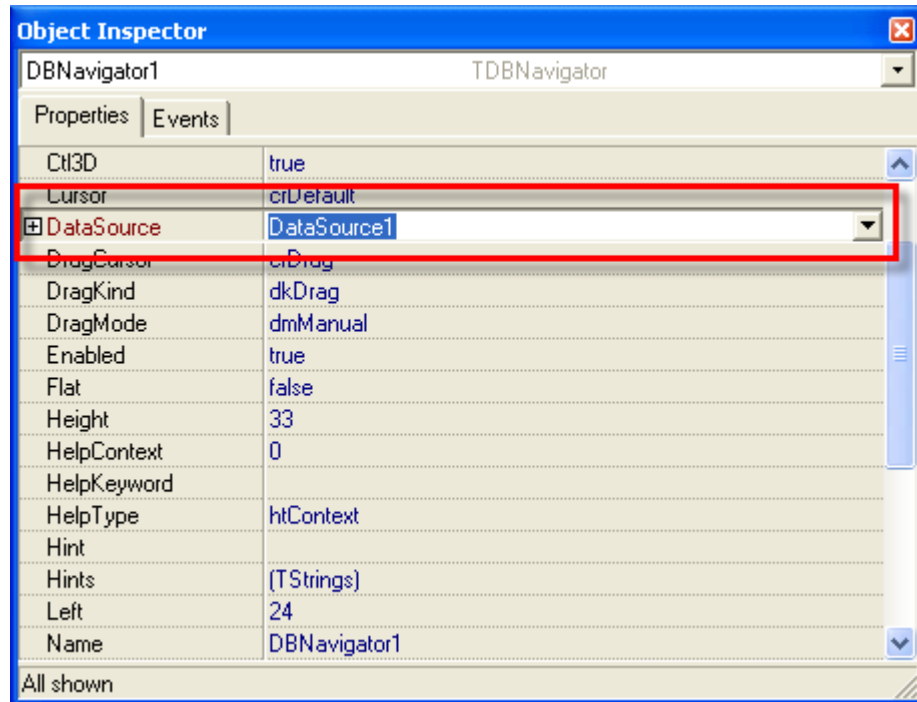
Chúng tôi sẽ đề cập sơ lược cách kết nối dữ và xử lý dữ liệu thông qua các hình minh họa tạo một Form đơn giản xử lý bảng Orders mà chúng ta đã tạo ở trên.



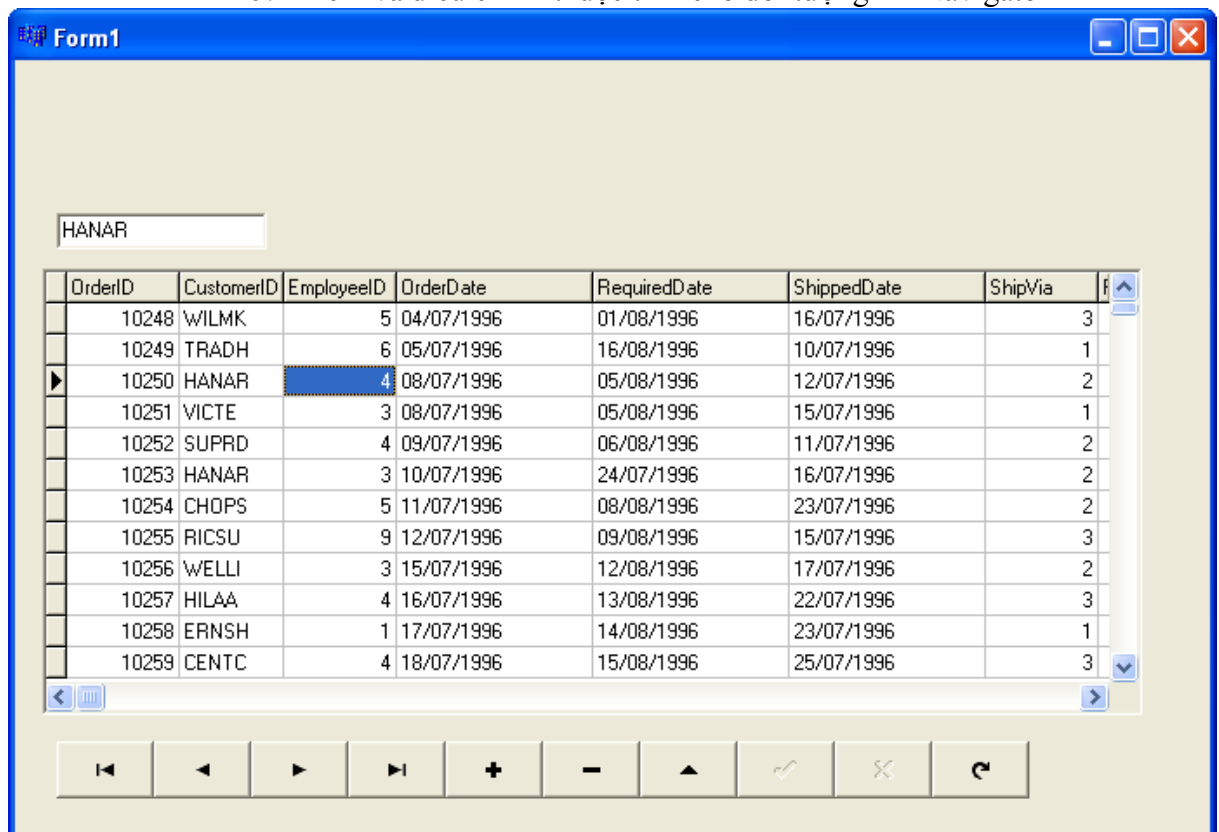
Hình 105-Thêm và điều chỉnh thuộc tính cho đối tượng DBEdit1 nhằm thao tác với trường CustomerID



Hình 106-Thêm và điều chỉnh DBGrid1



Hình 107-Thêm và điều chỉnh thuộc tính cho đối tượng DBNavigator1



Hình 108-Kết quả chương trình thực thi

Một thuộc tính khá quan trọng của các đối tượng cơ sở dữ liệu là `ReadOnly`. Nếu thuộc tính này được đặt bằng `false` thì đối tượng sẽ chỉ còn chức năng hiển thị dữ liệu mà không cho người dùng thay đổi dữ liệu.

Chú ý rằng, để thực hiện thao tác các câu lệnh SQL Insert, Delete, Update, chúng ta phải sử dụng đối tượng `ADOQuery`.

Ví dụ, để thực thi một câu lệnh xóa những bản ghi có `EmployeeID` bằng 4, chúng ta kết nối giống minh họa ở trên, ngoại trừ thuộc tính `Active` chúng ta để bằng `false`. Sau đó thực hiện đoạn lệnh sau:

```
this->ADOQuery1->SQL->Clear() ;
this->ADOQuery1->SQL->Add("delete from Orders where EmployeeID=4;");
this->ADOQuery1->ExecSQL();
```

Chúng ta có thể thực hiện nhiều câu lệnh SQL cùng lúc bằng cách ngăn cách chúng bởi dấu chấm phẩy (;).

5. Xây dựng chương trình quản lý danh bạ

Chương trình quản lý danh bạ dựa trên các thông tin cá nhân: họ và tên, số điện thoại nhà, điện thoại cơ quan, điện thoại di động, email, địa chỉ, hình ảnh. Giao diện chính của chương trình gồm 2 ngăn:

Hình 109-Tab chi tiết

Họ và tên	Điện thoại nhà	Điện thoại cơ quan	Điện thoại di động	Địa chỉ
Nguyễn Văn A	0633540108	0633540106	0983866866	01 Hoàng Văn Thụ
Nguyễn Văn B	0633580106	0633560484	0986866668	01 Hoàng Văn Thụ

Hình 110-Tab danh sách

Hình 111-Tab tìm kiếm nâng cao

Chúng ta tiến hành thiết kế chương trình như sau:

Bước 1: Thiết kế cơ sở dữ liệu DANHBA.MDB gồm có bảng DANHSACH chứa các thông tin sau:

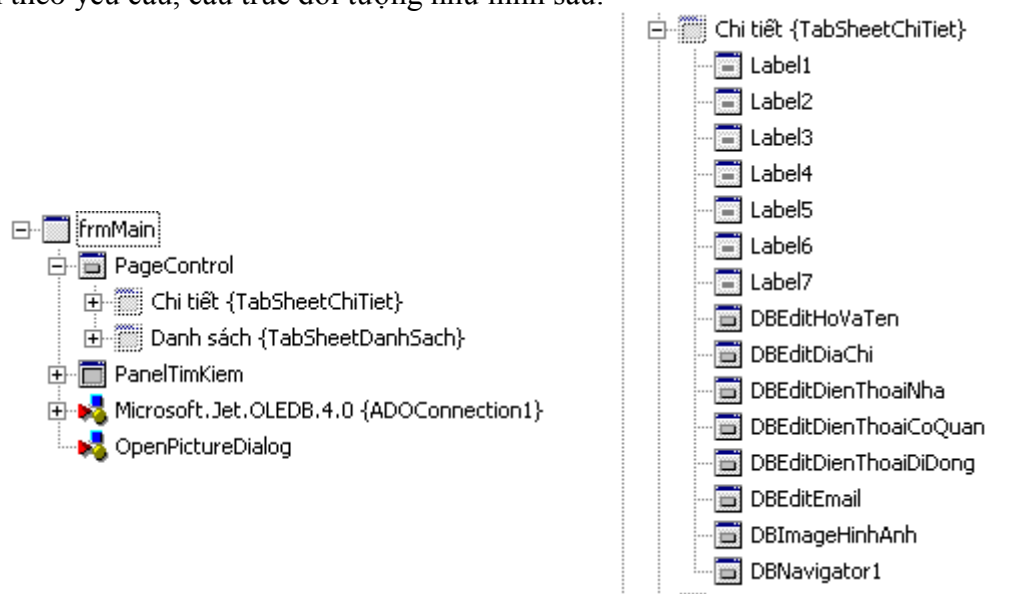
Tên trường	Kiểu dữ liệu
#ID	Autonumber
HOVATEN	Text(70)
DIACHI	Text(79)
DIENTHOAINHA	Text(11)
DIENTHOAICOQUAN	Text(11)
DIENTHOAIDIDONG	Text(11)
Email	Text(50)
HINHANH	OLE Oject

Bước 2: Thiết kế hình NoPic.bmp như hình sau:

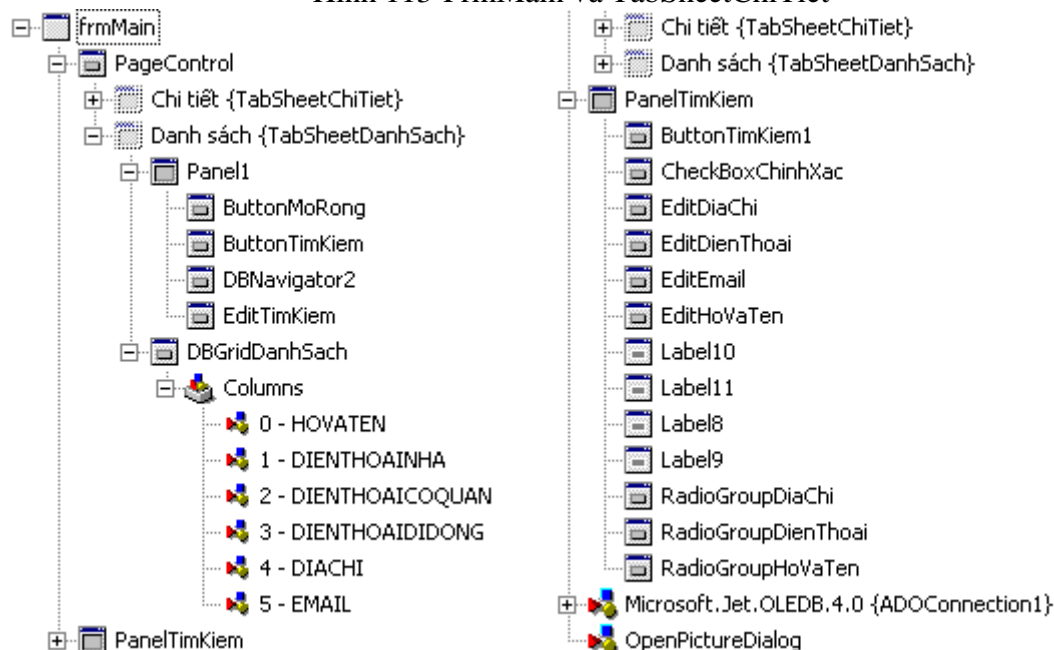


Hình 112- Hình NoPic.Bmp

Bước 3: Tạo đối tượng PageControl gồm có 2 Tab, TabSheetChiTiet và TabSheetDanhSach, sau đó đặt các đối tượng trong ngăn Data Controls, Data Access, dbGo, Standard để đạt được giao diện theo yêu cầu, cấu trúc đối tượng như hình sau:



Hình 113-FrmMain và TabSheetChiTiet



Hình 114-TabSheetDanhSach và Panel tìm kiếm

Bước 4: Viết mã lệnh:

- Sự kiện FormCreate để tự động điều chỉnh kết nối cơ sở dữ liệu, lấy đường dẫn chương trình (Khi biên dịch chạy phần mềm thường sẽ nằm trong thư mục Debug hay Release, chúng ta cần phải copy tập tin cơ sở dữ liệu, tập tin NoPic.bmp vào thư mục này):

```
duongdan = ExtractFilePath(Application->ExeName);
UnicodeString tam;
tam = "Provider=Microsoft.Jet.OLEDB.4.0;";
tam += "User ID=Admin;";
tam += "Data Source=" + duongdan + "\\DANHBA.mdb;";
tam += "Mode=Share Deny None;";
tam += "Extended Properties=\\\";";
tam += "Jet OLEDB:System database=\\\";";
tam += "Jet OLEDB:Registry Path=\\\";";
```

```

tam += "Jet OLEDB:Database Password=\\\\";";
tam += "Jet OLEDB:Engine Type=5;";
tam += "Jet OLEDB:Database Locking Mode=1;";
tam += "Jet OLEDB:Global Partial Bulk Ops=2;";
tam += "Jet OLEDB:Global Bulk Transactions=1;";
tam += "Jet OLEDB:New Database Password=\\\\";";
tam += "Jet OLEDB:Create System Database=False;";
tam += "Jet OLEDB:Encrypt Database=False;";
tam += "Jet OLEDB:Don't Copy Locale on Compact=False;";
tam += "Jet OLEDB:Compact Without Replica Repair=False;";
tam += "Jet OLEDB:SFP=False;";
this->ADOConnection1->Connected = false;
this->ADOConnection1->ConnectionString = tam;
this->DataSetDanhSach->Active = true;

```

- Sự kiện OnClick trên DBNavigator1 để khi thêm mới chúng ta sẽ tải hình tự động vào ô hình ảnh, hình tự động NoPic.bmp được thiết kế như sau:

```

if (Button == nbInsert) {
    this->DBImageHinhAnh->Picture->LoadFromFile(duongdan + "\\NoPic.bmp");
}

```

- Bỏ thuộc tính Delete Confirm trên DBNavigator1 để viết mã lệnh cho sự kiện OnBeforeAction nhằm thực hiện hiển thị hộp thoại hỏi người dùng khi xóa bản ghi:

```

if (Button == nbDelete) {
    if (Application->MessageBoxA(L"Bạn muốn xóa bản ghi này?", L"Confirm",
        MB_YESNO | MB_ICONQUESTION) == IDNO) {
        Abort();
    }
}

```

- Viết mã lệnh OnDbClick trên hình ảnh để hiển thị hộp thoại chọn hình ảnh và tải vào ô hình ảnh:

```

if (this->OpenPictureDialog->Execute(this->Handle)) {
    this->DBImageHinhAnh->Picture->LoadFromFile
        (this->OpenPictureDialog->FileName);
}

```

- Viết mã lệnh cho nút nhấn tìm kiếm để tìm kiếm theo ngữ cảnh, nghĩa là khi ở cột nào sẽ tìm kiếm theo cột đó:

```

UnicodeString st = this->EditTimKiem->Text.Trim();
if (st == "") {
    this->DataSetDanhSach->Filtered = false;
}
else {
    this->DataSetDanhSach->Filtered = false;
    TField *t = this->DBGridDanhSach->SelectedField;
    if (t == NULL) {
        this->DataSetDanhSach->Filter = "HOVATEN Like %" + st + "%";
    }
    else {
        this->DataSetDanhSach->Filter = t->FieldName + " like %" + st + "%";
    }
    this->DataSetDanhSach->Filtered = true;
}

```

- Xây dựng hàm DieuChinhKichThuoc(void) và khai báo thêm biến bool TimKiemNangCao để thực hiện điều chỉnh kích thước biểu mẫu và hiển thị panel tìm kiếm:

+ Khai báo trong ngăn private của tập tin .h như sau:

```

        UnicodeString duongdan; //Đường dẫn chương trình
        bool TimKiemNangCao;
        void DieuChinhKichThuoc();
+ Thêm dòng lệnh sau vào sự kiện FormCreate:
        this->TimKiemNangCao = false;
+ Viết mã lệnh cho phương thức DieuChinhKichThuoc như sau:
void TfrmMain::DieuChinhKichThuoc() {
    int PanelHeight = 166;
    int FormHeight = 457;
    if (this->PageControl->ActivePage == this->TabSheetChiTiet) {
        //Ấn thanh tìm kiếm nâng cao trong mọi tình huống
        this->PanelTimKiem->Visible = false;
        if (this->Height >= FormHeight) {
            for (int i = PanelHeight; i > 0; i -= 2) {
                this->Height -= 2;
                Application->ProcessMessages();
                Sleep(5);
            }
        }
    }
    else {
        if (!this->TimKiemNangCao) {
            //Nếu không tìm kiếm nâng cao thì điều chỉnh kích thước Form
            if (this->Height >= FormHeight) {
                for (int i = PanelHeight; i > 0; i -= 2) {
                    this->Height -= 2;
                    Application->ProcessMessages();
                    Sleep(5);
                }
            }
        }
        else {
            //Nếu tìm kiếm nâng cao thì mở rộng form
            this->PanelTimKiem->Visible = true;
            while (this->Height < FormHeight) {
                this->Height += 2;
                Application->ProcessMessages();
                Sleep(5);
            }
        }
    }
}

- Viết mã lệnh cho sự kiện OnActive của form và OnChage của PageControl để tiến hành
điều chỉnh các kích thước:
void __fastcall TfrmMain::FormActivate(TObject *Sender) {
    this->DieuChinhKichThuoc();
}
void __fastcall TfrmMain::PageControlChange(TObject *Sender) {
    this->DieuChinhKichThuoc();
}

- Viết mã lệnh cho nút nhấn mở rộng để tiến hành hiển thị hay đóng Panel tìm kiếm nâng
cao:
if (this->ButtonMoRong->Caption == "&>>") {

```

```

        this->ButtonMoRong->Caption = "&<<";
        this->TimKiemNangCao = true;
        this->EditTimKiem->Enabled = false;
        this->PanelTimKiem->Visible = true;
    }
    else {
        this->ButtonMoRong->Caption = "&>>";
        this->TimKiemNangCao = false;
        this->EditTimKiem->Enabled = true;
        this->PanelTimKiem->Visible = false;
    }
    this->DieuChinhKichThuoc();
- Viết mã lệnh cho sự kiện OnClick của nút tìm kiếm trên Panel tìm kiếm nâng cao:
    UnicodeString HoVaTen, SoDienThoai, DiaChi, Email;
    UnicodeString BoLoc;
    bool ChinhXac = this->CheckBoxChinhXac->Checked;
    HoVaTen = this->EditHoVaTen->Text.Trim();
    SoDienThoai = this->EditDienThoai->Text.Trim();
    DiaChi = this->EditDiaChi->Text.Trim();
    Email = this->EditEmail->Text.Trim();
    if (HoVaTen != "") {
        if (ChinhXac) {
            HoVaTen = "(HOVATEN = " + HoVaTen + ")";
        }
        else {
            HoVaTen = "(HOVATEN Like " + HoVaTen + "%)";
        }
    }
    BoLoc = HoVaTen;
    if (SoDienThoai != "") {
        if (ChinhXac) {
            SoDienThoai = "(DIENTHOAINHA = " + SoDienThoai +
                ") OR (DIENTHOAICOQUAN = " + SoDienThoai +
                ") OR (DIENTHOAIDIDONG = " + SoDienThoai + ")";
        }
        else {
            SoDienThoai = "(DIENTHOAINHA Like " + SoDienThoai +
                "%) OR (DIENTHOAICOQUAN Like " + SoDienThoai +
                "%) OR (DIENTHOAIDIDONG Like " + SoDienThoai + "%)";
        }
    }
    if (BoLoc != "") {
        BoLoc += " OR (" + SoDienThoai + ")";
    }
    else {
        BoLoc = SoDienThoai;
    }
}
    if (DiaChi != "") {
        if (ChinhXac) {
            DiaChi = "(DIACHI = " + DiaChi + ")";
        }
        else {

```

```
        DiaChi = "(DIACHI Like %" + DiaChi + "%)";
    }
    if (BoLoc != "") {
        BoLoc += " OR (" + DiaChi + ")";
    }
    else {
        BoLoc = DiaChi;
    }
}
if (Email != "") {
    if (ChinhXac) {
        Email = "(EMAIL = " + Email + ")";
    }
    else {
        Email = "(EMAIL Like %" + Email + "%)";
    }
    if (BoLoc != "") {
        BoLoc += " OR (" + Email + ")";
    }
    else {
        BoLoc = DiaChi;
    }
}
this->DataSetDanhSach->Filtered = false;
if (BoLoc != "") {
    this->DataSetDanhSach->Filter = BoLoc;
    this->DataSetDanhSach->Filtered = true;
}
```

BÀI 7 – LẬP TRÌNH MẠNG

1. Các tính năng đặc biệt trong C++ Builder

Trong nội dung mục này, chúng tôi chỉ giới thiệu sơ lược về một số tính năng đặc biệt khác trong C++ Builder.

- C++ Builder hỗ trợ đầy đủ các hàm API, do đó, người lập trình có thể sử dụng chức năng tùy biến của hệ điều hành hay tận dụng được sức mạnh của API để mang lại tính năng độc đáo cho chương trình ở cấp độ hệ điều hành.

- C++ Builder cung cấp một bộ công cụ lập trình song song nền tảng, nghĩa là nếu chúng ta lập trình trên Windows với bộ công cụ VCL thì sẽ có một bộ công cụ tương tự hỗ trợ cho hệ thống không Windows là CLX.

- C++ Builder hỗ trợ công cụ cho chúng ta chuyển các ứng dụng từ Visual C++ sang Borland C++ Builder một cách nhanh chóng nhất với sự sửa đổi ít nhất.

- C++ Builder là công cụ gần như không phụ thuộc nền tảng và bộ cài có thể có mã nguồn của VCL. Do đó, mang lại điều kiện thuận lợi để các công ty thứ 3 phát triển các công cụ mở rộng.

Đặc biệt nhất trong C++ Builder là chức năng lập trình mạng với ít mã lệnh nhất mà hiệu quả đạt được là cao nhất. Chúng ta sẽ nghiên cứu thông qua các đề mục dưới đây.

2. Lập trình mạng với Socket

Socket được hiểu như một ổ cắm nối giữa máy chủ và máy con. Mỗi một ổ cắm sẽ quy định một cổng (port) để mở kết nối. Người lập trình khi sử dụng socket để kết nối phải tránh những cổng được quy định thành chuẩn cho các mạng riêng biệt sau đây:

Dịch vụ	Số cổng
Ftp	21
Http	80
Telnet	23
Finger	79
Sntp	25

Mô hình lập trình mạng thông qua Socket sẽ thông qua hai đối tượng sau:

Đối tượng TServerSocket: Dùng phục vụ cho máy chủ. Sau đây là các thuộc tính và sự kiện quan trọng:

Thuộc tính	Ý nghĩa
Active	Đặt server socket vào trạng thái hoạt động hoặc tạm dừng
Port	Số cổng mà server socket phục vụ
Service	Tên của ứng dụng hay dịch (không bắt buộc)

Sự kiện	Ý nghĩa
OnAccept	Chấp nhận kết nối từ máy con hay không
OnClientConnect	Tiếp nhận kết nối từ client
OnClientDisconnect	Ngắt kết nối với client
OnClientRead	Đọc dữ liệu do máy client gửi đến

Đối tượng TClientSocket dùng để phục vụ cho máy con. Sau đây sẽ liệt kê các thuộc tính và sự kiện của TClientSocket.

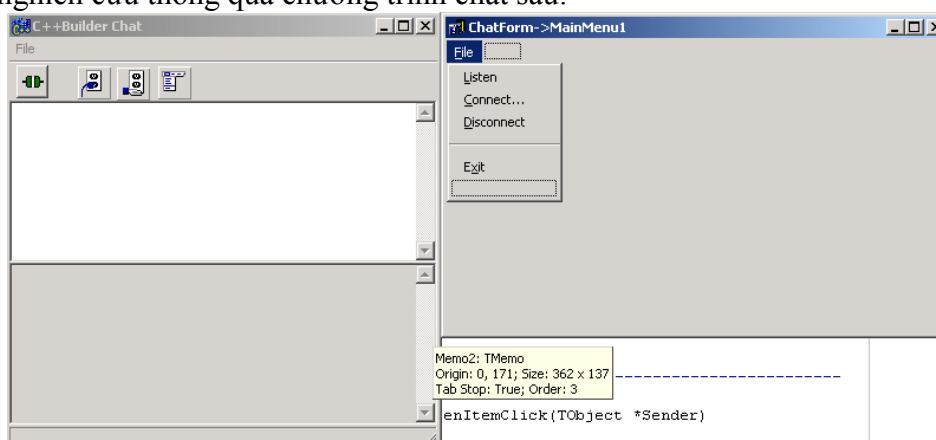
Thuộc tính	Phương thức
Active	Yêu cầu hoặc ngắt kết nối với Server

Address	Địa chỉ máy chủ
Port	Cổng phục vụ của máy chủ
Service	Tương tự máy chủ
Host	Địa chỉ IP

Các sự kiện thông dụng

Sự kiện	Ý nghĩa
OnConnect	Kết nối thành công với máy chủ
OnDisconnect	Ngắt kết nối với máy chủ
OnError	Quá trình kết nối hay chuyển dữ liệu bị lỗi
OnRead	Đọc dữ liệu trả về từ máy chủ

Chúng ta nghiên cứu thông qua chương trình chat sau:



Hình 115-Chương trình chat đơn giản

Mã lệnh được liệt kê kèm theo chú thích như sau:

```
void __fastcall TChatForm::FileListenItemClick(TObject *Sender)
{
    FileListenItem->Checked = !FileListenItem->Checked;
    if (FileListenItem->Checked)
    {
        ClientSocket->Active = false;
        ServerSocket->Active = true;
        StatusBar1->Panels->Items[0]->Text = "Listening...";
    } else
    {
        if (ServerSocket->Active)
        {
            ServerSocket->Active = false;
        }
        StatusBar1->Panels->Items[0]->Text = "";
    }
}
//-----
//-----
void __fastcall TChatForm::FileConnectItemClick(TObject *Sender)
{
    if (ClientSocket->Active)
    {
```



```

    ClientSocket->Active = false;
}
if (InputQuery("Computer to connect to", "Address Name:", Server))
{
    if (Server.Length() > 0)
    {
        ClientSocket->Host = Server;
        ClientSocket->Active = true;
        FileListenItem->Checked = false;
    }
}
}
//-----
//-----
void __fastcall TChatForm::Exit1Click(TObject *Sender)
{
    ServerSocket->Close();
    ClientSocket->Close();
    Close();
}
//-----
//-----
void __fastcall TChatForm::Memo1KeyDown(TObject *Sender, WORD &Key,
    TShiftState Shift)
{
    if (Key == VK_RETURN)
    {
        if (IsServer){ServerSocket->Socket->Connections[0]->SendText(
            Memo1->Lines->Strings[Memo1->Lines->Count - 1]); }
        else
        { ClientSocket->Socket->SendText(Memo1->Lines->Strings[
            Memo1->Lines->Count - 1]);}
    }
}
//-----
//-----
void __fastcall TChatForm::FormCreate(TObject *Sender)
{
    FileListenItemClick(NULL);
}
//-----
//-----
void __fastcall TChatForm::ClientSocketConnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    StatusBar1->Panels->Items[0]->Text = "Connect to: " + Socket->RemoteHost;
}
//-----
//-----
void __fastcall TChatForm::Disconnect1Click(TObject *Sender)
{
    ClientSocket->Active = false;
    ServerSocket->Active = true;
}

```

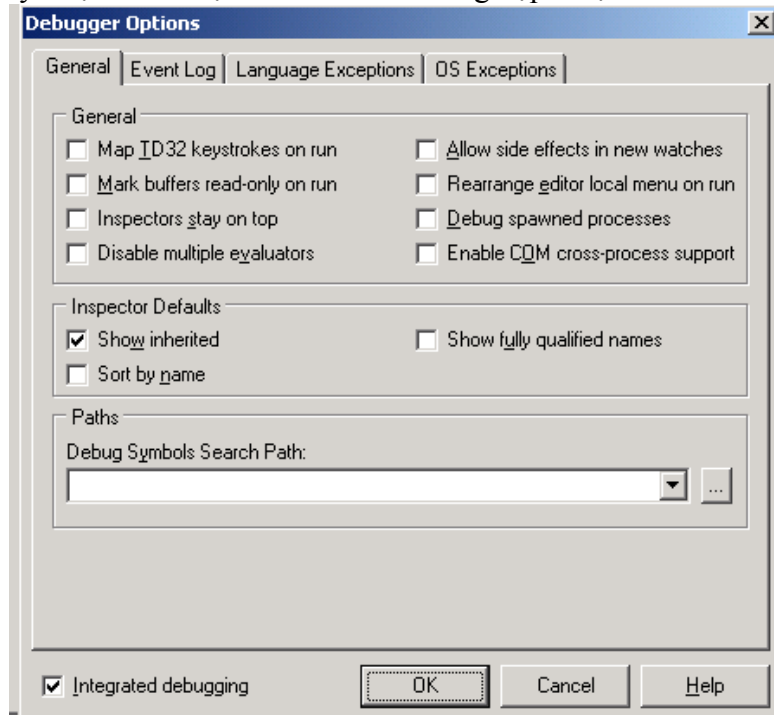
```
StatusBar1->Panels->Items[0]->Text = "Listening...";
}
//-----
//-----
void __fastcall TChatForm::ClientSocketRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    Memo2->Lines->Add(Socket->ReceiveText());
}
//-----
//-----
void __fastcall TChatForm::ServerSocketClientRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    Memo2->Lines->Add(Socket->ReceiveText());
}
//-----
//-----
void __fastcall TChatForm::ServerSocketAccept(TObject *Sender,
    TCustomWinSocket *Socket)
{
    IsServer = true;
    StatusBar1->Panels->Items[0]->Text = "Connect to: " + Socket->RemoteAddress;
}
//-----
//-----
void __fastcall TChatForm::ServerSocketClientConnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    Memo2->Lines->Clear();
}
//-----
//-----
void __fastcall TChatForm::ClientSocketDisconnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    FileListenItemClick(NULL);
}
//-----
//-----
void __fastcall TChatForm::ClientSocketError(TObject *Sender,
    TCustomWinSocket *Socket, TErrorEvent ErrorEvent, int &ErrorCode)
{
    Memo2->Lines->Add("Error connecting to:" + Server);
    ErrorCode = 0;
}
//-----
//-----
void __fastcall TChatForm::ServerSocketClientDisconnect(TObject *Sender,
    TCustomWinSocket *Socket)
{
    StatusBar1->Panels->Items[0]->Text = "Listening...";
}
```

BÀI 8- XÂY DỰNG VÀ TRIỂN KHAI CÁC ỨNG DỤNG

1. Các kỹ thuật gỡ rối trong C++ Builder

Khi lập trình với C++ Builder, chương trình được biên dịch thành tập tin .exe mặc định đã kèm chương trình gỡ rối được biên dịch bên trong. Chương trình này có nhiệm vụ kiểm soát các lỗi phát sinh trong quá trình thực thi chương trình cả lỗi do lập trình và lỗi do hệ điều hành phát sinh.

Các chức năng này được kích hoạt và điều chỉnh trong hộp thoại Tools/Debugger Options...



Hình 116-Hộp thoại tinh chỉnh Debugger Options

Thông thường thì các chức năng được cài đặt mặc định của C++ Builder phù hợp với đa số chương trình mà chúng ta tạo ra.

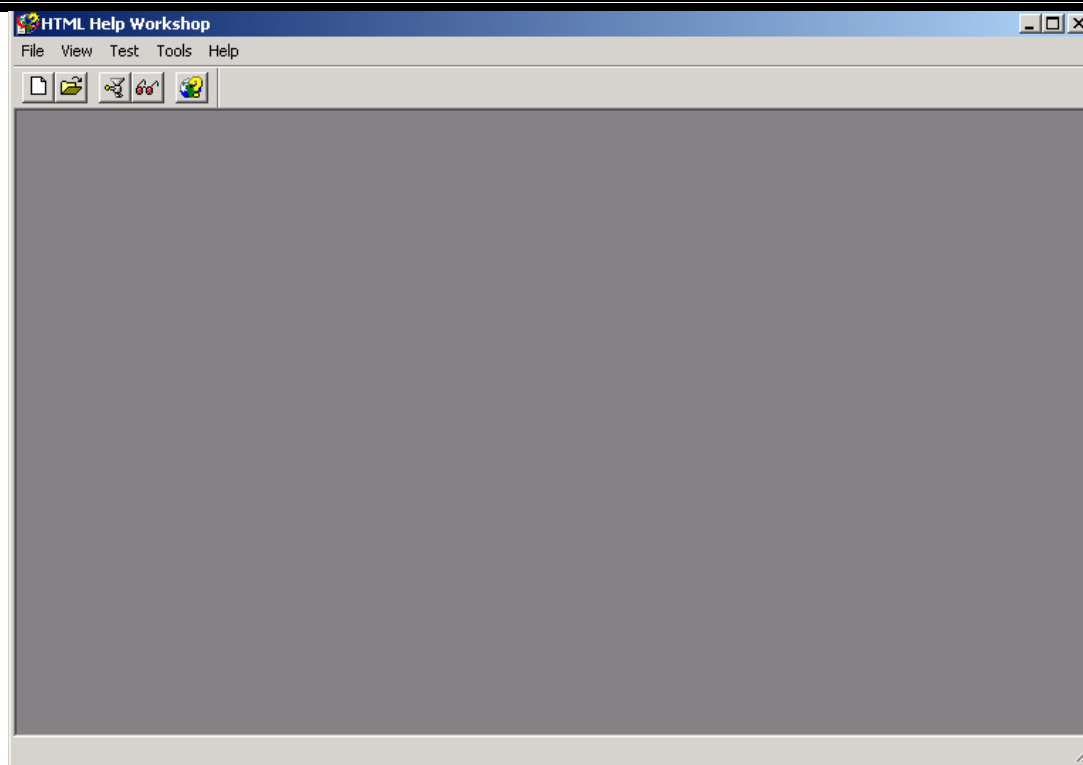
Tuy nhiên, chúng tôi đề nghị để truy bắt lỗi ngoại lệ có thể phát sinh do quá trình lập trình chủ quan. Chúng ta nên dùng chức năng truy bắt ngoại lệ của C++ Builder. C++ Builder cung cấp một khối lệnh try để truy bắt lỗi chắc chắn trong cả hai trường hợp lỗi do chương trình và lỗi do phát sinh.

Chúng ta có thể nghiên cứu thêm về lĩnh vực này trong tài liệu hướng dẫn của C++ Builder.

2. Sử dụng tài nguyên Windows để xây dựng trợ giúp trực tuyến.

Sau khi cài đặt xong C++ Builder, chúng ta sẽ nhận được công cụ miễn phí Html Help Workshop. Công cụ này cho phép chúng ta tạo các tập tin trợ giúp dạng .hlp.

Chúng ta có thể khởi động bằng cách chọn Start/Program Files/HTML Help Workshop/HTML Help Workshop. Khi khởi động xong chúng ta nhận giao diện như sau:



Hình 117-Cửa sổ chương trình HTML Help Workshop

Trước tiên chúng ta cần tạo nội dung cho file Help. Để tạo nội dung các file này chúng ta mở FrontPage express, hoặc FrontPage 98, 2000 và đánh vào nội dung mà chúng ta cần chỉ ra để hướng dẫn người sử dụng. Sau đó chúng ta lưu các file này lên đĩa với các tên như sau: Congty.htm, Diachi.htm, hdCaidat.htm, hdSudung.htm, Sanpham.htm, Ykien.htm, Index.htm (dùng trong ví dụ này).

Một ví dụ về cách tạo các file này dùng FrontPage 2000 (tương tự cho Frontpage 98 và FrontPage express):

Nhấn Start/Programs/Microsoft Frontpage màn hình chính xuất hiện, chúng ta gõ nội dung vào cửa sổ soạn thảo.

Chúng ta chọn Tab Normal ở dưới cùng (ngầm định chọn) và đánh nội dung vào màn hình bỏ trắng. Nếu thích chúng ta có thể dùng tab HTML để tạo nội dung thay cho tab Normal nếu chúng ta đã có kiến thức về HTML. Tab Preview cho chúng ta biết trước nội dung mình vừa nhập sẽ xuất hiện như thế nào.

Sau khi nhập đầy đủ các nội dung thì chúng ta lưu file lên đĩa bằng cách nhấn File/Save, chọn thư mục lưu trữ và đánh tên file vào, chẳng hạn Vidu.htm.

Sau khi đã hoàn thành nội dung cho các file Congty.htm, Diachi.htm, hdCaidat.htm, hdSudung.htm, Sanpham.htm, Ykien.htm chúng ta tiếp tục công việc tạo lập file trợ giúp cho ứng dụng Quản lý Nhân sự và Tiền lương. Đến đây các chúng ta gọi trình ứng dụng tạo lập Help ra (Start/Programs/HTML Help Workshop/HTML Help Workshop) và bắt đầu sắp đặt nội dung, cũng như chỉ mục cho các file nội dung của mình.

Đầu tiên chúng ta cần tạo một file Project (.hhp) là file mà sau này chúng ta sẽ dịch ra file Help.chm của riêng chúng ta, trong cửa sổ của HTML Help Workshop chọn File/New một cửa sổ như sau sẽ xuất hiện.

Chúng ta chọn vào Project (có màu sáng khác biệt) và nhấn OK, một màn hình trợ giúp sẽ xuất hiện.

Chúng ta đánh vào khoảng trống ở Browse C:\My Documents\nstlHelp.hhp, nếu chúng ta muốn tạo ở thư mục khác với thư mục C:\My Document thì chúng ta nhấn chọn Browse để tìm thư mục lưu file Project của mình và đánh vào tên file (ngầm định có phần mở rộng là .hhp), ở đây ta đánh vào nstlHelp.hhp và nhấn Open sẽ thấy xuất hiện tên đường dẫn của thư mục chứa file nstlHelp.hhp đi kèm.

Tiếp tục nhấn Next và màn hình tiếp theo xuất hiện.

Check vào HTML files (.htm) và tiếp tục nhấn Next màn hình tiếp theo sẽ xuất hiện:

Chúng ta nhấn vào nút Add và thêm vào tất cả các file.htm mà chúng ta đã tạo nội dung lúc trước bằng FrontPage 2000:

Tiếp tục nhấn Next và một màn hình tiếp theo sẽ xuất hiện, trong màn hình này chúng ta nhấn chọn Finish để tạo Project của chúng ta, nếu muốn chỉnh sửa thêm, bớt đi các file nội dung .htm hay... thì chúng ta có thể nhấn nút Back để thay đổi lại các mặc định lúc trước rồi cuối cùng ta cũng nhấn Finish để kết thúc việc tạo File Project của mình. Sau khi nhấn chọn Finish kết quả sẽ được là:

Trong tab Project nhấn vào Change Project Options ngay dưới tab Project sẽ thấy xuất hiện một cửa sổ, trong cửa sổ này lựa chọn tab General và đánh nội dung vào khung Title (hướng dẫn sử dụng) nhấn OK, title này sẽ xuất hiện ở góc trái trên cùng của file Help sau khi được dịch hoàn chỉnh.

- Tiếp tục nhấn chọn tab Contents (ngay bên phải tab Project) sẽ thấy xuất hiện cửa sổ sau:

Nhấn chọn Create a new contents file và nhấn OK, chúng ta sẽ thấy xuất hiện một cửa sổ tiếp theo yêu cầu chúng ta chỉ ra thư mục chứa và đánh vào tên cho file.hhc để lưu giữ các file nội dung của chúng ta (ở đây ta đánh tên file là Contents.hhc và lưu trong cùng một thư mục với các file.htm đã tạo).

Trong tab Contents chúng ta nhấn chọn Contents Properties (ngay bên dưới tab Contents) để chọn font chữ thích hợp nếu chúng ta muốn đánh tiếng Việt.

Tiếp tục nhấn vào nút insert a heading (ngay dưới nút Contents Properties và có hình như biểu tượng thư mục):

- Đánh nội dung "Hướng dẫn" vào phần Entry title rồi nhấn OK

Tiếp theo chúng ta nhấn vào nút Insert a page (ngay dưới nút Insert a heading) thấy xuất hiện:

Chúng ta chọn No (để trang chèn vào nằm dưới heading vừa tạo), tiếp theo chúng ta sẽ thấy xuất hiện một cửa sổ giống như lúc chúng ta nhấn vào nút Insert a heading, chúng ta đánh nội dung "hướng dẫn cài đặt" vào phần Entry Title, tiếp theo chúng ta nhấn vào nút Add trong cửa sổ này, một cửa sổ tiếp theo sẽ xuất hiện và chúng ta nhấn vào Nút Browse trong cửa sổ này để mở file hdCaidat.htm ra (file này chúng ta đã tạo nội dung từ bước đầu), cuối cùng chúng ta liên tiếp chọn Open, OK để đóng các cửa sổ này vào (khi này chúng ta đã mở xong được file nội dung hdSudung.htm).

Tương tự, chúng ta nhấn vào Insert a page và nhập nội dung "Hướng dẫn sử dụng" vào Phần Entry title, rồi nhấn Add để liên kết đến file hdSudung.htm.

Chúng ta tiếp tục nhìn vào màn hình chính trong phần Contents thấy phía gần dưới có 4 nút, chúng ta nhấn tuần tự vào các đề mục "Hướng dẫn cài đặt" và "Hướng dẫn sử dụng", sau đó nhấn vào mũi tên sang phải "đ" để các đề mục này lùi vào phía trong một chút (nhằm phân biệt rõ giữa một Heading và một Page - chỉ là hình thức).

Tương tự, chúng ta tiếp tục thực hiện cho Heading "Giới thiệu" gồm 2 đề mục là: "Giới thiệu công ty" liên kết với file Congty.htm và "Giới thiệu sản phẩm" liên kết với Sanpham.htm và hai đề mục này cũng được đẩy lùi vào một chút nhờ nhấn vào nút lệnh có hình mũi tên sang phải.

Cuối cùng chúng ta tạo tương tự cho 2 đề mục đứng độc lập là "Đóng góp ý kiến" liên kết với file Ykien.htm và "Địa chỉ liên lạc" liên kết tới file Diachi.htm, chỉ có điều là đối với 2 đề mục này thì chúng ta để nguyên vị trí chữ không nhấn vào nút lệnh có hình mũi tên sang phải để lùi vào một chút (vì chúng ngang hàng với các Heading kia).

Đến bước này chúng ta đã hoàn thành xong phần Contents, chúng ta tiếp tục làm việc với trang chính Index ngay sát bên phải trang chính Contents (chúng ta hãy cố kiên nhẫn một chút vì thành quả chúng ta sắp đạt được rồi).

Chúng ta nhấn chuột vào tab Index, một cửa sổ hiện ra, chúng ta nhấn vào lựa chọn Create a new index file và nhập vào tên file Index cho hệ thống trợ giúp của mình (ở đây chúng ta nhập vào tên file là Index.hhk trong cùng một thư mục lưu trữ các file nội dung lúc trước):

Để thay đổi font chữ chúng ta nhấn vào nút lệnh Index properties (ở ngay trên cùng trong tab Index) và chọn font chữ thích hợp.

Chúng ta tiếp tục công việc của mình bằng cách nhấn vào nút lệnh Insert a keyword có hình chiếc chìa khoá và ở ngay bên dưới nút lệnh Index properties, chúng ta nhập vào nội dung "About Company" cho phần Keyword, rồi tiếp tục nhấn vào nút Add để tạo liên kết tới file Congty.htm cho keyword này:

Tương tự, chúng ta tiếp tục tạo các keyword: "About Software" liên kết với file Sanpham.htm, "Address" liên kết tới file Diachi.htm, "Feedback" liên kết với file Ykien.htm, "Guide to Setup" liên kết tới file hdCaidat.htm và "Guide to Use" liên kết tới file hdSudung.htm.

Cuối cùng, chúng ta nhấn vào nút lệnh có 2 chữ cái A và Z và một mũi tên kèm theo chúng để sắp xếp các Index này theo thứ tự "a, b, c".

Bước cuối cùng, chúng ta nhấn chọn lại tab Project và nhấn vào nút lệnh Save all project files and compile có hình đĩa mềm ở trên và cái phễu ở dưới để lưu lại tất cả các file Contents, Index... chúng ta đã tạo ở các bước trên và dịch chúng thành một file Help tổng hợp có thể chạy độc lập trong các môi trường khác nhau (ở đây sau khi dịch sẽ tạo ra file Help có tên là nstlHelp.chm chứa trong thư mục C:\My Documents):

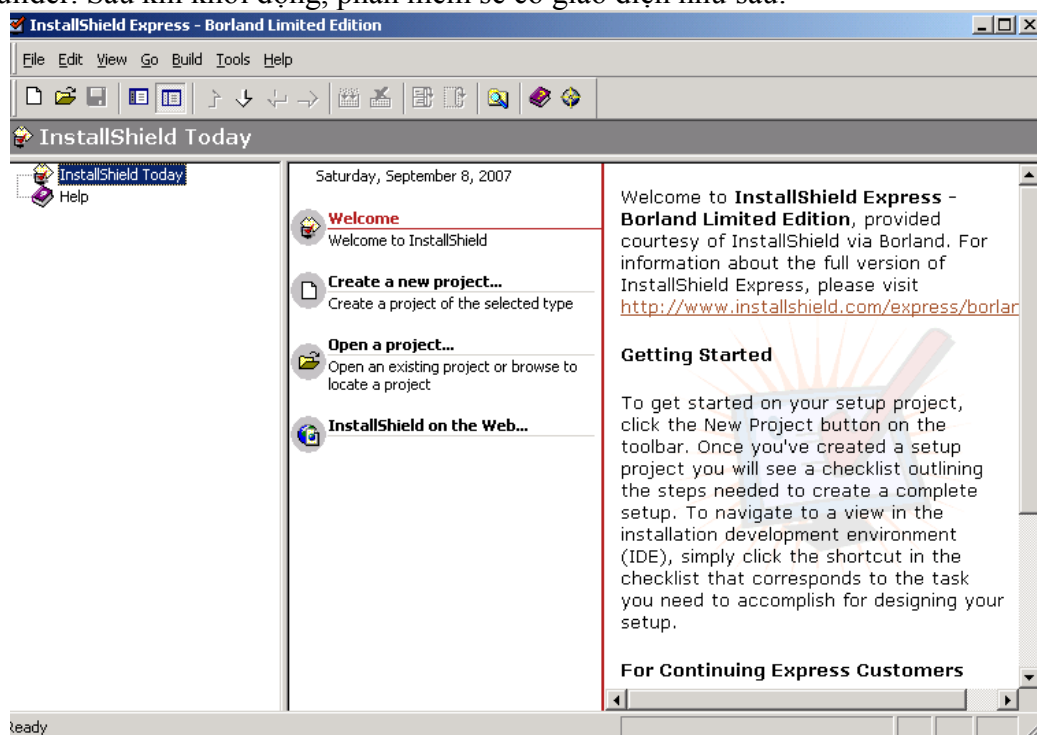
Đến đây chúng ta đã hoàn toàn làm chủ được file Help dạng .chm của mình và đưa vào các ứng dụng khác một cách dễ dàng (tương tự như cách đưa vào các ứng dụng file Help dạng .hlp loại cũ mà trong rất nhiều số báo đã đề cập đến).

Cuối cùng, chúng ta sử dụng chương trình ABC Chm Convert để chuyển tập tin .chm thành tập tin .hlp.

Sau khi chuyển xong thành tập tin .hlp, chúng ta vào cửa sổ Project/Options, chọn ngăn Application và tiến hành liên kết tập tin hướng dẫn .hlp vào ứng dụng thông qua mục chọn Help File.

3. Sử dụng chương trình InstallShield Express.

Phần mềm InstallShield Express (ISXpress) là phần mềm đi kèm theo bộ cài của Borland C++ Builder. Sau khi khởi động, phần mềm sẽ có giao diện như sau:

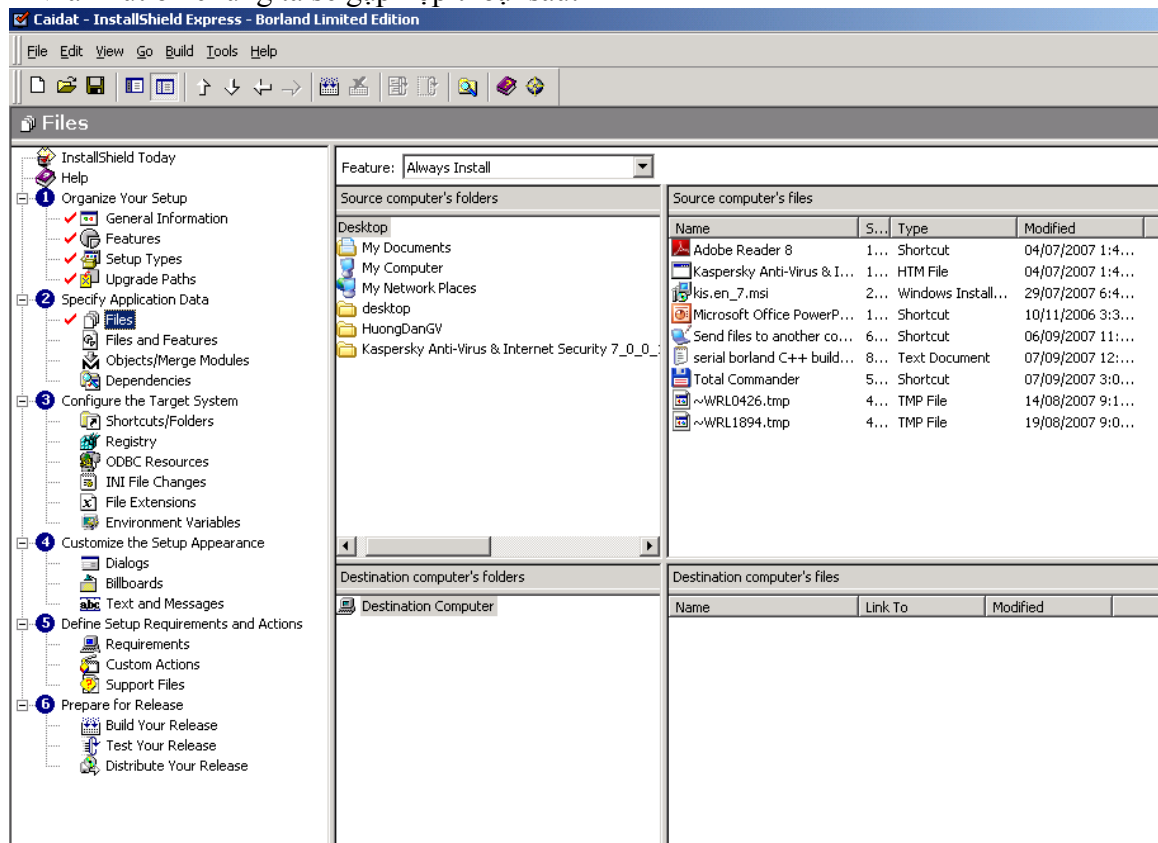


Hình 118-Cửa sổ chương trình InstallShield Express.

Để tạo một bộ cài mới, chúng ta theo các bước sau (bộ cài trong ví dụ là caidat.exe hay caidat.msi)

- Chọn lệnh File/New Project và đặt tên cho project đầy đủ như sau:
C:\Documents and Settings\lt01\My Documents\caidat\Caidat.ism

- Nhấn nút ok chúng ta sẽ gặp hộp thoại sau:



Hình 119-Cửa sổ tinh chỉnh cho cài đặt

Trong cửa sổ làm việc bên khung bên trái chúng ta sử dụng các chức năng sau:

- Nhóm mục chọn 1: Cho phép chúng ta khai báo các thông tin về bộ cài đặt của chúng ta như: thông tin về chương trình, các gói cài đặt tùy chọn, đường dẫn cập nhật.
- Nhóm mục chọn 2: Những chỉ định về dữ liệu của chương trình bao gồm: các tập tin và các thành phần, các mô đun cần thêm vào, cơ sở dữ liệu, ...
- Nhóm mục chọn 3: Cho phép chúng ta khai báo những cài đặt cho máy tính được cài phần mềm của chúng ta bao gồm: lỗi tắt, các khóa registry, các kết nối ODBC, các tập tin INI, các biến môi trường, ...
- Nhóm mục chọn 4: Cho phép chúng ta tinh chỉnh và đưa vào các văn bản, hình ảnh, hộp thoại hiển thị trong quá trình cài đặt chương trình.
- Nhóm mục chọn 5: Chứa đựng những yêu cầu về hệ thống như: RAM, hệ điều hành, các tập tin yêu cầu và khai báo các hành động tùy biến đáp ứng cho từng nhu cầu của hệ thống.
- Nhóm mục chọn 6: Cho phép chúng ta tiến hành đóng gói sản phẩm của chúng ta thành bộ cài .EXE.