

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**  
**TRƯỜNG ĐẠI HỌC LẠC HỒNG**

**GIÁO TRÌNH**  
**LẬP TRÌNH WINDOWS FORMS VỚI C#**

## LỜI NÓI ĐẦU

Giáo trình *Lập trình Windows Forms với C#.NET* sử dụng cho sinh viên ngành Công nghệ thông tin. Giáo trình giúp sinh viên có cái nhìn tổng quan về ứng dụng Windows Application trong bộ công cụ Visual Studio .NET của Microsoft. Ngoài ra giáo trình còn cung cấp những kiến thức về các điều khiển mà Windows Application hỗ trợ, đi kèm đó là các ví dụ đơn giản được mô tả cụ thể để sinh viên có thể đọc và hiểu một cách nhanh chóng nhất.

Nội dung của giáo trình gồm 11 chương.

Chương 1: Giới thiệu tổng quan về ngôn ngữ C#, Cấu trúc của một chương trình C#, các kiểu dữ liệu cơ bản, cấu trúc lặp và cấu trúc rẽ nhánh, lớp String và cách tạo thư viện DLL.

Chương 2: Giới thiệu về ứng dụng Windows Form, tạo form với hình dạng khác nhau, các sự kiện và phương thức của form.

Chương 3: Giới thiệu về cách sử dụng của các điều khiển thông thường như: Label, Button, Textbox, ComboBox, ListBox, CheckBox và RadioButton.

Chương 4: Giới thiệu về cách sử dụng các điều khiển đặc biệt như: Tooltip, HelpProvider, ErrorProvider, ProgressBar, Timer, ListView, TreeView, DateTimePicker và MonthlyCalendar.

Chương 5: Giới thiệu về cách sử dụng của các điều khiển tạo Menu như: ToolStrip, ContextMenuStrip. Cách sử dụng các điều khiển: ImageList, NotifyIcon, ToolStrip, StatusStrip.

Chương 6: Giới thiệu về các điều khiển có thể chứa điều khiển khác như: Panel, GroupBox, FlowLayoutPanel, TableLayoutPanel, TabControl, SplitContainer.

Chương 7: Giới thiệu về các sử dụng của các điều khiển dạng hộp thoại: MessageBox, ColorDialog, FontDialog, OpenFileDialog, SaveFileDialog, FolderBrowseDialog.

Chương 8: Giới thiệu về các điều khiển làm việc với in ấn: PageSetupDialog, PrintPreviewDialog, PrintPreviewControl, PrintDialog, PrintDocument.

Chương 9: Giới thiệu về cách xây dựng và sử dụng UserControl.

Chương 10: Giới thiệu về các lớp đối tượng làm việc với màn hình và hệ thống như: Lớp SystemInformation, lớp Screen, lớp Sendkeys, lớp PowerStatus, Lớp Application, lớp Clipboard, lớp Cursor.

Chương 11: Bài tập tổng hợp kiến thức.

# MỤC LỤC

## Lời nói đầu

Chương 1: Giới thiệu về C# .....	1
1.1. Giới thiệu về Microsoft .NET Framework .....	1
1.1.1. .NET Framework là gì .....	1
1.1.2. Các thành phần của Microsoft .NET Framework .....	1
1.2. Tổng quan .....	2
1.3. Cấu trúc tổng quát của một chương trình C#.....	4
1.3.1. Soạn thảo bằng Notepad .....	4
1.3.2. Soạn thảo bằng Microsoft Visual Studio 2010 .....	5
1.4. Biến và kiểu dữ liệu .....	8
1.4.1. Biến .....	8
1.4.2. Kiểu dữ liệu.....	9
1.4.2.1. Kiểu giá trị .....	9
1.4.2.2. Kiểu tham chiếu .....	10
1.4.2.3. So sánh sự khác biệt giữa kiểu giá trị và kiểu tham chiếu .....	11
1.4.2.4. Chuyển kiểu dữ liệu .....	11
1.5. Câu lệnh phân nhánh.....	13
1.5.1. Câu lệnh if .....	13
1.5.2. Câu lệnh switch .....	13
1.6. Câu lệnh lặp .....	14
1.6.1. Lệnh lặp while.....	14
1.6.2. Lệnh lặp do/ while.....	15
1.6.3. Lệnh lặp for .....	15
1.6.4. Lệnh lặp foreach.....	16
1.7. Lớp String .....	17
1.7.1. Giới thiệu về chuỗi ký tự .....	17
1.7.2. Phương thức và thuộc tính lớp String .....	18
1.8. Mảng .....	31
1.8.1. Mảng một chiều .....	31
1.8.2. Mảng hai chiều .....	32
1.9. Tạo và sử dụng DLL trong C# .....	33
1.9.1. Ưu điểm và nhược điểm khi sử dụng DLL .....	33

1.9.2. Các bước để tạo tập tin DLL.....	34
1.9.3. Các bước để sử dụng tập tin DLL .....	36
1.10. Bài tập cuối chương.....	39
 Chương 2: Giới thiệu về Windows Forms .....	40
2.1. Giới thiệu .....	40
2.1.1. Ứng dụng Windows Forms .....	40
2.1.2. Không gian tên (namespace).....	44
2.1.3. Thanh công cụ (Toolbox).....	46
2.1.4. Định dạng mã C# .....	48
2.2. Các loại Form.....	48
2.2.1. Thuộc tính Form.....	48
2.2.2. Các loại Form.....	53
2.2.3. Các hình dạng của Form .....	55
2.2.4. Biến cố của Form .....	57
2.2.5. Phương thức .....	60
2.3. Bài tập cuối chương .....	64
 Chương 3: Các điều khiển thông thường .....	65
3.1. Điều khiển Label.....	66
3.2. Điều khiển Button.....	70
3.3. Điều khiển TextBox.....	73
3.4. Điều khiển ComboBox và ListBox.....	77
3.4.1. ListBox .....	78
3.4.2. ComboBox .....	82
3.4.3. Phương thức và sự kiện của ComboBox và ListBox .....	85
3.5. Điều khiển CheckBox và RadioButton.....	89
3.5.1. CheckBox .....	89
3.5.2. RadioButton .....	93
3.6. Bài tập cuối chương .....	97
 Chương 4: Các điều khiển đặc biệt .....	102
4.1. Điều khiển Tooltip, HelpProvider, ErrorProvider .....	103
4.1.1. Điều khiển Tooltip .....	103
4.1.2. Điều khiển HelpProvider .....	107
4.1.3. Điều khiển ErrorProvider .....	110
4.2. Điều khiển ProgressBar và Timer.....	115
4.2.1. Điều khiển ProgressBar .....	115
4.2.2. Điều khiển Timer .....	116

4.3. Điều khiển ListView .....	120
4.4. Điều khiển TreeView .....	130
4.5. Điều khiển DateTimePicker, MonthCalendar .....	136
4.5.1. Điều khiển DateTimePicker.....	136
4.5.2. Điều khiển MonthCalendar .....	138
4.6. Bài tập cuối chương .....	142
 Chương 5: Điều khiển dùng để xây dựng Menu .....	146
5.1. Điều khiển ImageList.....	146
5.2. Điều khiển ToolStrip.....	149
5.3. Điều khiển ContextMenuStrip .....	159
5.4. Điều khiển NotifyIcon .....	164
5.5. Điều khiển ToolStrip .....	167
5.5.1. Điều khiển chứa trong ToolStrip .....	170
5.5.2. ToolStripContainer.....	172
5.6. Điều khiển StatusStrip .....	178
5.7. Bài tập cuối chương .....	180
 Chương 6: Điều khiển chứa các điều khiển khác .....	185
6.1. Điều khiểnGroupBox .....	185
6.2. Điều khiển Panel.....	190
6.3. Điều khiển FlowLayoutPanel .....	191
6.4. Điều khiển TableLayoutPanel .....	197
6.5. Điều khiển TabControl .....	200
6.6. Điều khiển SplitContainer .....	211
6.7. Bài tập cuối chương .....	218
 Chương 7: Điều khiển Dialog và phương thức Message .....	222
7.1. Lớp MessageBox .....	223
7.2. Điều khiển ColorDialog.....	228
7.3. Điều khiển FontDialog.....	231
7.4. Điều khiển OpenFileDialog .....	234
7.5. Điều khiển SaveFileDialog .....	238
7.6. Điều khiển FolderBrowserDialog .....	243
7.7. Bài tập cuối chương .....	248
 Chương 8: Làm việc với điều khiển in ấn .....	253
8.1. Điều khiển PageSetupDialog .....	254
8.2. Điều khiển PrintPreviewDialog .....	258

8.3. Điều khiển PrintPreviewControl.....	261
8.4. Điều khiển PrintDialog .....	266
8.5. Điều khiển PrintDocument .....	273
Chương 9: Điều khiển người dùng tự tạo	
9.1. User Control.....	280
9.2. Xây dựng User Control.....	280
9.2.1. Thêm mới User Control ngay trong dự án .....	280
9.2.2. Tạo lớp và khai báo thừa kế từ lớp Control .....	285
9.2.3. Tạo dự án mới bằng cách chọn loại dự án là Windows Control Library...	288
9.3. Sử dụng User Control .....	291
9.4. Cách sử dụng những thư viện tạo sẵn để thiết kế giao diện .....	294
9.5. Bài tập cuối chương .....	294
Chương 10: Làm việc với màn hình và hệ thống .....	
10.1. Lớp SystemInformation .....	298
10.2. Lớp Screen .....	301
10.3. Lớp SendKeys.....	304
10.4. Lớp PowerStatus .....	309
10.5. Lớp Application .....	310
10.6. Lớp Clipboard .....	312
10.7. Lớp Cursors .....	316
Chương 11: Bài tập tổng hợp .....	
Tài liệu tham khảo .....	318
	325

# CHƯƠNG 1: GIỚI THIỆU VỀ C#

## 1.1. Giới thiệu về Microsoft .NET Framework

### 1.1.1. .NET Framework là gì ?

.NET Framework là nền tảng phát triển hoàn hảo của Microsoft, cung cấp cho lập trình viên các thư viện dùng chung hỗ trợ cho việc phát triển các kiểu ứng dụng khác nhau bao gồm:

- Ứng dụng ASP.NET
- Ứng dụng Windows Form
- Web Services
- Windows Services
- Ứng dụng mạng và các ứng dụng điều khiển truy cập từ xa

### 1.1.2. Các thành phần của Microsoft .NET Framework

.NET Framework gồm hai thành phần chính là: Common Language Runtime (CLR) và thư viện lớp.

- CLR: là nền tảng của .NET Framework, giúp Microsoft có thể tích hợp nhiều ngôn ngữ lập trình khác nhau như: VB.NET, C#.NET, ASP.NET,... vào bộ công cụ lập trình Visual Studio.NET. Đồng thời giúp các ứng dụng viết trên các ngôn ngữ này có thể chạy được chung trên nền tảng hệ điều hành Windows. Sở dĩ Microsoft có thể làm được điều này, bởi vì các ngôn ngữ lập trình đều được biên dịch ra một ngôn ngữ trung gian (Intermediate Language – IL) và sử dụng chung kiểu dữ liệu hệ thống (Common Type System). Sau đó CLR sử dụng một trình biên dịch gọi là Just-in-Time (JIT) Compiler chuyển các đoạn mã IL thành mã máy và thực thi.

Ngoài ra CLR còn làm các thành phần khác như:

- *Garbage Collection (GC)*: Gọi là bộ phận thu gom rác; có chức năng tự động quản lý bộ nhớ. Tại một thời điểm nhất định sẵn, GC sẽ tiến hành thực hiện việc thu hồi những vùng nhớ không còn được sử dụng.
- *Code Access Security (CAS)*: Cung cấp quyền hạn cho các chương trình, tùy thuộc vào các thiết lập bảo mật của máy. Chẳng hạn, thiết lập bảo mật của máy cho phép chương trình chạy trên đó được sửa hay tạo file mới, nhưng không cho phép nó xóa file. CAS sẽ chăm sóc các đoạn mã, không cho phép chúng làm trái với các qui định này.
- *Code Verification*: Bộ phận chứng nhận đoạn mã. Bộ phận này đảm bảo cho việc chạy các đoạn mã là đúng đắn, và đảm bảo an toàn kiểu dữ liệu.

ngăn chặn các đoạn mã hoạt động vượt quyền như truy nhập vào các vùng nhớ không được phép.

- Thư viện lớp: là một tập hợp lớn các lớp được viết bởi Microsoft, những lớp này được xây dựng một cách trực quan và dễ sử dụng; cho phép lập trình viên thao tác rất nhiều các tác vụ sẵn có trong Windows.
  - Base class library: Đây là thư viện các lớp cơ bản nhất, được dùng trong khi lập trình hay bản thân những người xây dựng .NET Framework cũng phải dùng nó để xây dựng các lớp cao hơn. Một số thư viện lớp base class library như: String, Interger, Exception, ...
  - ADO.NET và XML: Bộ thư viện này gồm các lớp dùng để xử lý dữ liệu. ADO.NET thay thế ADO để trong việc thao tác với các dữ liệu thông thường. Các lớp đối tượng XML được cung cấp để xử lý các dữ liệu theo định dạng mới XML. Một số thư viện trong ADO.NET và XML như: sqlDataAdapter, SqlCommand, DataSet, ...
  - Windows Forms: Bộ thư viện về lập trình Windows Forms gồm các lớp đối tượng dành cho việc xây dựng các ứng dụng Windows cơ bản. Một số thư viện thường dùng như: Form, UserControl, TextBox, Label, Button, ComboBox, ListBox, ListView, TreeView, ...
  - Web Services: là các dịch vụ được cung cấp qua Web (hay Internet). Dịch vụ được coi là Web Service không nhắm vào người dùng mà nhắm vào người xây dựng phần mềm. Web Services có thể dùng để cung cấp các dữ liệu hay một chức năng tính toán.
  - ASP.NET: Ứng dụng Web xây dựng bằng ASP.NET tận dụng được toàn bộ khả năng của .NET Framework. ASP.Net cung cấp một bộ các Server Control để lập trình viên bắt sự kiện và xử lý dữ liệu của ứng dụng như đang làm việc với ứng dụng của Windows. Một số thư như: WebControl, HTML Control, ...

## 1.2. Tổng quan về C#

Vào ngày 12/2/2002, Microsoft chính thức cho ra mắt Visual Studio 2002 cùng với bản .NET Framework 1.0. Với nền tảng .NET Framework thì các ứng dụng về Windows Form, web, ... đều có thể được xây dựng trên nền tảng .NET. Hai ngôn ngữ chính mà Microsoft sử dụng để phát triển các ứng dụng trên là C# và Visual Basic.NET.

C# được phát triển bởi đội ngũ kỹ sư của Microsoft, trong đó hai người dẫn đầu là Anders Hejlsberg và Scott Wiltamult. Vì là ngôn ngữ ra đời sau các ngôn ngữ khác

như: Java, C, C++, Perl, Visual Basic, ... do đó C# có những đặc điểm nổi trội và mạnh mẽ hơn nhưng không kém phần đơn giản.

➤ Các đặc điểm của C#:

Microsoft thêm vào những đặc điểm mới vào để C# dễ sử dụng hơn:

- C# là ngôn ngữ đơn giản: Các cú pháp, biểu thức, toán tử đều giống với C, C++. Ngoài ra C# loại bỏ một số khái niệm phức tạp khó hiểu như: Con trỏ, Macro, đa kế thừa, template, lớp cơ sở ảo.
- C# là ngôn ngữ hiện đại: Có tất cả các đặc tính của một ngôn ngữ hiện đại như: xử lý ngoại lệ, thu gom bộ nhớ tự động, kiểu dữ liệu mở rộng, và bảo mật mã nguồn.
- C# là ngôn ngữ hướng đối tượng: Là một ngôn ngữ thuận hướng đối tượng, do đó chứa tất cả các đặc điểm để có thể lập trình theo hướng đối tượng: Sự đóng gói (Encapsulation), sự thừa kế (Inheritance), và sự đa hình (Polymorphism).
- C# là ngôn ngữ mạnh mẽ và mềm dẻo: Có thể sử dụng cho những dự án khác nhau như: xử lý văn bản, ứng dụng đồ họa, bảng tính, ...
- C# là ngôn ngữ có ít từ khóa: là ngôn ngữ sử dụng giới hạn các từ khóa, chỉ khoảng 80 từ khóa và hơn 10 kiểu dữ liệu được xây dựng sẵn như bảng 1.1.

Bảng 1.1: Bảng mô tả các từ khóa của C#

abstract	default	foreach	object	sizeof	unsafe
as	delegate	goto	operator	stackalloc	ushort
base	do	if	out	static	using
bool	double	implicit	override	string	virtual
break	else	in	params	struct	volatile
byte	enum	int	private	switch	void
case	event	interface	protected	this	while
catch	explicit	internal	public	throw	decimal
char	extern	is	readonly	true	for
checked	false	lock	ref	try	null
class	finally	long	return	typeof	short
const	fixed	namespace	sbyte	uint	unchecked

continue	float	new	sealed	ulong	
----------	-------	-----	--------	-------	--

### 1.3. Cấu trúc tổng quát của một chương trình C#

Các đoạn mã C# có thể được soạn thảo trong một trình soạn thảo văn bản bất kỳ, sau khi biên soạn xong cần lưu tập tin với đuôi .cs

#### 1.3.1. Soạn thảo bằng Notepad

Trong quá trình soạn thảo cần lưu ý C# là một ngôn ngữ thuận hướng đối tượng. Do đó tất cả đoạn mã của chương trình đều phải thuộc lớp.

Ví dụ 1.1: Soạn thảo một chương trình C# in dòng chữ Hello World ra màn hình

➤ Bước 1:

```
namespace hello
{
    class Program
    {
        static void Main()
        {
            System.Console.WriteLine("Hello World");
            System.Console.ReadLine();
        }
    }
}
```

Giải thích:

- Ở đoạn chương trình trên, hàm Main được khai báo là static. Việc khai báo này cho biết hàm Main có thể được sử dụng mà không cần thông qua đối tượng
- Hàm WriteLine nằm trong lớp **Console**, do đó nếu sử dụng phải khai báo thêm dòng lệnh “**Using System;**” phía trên cùng để việc gõ lệnh được nhanh chóng hơn như sau: “**Console.WriteLine()**”. Nếu không khai báo “**Using System;**” thì buộc phải viết đầy đủ cú pháp: “**System.Console.WriteLine()**”, với System là một namespace.
- C# phân biệt chữ thường và chữ hoa, do đó lệnh writeline khác lệnh WriteLine.

➤ Bước 2: Khi soạn thảo xong. Lưu tập tin với tên: **ChaoMung.cs**

➤ Bước 3: Thực thi chương trình phải thực hiện hai bước:

- Mở cửa sổ Command Line và biên dịch tập tin ChaoMung.cs vừa tạo ra sang mã máy (tạo ra file có đuôi .exe) như hình 1.1: Để biên dịch dùng trình biên dịch dòng lệnh C# (csc.exe) chương trình này được chép vào máy

trong quá trình cài .NET Framework. Nếu cài Framework 4.0 thì tiến hành vào đường dẫn:

C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe

**Cú pháp:**

csc.exe [/out: <file thực thi>] <file nguồn>

A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command entered is 'c:\Windows\Microsoft.NET\Framework\v4.0.30319>csc.exe /out:d:\hello.exe d:\hello.cs'. The output shows multiple lines of the same path being repeated, followed by the command itself.

Hình 1.1: Biên dịch tập tin hello.cs

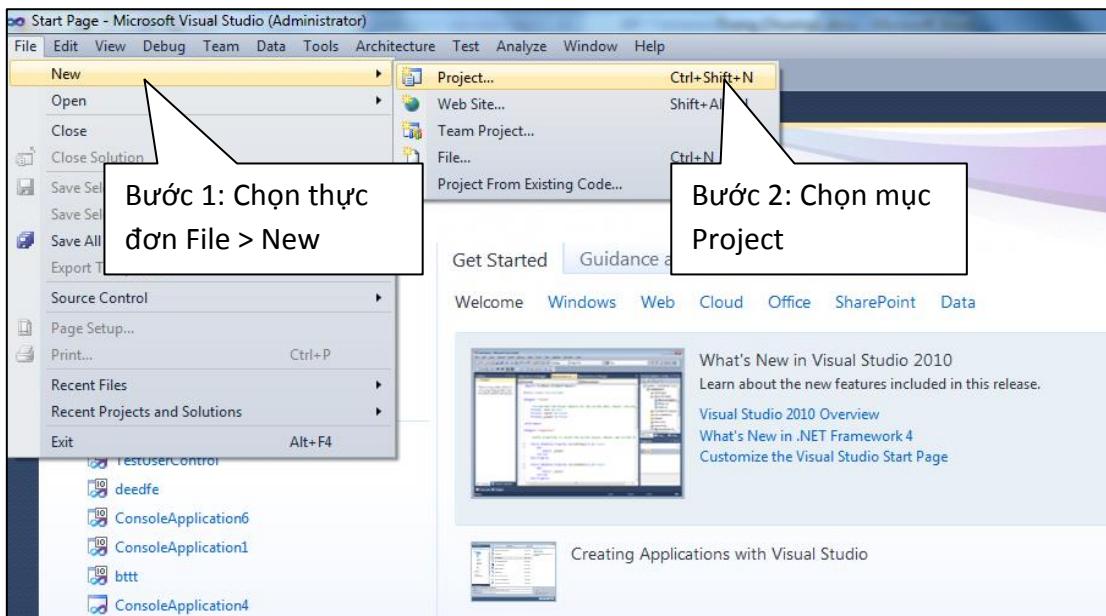
- Thực thi tập tin có đuôi .exe như hình 1.2:

A screenshot of a Windows Command Prompt window titled 'C:\Windows\system32\cmd.exe'. The command entered is 'c:\Windows\Microsoft.NET\Framework\v4.0.30319>d:\hello.exe'. The output shows multiple lines of the same path being repeated, followed by the command itself.

Hình 1.2: Thực thi tập tin Hello.exe

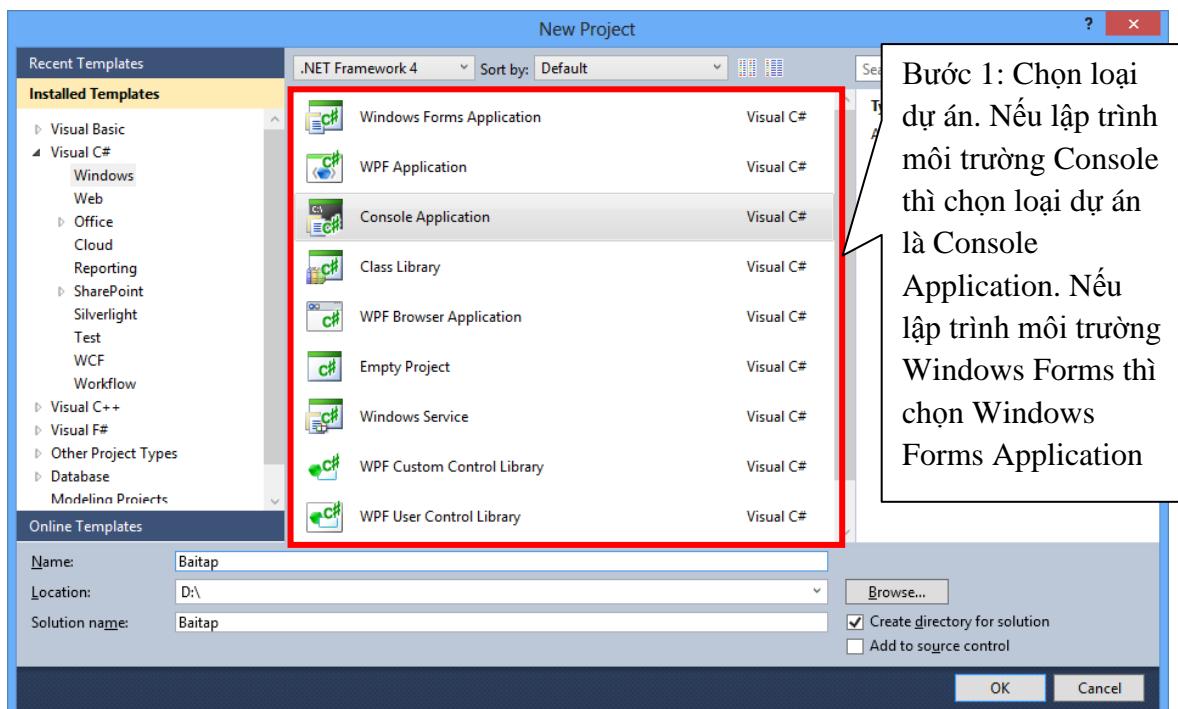
### 1.3.2. Soạn thảo bằng Microsoft Visual Studio 2010

- Bước 1: Mở Microsoft Visual Studio 2010 và tạo 1 dự án mới như hình 1.3



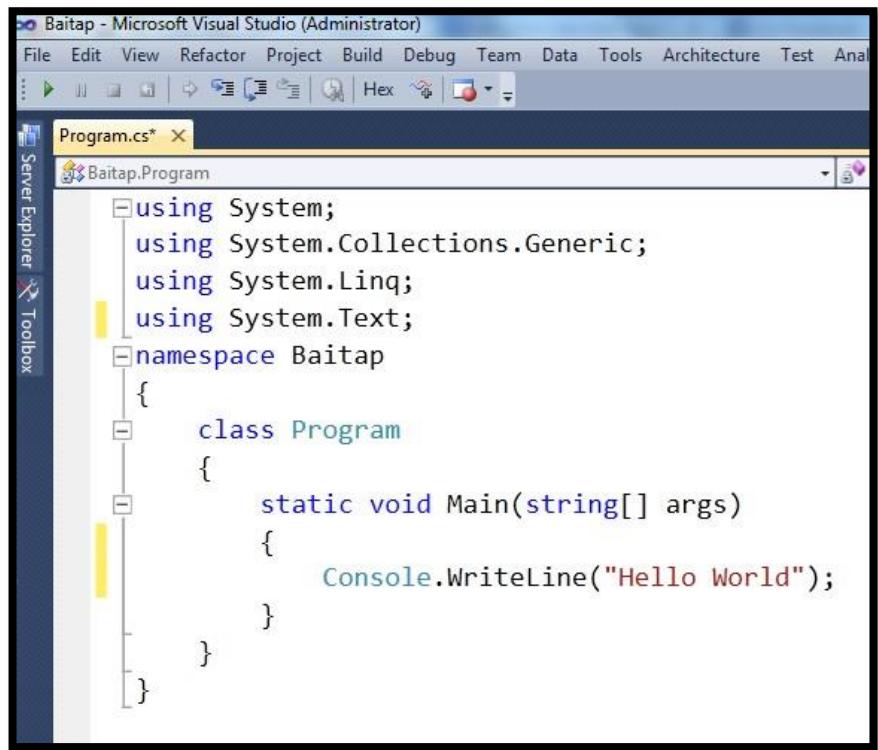
Hình 1.3: Tạo dự án mới

- Bước 2: Tạo một loại dự án đơn giản là Console Application để minh họa bằng ngôn ngữ C# và đặt tên Baitap lưu ở ô đĩa D như hình 1.4



Hình 1.4: Tạo loại dự án và đặt tên cho dự án

- Bước 3: Soạn thảo mã C# trong trình biên dịch như hình 1.5

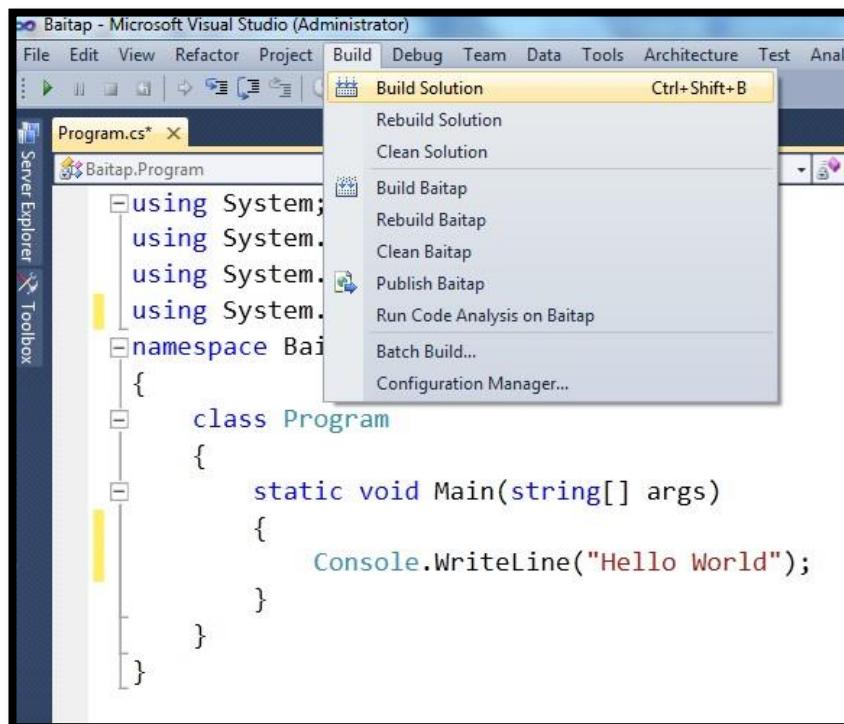


The screenshot shows the Microsoft Visual Studio interface with the title bar "Baitap - Microsoft Visual Studio (Administrator)". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Team, Data, Tools, Architecture, Test, and Analyze. The toolbar has icons for file operations like Open, Save, and Build. The code editor window displays the following C# code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Baitap
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
        }
    }
}
```

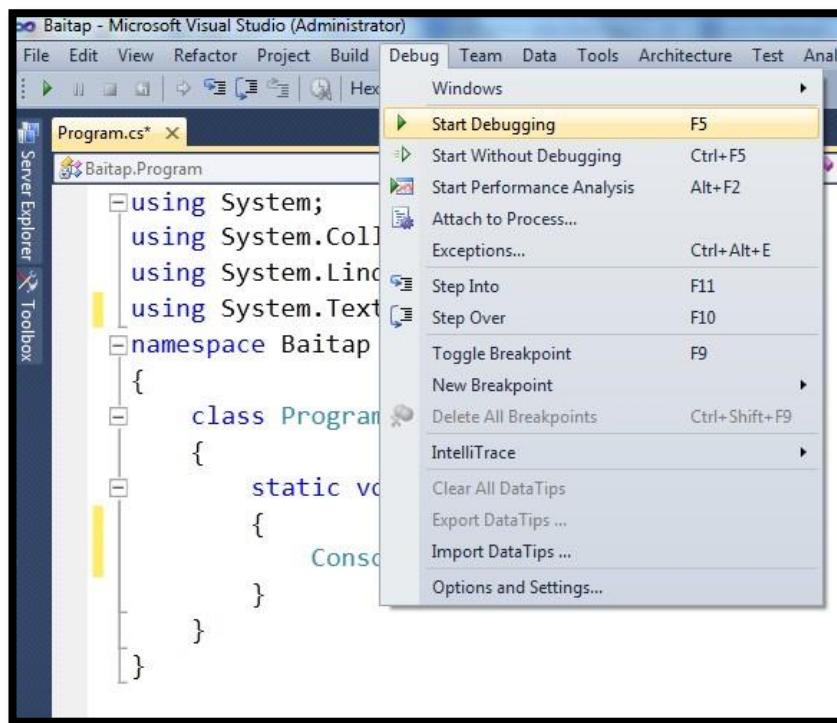
Hình 1.5: Giao diện soạn thảo mã lệnh

- Bước 4: Biên dịch chương trình: trên thanh menu chọn Build > Build Solution như hình 1.6 hoặc ấn tổ hợp phím Ctrl + Shift + B.



Hình 1.6: Biên dịch chương trình

- Bước 5: Thực thi chương trình: trên thanh menu chọn Debug > Start Debugging như hình 1.7 hoặc phím F5.



Hình 1.7: Thực thi chương trình

## 1.4. Biến và Kiểu dữ liệu

### 1.4.1. Biến

Biến là nơi lưu trữ dữ liệu của chương trình. Dữ liệu của biến nằm trong bộ nhớ vật lý của Ram và có thể thay đổi được. Muốn sử dụng biến trước tiên lập trình phải chỉ định biến có một kiểu dữ liệu cụ thể nào đó.

- Cú pháp khai báo biến:

**<Kiểu dữ liệu> <Tên biến>**

Lưu ý: Tên biến phải được đặt phù hợp với quy tắc đặt tên

- Cú pháp khởi tạo giá trị cho biến:

**<Tên biến> = <giá trị>**

Trong đó:

= là phép toán gán giá trị

**Quy tắc đặt tên:** Các tên biến, tên hằng, tên hàm, ... trong C# đều phải đặt tên đúng với quy tắc sau:

- Một tên biến có thể bắt đầu bằng chữ hoa hay chữ thường. Tên có thể chứa ký tự hay số và ký tự gạch dưới (\_).
- Ký tự đầu tiên của biến phải là ký tự , không được là số.
- Trong C# phân biệt hoa thường.

- Các từ khóa không thể sử dụng để đặt tên cho biến. Nếu muốn dùng từ khóa đặt tên thì ta thêm ký tự @ phía trước.

Ví dụ 1.2:

Employee:	Hợp lệ
student:	Hợp lệ
_Name:	Hợp lệ
Emp_Name:	Hợp lệ
@goto:	Hợp lệ
static:	Không hợp lệ, trùng từ khóa
4myclass:	Không hợp lệ, không thể bắt đầu bằng ký tự số
Student&Falculty:	Không hợp lệ, không được chứa ký tự đặc biệt

### 1.4.2. Kiểu dữ liệu

Cũng như ngôn ngữ lập trình C++ hay Java, C# chia thành hai tập kiểu dữ liệu chính:

- Kiểu giá trị
- Kiểu tham chiếu

Mỗi kiểu dữ liệu trong C# sẽ thuộc một trong hai kiểu giá trị hoặc kiểu tham chiếu. Kiểu giá trị thì lưu trong stack, còn kiểu tham chiếu lưu trong heap.

**Stack:** một vùng bộ nhớ dành lưu trữ dữ liệu với chiều dài cố định.

Ví dụ 1.3: số nguyên kiểu int luôn chiếm dung lượng 4 bytes.

Mỗi chương trình khi thực thi đều được cấp riêng 1 stack mà các chương trình khác không truy cập tới được

**Heap:** một vùng bộ nhớ dùng lưu trữ dữ liệu có dung lượng thay đổi.

Ví dụ 1.4: như kiểu string, khi ta tạo đối tượng thuộc lớp string, thì đối tượng sẽ được xác định bởi hai thành phần: Địa chỉ đối tượng lưu ở stack, còn giá trị đối tượng thì sẽ lưu ở heap.

#### 1.4.2.1. Kiểu giá trị

Kiểu giá trị thường là các kiểu do C# định nghĩa sẵn bao gồm: double, char, int, float, enum, struct, ....

Biến của kiểu giá trị lưu trữ một giá trị thực, giá trị này được lưu trữ trong stack, không thể mang giá trị null và phải chứa giá trị xác định.

Bảng 1.2: Bảng mô tả các kiểu dữ liệu giá trị trong C#

Kiểu C#	Số Byte	Kiểu .NET	Mô tả
byte	1	Byte	Số nguyên dương không dấu: 0 đến 255
char	2	Char	Ký tự Unicode
bool	1	Boolean	Giá trị logic true/ false
sbyte	1	Sbyte	Số nguyên có dấu: -128 đến 127
short	2	Int16	Số nguyên có dấu: -32.768 đến 32.767

ushort	2	UInt16	Số nguyên không dấu: 0 đến 65.535
int	4	Int32	Số nguyên có dấu: -2.147.483.647 đến 2.147.483.647
uint	4	UInt32	Số nguyên không dấu: 0 đến 4.294.967.295
float	4	Single	Kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38; với 7 chữ số có nghĩa
double	8	Double	Kiểu dấu chấm động, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308; với 15, 16 chữ số có nghĩa
decimal	8	Decimal	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố "m" hay "M" theo sau giá trị
long	8	Int64	Kiểu số nguyên có dấu có giá trị trong khoảng: -9.233.370.036.854.775.808 đến -9.233.372.036.854.775.807
ulong	8	UInt64	Số nguyên không dấu từ 0 đến 0xffffffffffffffff

Lưu ý:

Kiểu số thực: Trình biên dịch luôn hiểu bất cứ một số thực nào cũng là một số kiểu double trừ khi khai báo rõ ràng, do đó để gán một số kiểu float thì số phải có ký tự f theo sau.

ví dụ 1.5: `float a=0.15f;`

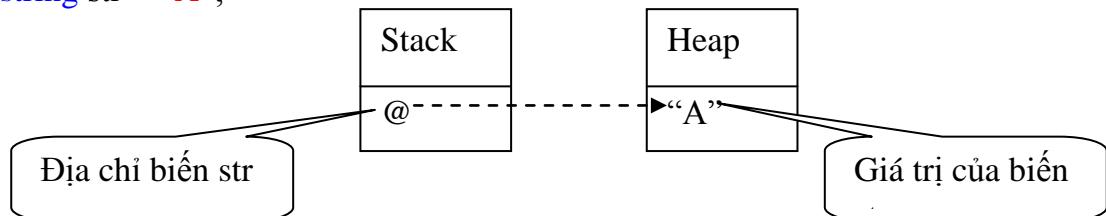
Kiểu giá trị bool trong C# nhất thiết phải là true hoặc false.

#### 1.4.2.2. Kiểu tham chiếu

Biến có kiểu dữ liệu tham chiếu lưu trữ địa chỉ của biến khác trên bộ nhớ heap. Như vậy các lớp đối tượng String, Object, ... đều được hiểu như kiểu dữ liệu tham chiếu. Ngoài ra các kiểu dữ liệu do người dùng xây dựng nên cũng được gọi là kiểu tham chiếu.

Ví dụ 1.6:

`string str = "A";`



### 1.4.2.3. Bảng so sánh sự khác biệt giữa kiểu giá trị và kiểu tham chiếu

Bảng 1.3: Bảng so sánh sự khác biệt giữa kiểu giá trị và kiểu tham chiếu

	Kiểu giá trị	Kiểu tham chiếu
Biến lưu trữ	Giá trị của biến	Địa chỉ của biến
Vị trí lưu trữ giá trị	Stack	Heap
Giá trị mặc định	0	Null
Phép toán gán	Sao chép giá trị	Sao chép địa chỉ

### 1.4.2.4. Chuyển kiểu dữ liệu

Việc chuyển đổi kiểu dữ liệu trong C# có thể thực hiện bằng một trong các cách sau:

- Sử dụng lớp Convert:

Đây là một lớp được C# xây dựng sẵn phục vụ cho việc chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác. Các phương thức trong lớp Convert phần lớn đều là phương thức tĩnh:

- `ToInt32(<Tham số>)`
- `ToInt64(<Tham số>)`
- `ToString(<Tham số>)`
- `.ToDouble(<Tham số>)`
- `ToBoolean(<Tham số>)`
- `ToByte(<Tham số>)`
- ...

`<Tham số>`: có thể là chuỗi ký tự, hằng số, biến số nguyên, số thực hoặc kiểu bool. Nếu trong quá trình chuyển kiểu đặt tham số là null thì hàm Convert sẽ trả về giá trị mặc định.

Ví dụ 1.7:

```
int a = Convert.ToInt32("10"); //chuyển chuỗi 10 sang số nguyên  
bool b = Convert.ToBoolean(27); //chuyển số 27 sang kiểu boolean  
bool a = Convert.ToBoolean("hello"); //Sai định dạng, không chuyển được  
int b = Convert.ToInt32("123456787654"); //Tràn bộ nhớ, không chuyển được  
double d = Convert.ToDouble(null); //Trả về giá trị mặc định
```

- Sử dụng phương thức Parse:

Phương thức Parse thường được sử dụng để chuyển đổi chuỗi sang một kiểu dữ liệu cụ thể nào đó. Mỗi kiểu dữ liệu xây dựng sẵn trong C# đều có hỗ trợ phương thức Parse để chuyển đổi chuỗi sang kiểu dữ liệu tương ứng:

- `Double.Parse(<chuoi>)`: Chuyển chuỗi về kiểu Double
- `Int32.Parse(<chuoi>)`: Chuyển chuỗi về kiểu Int32
- `Int64.Parse(<chuoi>)`: Chuyển chuỗi về kiểu Int64
- `Boolean.Parse(<chuoi>)`: Chuyển chuỗi về kiểu Boolean
- `Single.Parse(<chuoi>)`: Chuyển chuỗi về kiểu Single
- ...

Ví dụ 1.8:

```
int a = Int32.Parse("123"); //a sẽ mang giá trị 123
float b = Float.Parse("20.7"); //b sẽ mang giá trị 20.7
bool c = Boolean.Parse("true"); //c sẽ mang giá trị true
int a = Int32.Parse("Hello"); //sai định dạng, không chuyển được
byte b = Byte.Parse("1000000000"); //quá giới hạn, không chuyển được
bool c = Boolean.Parse(null); //tham số là null, không chuyển được
```

➤ Sử dụng phương thức TryParse:

Phương thức TryParse cũng được sử dụng để chuyển đổi chuỗi sang một kiểu dữ liệu cụ thể như Parse. Các kiểu dữ liệu xây dựng sẵn trong C# đều có phương thức TryParse. Tuy nhiên, sự khác biệt của TryParse và Parse là:

- Cú pháp sử dụng có sự khác biệt: Có hai tham số truyền vào TryParse; Tham số thứ nhất là chuỗi cần chuyển đổi và tham số thứ hai là biến chứa giá trị chuyển đổi (biến truyền vào phải truyền ở dạng tham chiếu)
- Giá trị trả về của TryParse sẽ là true nếu chuyển kiểu thành công và trả về false nếu chuyển kiểu không thành công.

Ví dụ 1.9:

```
int a;
Int32.TryParse("123", out a); //a sẽ mang giá trị 123
bool b;
Boolean.TryParse("false", out b); //b sẽ mang giá trị false
int a;
Int32.TryParse("hello", out a); //trả về giá trị false, a mang giá trị 0
bool b;
Boolean.TryParse("", out b); //trả về giá trị false, b mang giá trị false
```

➤ Casting (ép kiểu):

Đây là cách chuyển kiểu được sử dụng khi muốn chuyển đổi giữa những kiểu dữ liệu gần tương tự nhau. Thường áp dụng với kiểu số.

Ví dụ 1.10:

```
int x= 10;
float y = x;      //chuyển đổi ngầm định, y = 10
int z = (int)y;   //chuyển đổi rõ ràng, z = 10
int x = 10;
```

Trong quá trình chuyển kiểu có thể xảy ra lỗi do chuyển giữa các kiểu dữ liệu không tương thích: Kiểu chuỗi sang kiểu số, kiểu bool sang kiểu float, ...

## 1.5. Câu lệnh phân nhánh

Việc phân nhánh có thể được tạo ra bằng các từ khóa: if, else, switch, case. Sự phân nhánh chỉ được thực hiện khi biểu thức điều kiện phân nhánh được xác định là đúng.

### 1.5.1. Câu lệnh if

➤ Cú pháp:

```
if (biểu thức điều kiện)
    <Câu lệnh thực hiện khi điều kiện đúng>
[else
    <Câu lệnh thực hiện khi điều kiện sai> ]
```

Nếu các câu lệnh trong thân của if hay else mà lớn hơn một lệnh thì các lệnh này phải được bao trong một khối lệnh, tức là phải nằm trong dấu khối { }:

➤ Cú pháp:

```
if (biểu thức điều kiện)
{
    <lệnh 1>; <lệnh 2>;....
}
[else
{
    <lệnh 3>; <lệnh 3>;....
}]
```

### 1.5.2. Câu lệnh switch

Khi có quá nhiều điều kiện để chọn thực hiện thì dùng câu lệnh if sẽ rất rối rắm và dài dòng, Các ngôn ngữ lập trình cấp cao đều cung cấp một dạng câu lệnh switch liệt kê các giá trị và chỉ thực hiện các giá trị thích hợp.

- Cú pháp:

```
switch (biểu thức điều kiện)
{
    case <giá trị>:
        <Các câu lệnh thực hiện>
        <lệnh nhảy>
    [default:
        <Các câu lệnh thực hiện mặc định>
}
}
```

Lưu ý:

- <lệnh nhảy>: Là lệnh chỉ định việc kết thúc một case, lệnh nhảy trong switch là break hoặc goto; Trường hợp trong case không có lệnh nhảy thì trình biên dịch sẽ tuần tự thực hiện các câu lệnh trong từng case của switch theo thứ tự từ trên xuống.
- Trong trường hợp switch sử dụng nhiều case, có thể nhảy trực tiếp đến case cụ thể nào đó bằng goto; hoặc thoát khỏi switch sử dụng break.
- <giá trị>: là hằng nguyên hoặc biến; Nếu là biến bắt buộc phải là biến kiểu số nguyên.

## 1.6. Câu lệnh lặp

C# kế thừa cú pháp câu lệnh của C/C++:

- Lệnh lặp while
- Lệnh lặp do/ while
- Lệnh lặp for

Ngoài ra 3 lệnh lặp cơ bản trên, C# còn có thêm lệnh lặp foreach dùng để làm việc với mảng, tập hợp.

C# cũng cung cấp các lệnh nhảy như: break, goto, continue và return sử dụng kết hợp với lệnh lặp

### 1.6.1. Lệnh lặp while

Ý nghĩa của vòng lặp while là: “Trong khi điều kiện đúng thì thực hiện các công việc này”.

- Cú pháp:

```
while (Biểu thức)
{
    <Câu lệnh thực hiện>
}
```

Biểu thức của vòng lặp **while** là điều kiện để các lệnh được thực hiện, biểu thức này bắt buộc phải trả về một giá trị kiểu bool là true/false

Ví dụ 1.11: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
    int i = 1, tong = 0;
    while (i < 10)
    {
        tong = tong + i; i = i+2;
    }
}
```

### 1.6.2. Lệnh lặp do/ while

Ý nghĩa của vòng lặp **do/ while** là: “*làm điều này trong khi điều kiện vẫn còn đúng*”.

➤ Cú pháp

```
do
{
    <Câu lệnh thực hiện>
}while ( điều kiện );
```

Sự khác biệt tối quan trọng của **while** và **do..while** là tối thiểu sẽ có một lần các câu lệnh trong **do...while** được thực hiện

Ví dụ 1.12: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
    int i = 1, tong = 0;
    do
    {
        tong = tong + i;
        i = i+2;
    }while(i<10);
```

### 1.6.3. Lệnh lặp for

Vòng lặp for bao gồm ba phần chính:

- Khởi tạo biến đếm vòng lặp
- Kiểm tra điều kiện biến đếm, nếu đúng thì sẽ thực hiện các lệnh bên trong vòng for

➤ Thay đổi bước lặp.

➤ Cú pháp:

```
for ( [phần khởi tạo] ; [biểu thức điều kiện]; [bước lặp])
{
    <Câu lệnh thực hiện>
}
```

Ví dụ 1.13: Viết chương trình tính tổng các số lẻ >0 và nhỏ hơn 10

```
static void Main(string[] args)
{
    int i, tong = 0;
    for(i=1; i < 10; i=i+2)
        tong = tong + i;
}
```

#### 1.6.4. Lệnh lặp foreach

Vòng lặp foreach cho phép tạo vòng lặp thông qua một tập hợp hay một mảng.

➤ Cú pháp:

```
foreach ( <kiểu tập hợp> <tên truy cập thành phần> in <tên tập hợp>)
{
    <Các câu lệnh thực hiện>
}
```

Do lặp dựa trên một mảng hay tập hợp nên toàn bộ vòng lặp sẽ duyệt qua tất cả các thành phần của tập hợp theo thứ tự được sắp. Khi duyệt đến phần tử cuối cùng trong tập hợp thì chương trình sẽ thoát ra khỏi vòng lặp foreach

Ví dụ 1.14: Tính tổng các số lẻ trong mảng A = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

```
static void Main(string[] args)
{
    int[] intArray = {0,1,2,3,4,5,6,7,8,9,10}, tong = 0;
    foreach( int item in intArray)
    {
        if (item % 2 != 0)
            tong = tong + item;
    }
}
```

## 1.7. Lớp String

### 1.7.1. Giới thiệu về chuỗi ký tự

C# xem những chuỗi như là những kiểu dữ liệu cơ bản, việc sử dụng chuỗi trong C# rất linh hoạt, mạnh mẽ, và quan trọng nhất là dễ sử dụng. Chỉ mục trong chuỗi được tính từ số 0.

Để làm việc với chuỗi, C# cung cấp lớp System.String. Khi khai báo một chuỗi C# bằng cách dùng từ khóa string, đồng nghĩa với việc người dùng đã khai báo một đối tượng của lớp System.String.

Ví dụ 1.15:

```
string chuoit="hello";
```

Ngoài việc, ta cũng có thể khởi tạo chuỗi từ 1 mảng char:

Ví dụ 1.16:

```
// Khởi tạo mảng ký tự
char[] chars = {'c', 's', 'h', 'a', 'r', 'p'};
// Khởi tạo chuỗi từ mảng ký tự
string str = new string(chars);
```

Lưu ý:

- Khi gán giá trị trong chuỗi có thể chứa ký tự escape: '\n', '\t', '\a', ...

Ví dụ 1.17:

```
string newString = "Day la chuoi \n trich dan";
```

Ký tự xuống

- Trong trường hợp muốn lưu đường dẫn thư mục vào một chuỗi:

Ví dụ 1.18:

```
string path = " c:\baitap";
```

Trình biên dịch báo lỗi, vì  
không hiểu ký tự escape '\b'

- Do đó muốn trình biên dịch hiểu '\b' như những ký tự thông thường thì người dùng phải thêm một ký tự '\\' phía trước:

Ví dụ 1.19:

```
string path = " c:\\baitap";
```

- Ngoài việc sử dụng dấu \ để giữ nguyên các ký tự trong chuỗi ta cũng có thể khai báo chuỗi như là 1 chuỗi nguyên văn bằng cách thêm @ phía trước chuỗi.

Ví dụ 1.20:

```
string path = @“D:\\soft\\VMWare”
```

### 1.7.2. Phương thức và thuộc tính lớp String

Bảng 1.4: Bảng tóm hợp các phương thức lớp String

System.String	
Phương thức/ Thuộc tính	Ý nghĩa
Empty	Thuộc tính tĩnh; Thể hiện chuỗi rỗng
Compare()	Phương thức tĩnh; So sánh hai chuỗi
CompareOrdinal()	Phương thức tĩnh; So sánh hai chuỗi không quan tâm đến thứ tự
Concat()	Phương thức tĩnh; Tạo ra chuỗi mới từ một hay nhiều chuỗi khác
Copy()	Phương thức tĩnh; Tạo ra một chuỗi mới bằng cách sao chép từ chuỗi khác
Equal()	Phương thức tĩnh; Kiểm tra xem hai chuỗi có cùng giá trị hay không
Insert()	Trả về chuỗi mới đã được chèn một chuỗi xác định
LastIndexOf()	Chỉ ra vị trí xuất hiện cuối cùng của một chuỗi xác định trong chuỗi ban đầu
PadLeft()	Canh lề phải những ký tự trong chuỗi, chèn vào bên trái khoảng trắng hay các ký tự xác định
PadRight()	Canh lề trái những ký tự trong chuỗi, chèn vào bên phải khoảng trắng hay các ký tự xác định
Remove()	Xóa đi một số ký tự xác định
Split()	Trả về chuỗi được phân định bởi những ký tự xác định trong chuỗi
StartsWith()	Xem chuỗi có bắt đầu bằng một số ký tự xác định hay không
SubString()	Lấy một chuỗi con
ToCharArray()	Sao chép những ký tự từ một chuỗi đến mảng ký tự
ToLower()	Trả về bản sao của chuỗi ở kiểu chữ thường
ToUpper()	Trả về bản sao của chuỗi ở kiểu chữ hoa
Format()	Phương thức tĩnh; Định dạng một chuỗi dùng ký tự lệnh định dạng xác định
Join()	Phương thức tĩnh; Kết nối các chuỗi xác định giữa mỗi thành phần của mảng chuỗi
Length	Trả về chiều dài của chuỗi
CompareTo()	So sánh hai chuỗi
CopyTo()	Sao chép một số các ký tự xác định đến một mảng ký tự Unicode
EndsWith()	Chỉ ra vị trí của chuỗi xác định phù hợp với chuỗi đưa ra
Trim()	Xóa bỏ tất cả sự xuất hiện của khoảng trắng trong chuỗi
TrimEnd()	Xóa các khoảng trắng ở vị trí cuối
TrimStart()	Xóa các khoảng trắng ở vị trí đầu

- **Thuộc tính Empty:** Đại diện cho 1 chuỗi rỗng. Sử dụng khi muốn khai báo 1 chuỗi là rỗng.

Ví dụ 1.21:

```
string chuoit = string.Empty;
```

Khi ta khai báo như trên nghĩa là tương đương với khai báo:

```
string chuoit = "“”;
```

- **Thuộc tính Length:** Dùng để xác định chiều dài của chuỗi.

Ví dụ 1.22:

```
static void Main(string[] args)
{
    string str = "pham phuong nguyen";
    int chieudai = str.Length;
}
//Kết quả: chieudai = 18
```

- **Nhóm các phương thức so sánh chuỗi:** Compare, CompareOrdinal, CompareTo, Equal

- Phương thức Compare: Phương thức tĩnh so sánh hai chuỗi

Cách sử dụng:

```
int gt = String.Compare(<chuoit1>, <chuoit2>, [true]);
```

- Nếu gt = 0: hai chuỗi bằng nhau
- Nếu gt = 1: chuỗi 1 lớn hơn chuỗi 2
- Nếu gt = -1: chuỗi 1 nhỏ hơn chuỗi 2
- Nếu sử dụng phương thức không có tham số true, thì sẽ phân biệt chữ thường, chữ hoa. Nếu có tham số true thì sẽ không biệt chữ thường hay chữ hoa.

Ví dụ 1.23:

```
public static void Main(string []args)
{
    int gt = String.Compare("abDb", "abdb"); //gt = 1
    int gt1= String.Compare("abDb", "abdb", true); //gt1 = 0
    int gt2= String.Compare("abdaF", "abdb"); //gt2 = -1
}
```

- Phương thức so sánh chuỗi CompareOrdinal: So sánh chuỗi trả về một giá trị kiểu int; Lần lượt tìm từ đầu chuỗi đến cuối chuỗi nếu thấy có ký tự khác biệt giữa hai chuỗi. Tính hiệu mã ASCII của hai ký tự này và trả kết quả vừa tính được.

Cách sử dụng:

```
int gt = String.CompareOrdinal(<chuoi1>,<chuoi2>);
```

Ví dụ 1.24:

```
public static void Main(string []args)
{
    int gt1 = String.CompareOrdinal("abcd", "aBcd");      //gt1=32
    int gt2 = String.CompareOrdinal("abcd", "aBcdefg"); //gt2=32
    int gt3 = String.CompareOrdinal("Abcd", "aBcdefg"); //gt3=-32
    int gt4 = String.CompareOrdinal("abcd", "abcd");      //gt4=0
}
```

- Phương thức so sánh chuỗi CompareTo: Phương thức thành viên lớp, giúp so sánh 2 chuỗi và trả về 1 giá trị kiểu int.

Cách sử dụng:

```
int gt = <chuoi1>.CompareTo(<chuoi2>);
```

- Nếu gt = 0: hai chuỗi bằng nhau
  - Nếu gt = 1: chuỗi 1 lớn hơn chuỗi 2
  - Nếu gt = -1: chuỗi 1 nhỏ hơn chuỗi 2
- Phương thức so sánh chuỗi Equal: Phương thức này dùng để so sánh các chuỗi ký tự có giống nhau hay không. Nếu 2 chuỗi hoặc ký tự giống nhau sẽ trả về giá trị true , ngược lại trả về giá trị false.

Cách sử dụng:

```
bool gt = String.Equal(<str1>,<str2>, [StringComparisonType]);
```

Lưu ý:

Nếu sử dụng phương thức không chỉ định tham số StringComparisonType thì sẽ so sánh từng ký tự một giữa hai chuỗi và phân biệt hoa thường.

Nếu muốn không phân biệt chữ hoa thường thì thêm tham số StringComparison.OrdinalIgnoreCase

Ví dụ 1.25:

```
public static void Main(string []args)
{
    string str1 = "ITlab";
    string str2 = "itlac";
    if (String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
        MessageBox.Show("Chuoi giong nhau ");
    else
        MessageBox.Show("Chuoi khong giong nhau ");
}
```

➤ Nhóm phương thức nối chuỗi và chèn chuỗi: Concat, Join, Insert

- Phương thức Concat: Phương thức tĩnh Concat dùng để nối hai hay nhiều chuỗi lại với nhau và trả về 1 chuỗi mới

Cách sử dụng:

```
string <chuoiKQ> = String.Concat(<chuoi1>,<chuoi2>);
```

Ví dụ 1.26:

```
public static void Main(string []args)
{
    string chuoi1="abc";
    string chuoi2="aBc";
    string kq = String.Concat(chuoi1,chuoi2); //kq = "abcaBc"
}
```

- Phương thức Join: dùng để nối 1 mảng chuỗi lại với nhau, giữa các phần tử có thể chèn thêm 1 chuỗi nào đó để phân cách.

Cách sử dụng:

```
string <chuoiKQ> = String.Join(<chuỗi phân cách>,<Mảng chuỗi>);
```

Ví dụ 1.27:

```
public static void Main(string []args)
{
    string[] chuoi = { "mot", "hai", "ba" };
    string str = String.Join("---", chuoi);
}
//str = "mot---hai---ba"
```

- Phương thức Insert: Phương thức thuộc lớp, giúp chèn 1 chuỗi vào 1 vị trí xác định trong chuỗi khác.

Cách sử dụng:

```
string <chuoiKQ> = <Tên đối tượng>.Join(<vị trí chèn>,<Chuỗi chèn vào>);
```

Ví dụ 1.28:

```
public static void Main(string []args)
{
    string chuoi = "mot hai bon";
    string str = chuoi.Insert(3, " hai"); //str = "mot hai ba bon"
}
```

➤ **Nhóm phương thức kiểm tra và định vị:** Contain, StartsWith, EndsWith, IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny

- Phương thức Contain: Phương thức thuộc lớp, giúp các tìm một từ hoặc một chuỗi bất kỳ trong một chuỗi khác. Nếu tìm thấy trả về true, không tìm thấy trả về false

Cách sử dụng:

```
bool gt= <chuoi1>.Contain(<chuoi2>)
```

Chuoi2: có thể là một ký tự hoặc 1 chuỗi. Phương thức Contain sẽ kiểm tra chuoi2 có trong chuoi1 không, việc tìm kiếm này có sự phân biệt giữa ký tự hoa và ký tự thường.

Ví dụ 1.29:

```
public static void Main(string []args)
{
    string str = null;
    str = "Diễn đàn công nghệ và giải pháp";
    if (str.Contains("công nghệ") == true)
        MessageBox.Show("Chuỗi trên có chứa từ 'công nghệ' ");
    else
        MessageBox.Show("Chuỗi trên không chứa từ 'công nghệ'");
    //Hiển thị Messagebox: Chuỗi trên có chứa từ 'công nghệ'
```

- Phương thức StartsWith: Phương thức thuộc lớp, kiểm tra xem chuỗi có bắt đầu bởi 1 số ký tự nào đó không. Phương thức này có kiểu trả về là Boolean: true/false.

```
bool gt= <chuoi1>.StartsWith(<chuoi2>)
```

Chuoi2: Là chuỗi cần kiểm tra xem có là chuỗi bắt đầu của chuoi1 không. Việc kiểm tra này có phân biệt ký tự hoa và ký tự thường.

Ví dụ 1.30:

```
public static void Main(string []args)
{
    string str = null;
    str = "Diễn đàn công nghệ và giải pháp";
    if (str.StartsWith("diễn đàn") == true)
        MessageBox.Show("Chuỗi str bắt đầu với 'diễn đàn'");
    else
        MessageBox.Show("Chuỗi str không bắt đầu với 'diễn
đàn'");
}//Hiển thị Messagebox: Chuỗi str không bắt đầu với 'diễn đàn'
```

Nếu muốn việc tìm kiếm không phân biệt ký tự hoa và ký tự thường cần thêm tham số StringComparisonType trong phương thức:

Cách sử dụng:

```
bool gt= <chuoi1>.StartsWith(<chuoi2>, [StringComparisonType])
```

Ví dụ 1.31:

```
public static void Main(string []args)
{
    string str = null;
    str = "Diễn đàn công nghệ và giải pháp";
    if (str.StartsWith("diễn đàn", StringComparison.OrdinalIgnoreCase) ==
true)
        MessageBox.Show("Chuỗi str bắt đầu với 'diễn đàn'");
    else
        MessageBox.Show("Chuỗi str không bắt đầu với 'diễn đàn'");
}//Hiển thị Messagebox: Chuỗi str bắt đầu với 'diễn đàn'
```

- Phương thức EndsWith: Phương thức thuộc lớp, kiểm tra xem chuỗi có kết thúc bởi 1 số ký tự nào đó không. Phương thức này có kiểu trả về là Boolean: true/false.

```
bool gt= <chuoi1>.EndsWith(<chuoi2>)
```

Chuoi2: Là chuỗi cần kiểm tra xem có là chuỗi kết thúc của chuoi1 không. Việc kiểm tra này có phân biệt ký tự hoa và ký tự thường.

Ví dụ 1.32:

```
public static void Main(string []args)
{
    string str = null;
    str = "Diễn đàn công nghệ và giải pháp";
    if (str.EndsWith("giải pháp") == true)
        MessageBox.Show("Chuỗi str kết thúc với 'giải pháp'");
    else
        MessageBox.Show("Chuỗi str không kết thúc với 'giải pháp'");
}//Hiển thị Messagebox: Chuỗi str kết thúc với 'giải pháp'
```

- Phương thức IndexOf: Phương thức thuộc lớp, trả về chỉ số của ký tự đầu tiên mà chuỗi xuất hiện, nếu không tìm thấy sẽ trả giá trị -1.

Cách sử dụng:

```
int vitri= <chuoi1>.IndexOf(<chuoi2>)
```

Chuoi2: là chuỗi cần tìm kiếm trong chuoi1.

Ví dụ 1.33:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay ngay moi";
    string str2 = "ngay";
    int vitri = str1.IndexOf(str2));
}
//vitri = 5
```

- Phương thức LastIndexOf: Phương thức thuộc lớp, tìm kiếm bắt đầu từ cuối chuỗi về đầu, trả về chỉ số của ký tự đầu tiên mà chuỗi xuất hiện. Nếu không tìm thấy sẽ trả giá trị -1.

Cách sử dụng:

```
int vitri= <chuoi1>.LastIndexOf(<chuoi2>)
```

Chuoi2: là chuỗi cần tìm kiếm trong chuoi1.

Ví dụ 1.34:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay ngay moi";
    string str2 = "ngay";
    int vitri = str1.LastIndexOf(str2));
}
//vitri = 10
```

- Phương thức IndexOfAny: Phương thức thuộc lớp, tìm kiếm chỉ số xuất hiện của từng ký tự trong chuỗi so với mảng ký tự đầu vào, nếu không tìm thấy sẽ trả giá trị -1.

Cách sử dụng:

```
int vitri= <chuoi1>.IndexOf(<mảng các ký tự>)
```

Từng ký tự trong chuoi1, bắt đầu từ ký tự đầu tiên của chuoi1 sẽ lần lượt được so sánh với các ký tự trong mảng ký tự đầu vào, nếu tìm thấy một ký tự nào đó trong chuoi1 xuất hiện trong mảng ký tự sẽ trả về vị trí của ký tự đó trong chuoi1.

Ví dụ 1.35:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay ngay ngay moi";
    char[] mangkytu = {'n','g','a','y'};
    int vitri = str1.IndexOfAny(mangkytu); //vitri = 2
}
```

- Phương thức LastIndexOfAny: Phương thức thuộc lớp, tìm kiếm chỉ số xuất hiện của từng ký tự trong chuỗi so với mảng ký tự đầu vào, nếu không tìm thấy sẽ trả giá trị -1.

Cách sử dụng:

```
int vitri= <chuoi1>.LastIndexOf(<mảng các ký tự>)
```

Từng ký tự trong chuoi1, bắt đầu từ ký tự cuối chuoi1 sẽ lần lượt được so sánh với các ký tự trong mảng ký tự đầu vào, nếu tìm thấy một ký tự nào đó trong chuoi1 xuất hiện trong mảng ký tự sẽ trả về vị trí của ký tự đó trong chuoi1.

Ví dụ 1.36:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay ngay ngay moi";
    char[] mangkytu = {'n','g','a','y'};
    int vitri = str1.LastIndexOfAny(mangkytu); //vitri = 18
}
```

## ➤ Nhóm phương thức trích chọn xâu con từ chuỗi: Substring, Split.

- Phương thức Substring: Trích một chuỗi con từ một chuỗi có sẵn, trả về một chuỗi mới.

Cách sử dụng:

```
string kq = <chuoi1>.Substring(start, len);
```

start: Vị trí ký tự bắt đầu trích trong chuoi1

len: Số ký tự sẽ trích ra trong chuoi1

Ví dụ 1.37:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay moi";
    string str=str1.Substring(5, 4);
    //str = "ngay"
}
```

Ngoài ra có thể trích chọn chuỗi con từ vị trí chỉ định đến cuối chuỗi bằng cách bỏ đi tham số len trong phương thức Substring.

Cách sử dụng:

```
string kq = <chuoi1>.Substring(start);
```

start: Vị trí ký tự bắt đầu trích trong chuoi1

Ví dụ 1.38:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay moi";
    string str=str1.Substring(5);
    //str = "ngay moi"
}
```

- Phương thức Split: Phương thức thành viên của lớp. Sử dụng tách chuỗi lớn thành các chuỗi con.

Cách sử dụng:

```
string []kq = <chuoi1>.Substring(<danh sách ký tự>);
```

<danh sách ký tự>: Là một ký tự hoặc một mảng các ký tự sử dụng để làm tiêu chí tách chuoi1 thành các chuỗi con.

Ví dụ 1.39:

```
public static void Main(string []args)
{
    string str1 = "Son,Tung,Tuan";
    string[] ten = str1.Split(',');
}
//Kết quả được mảng tên gồm các phần tử: {"Son", "Tung", "Tuan"}
```

- Phương thức Remove: Phương thức thuộc lớp, sử dụng để xóa chuỗi

Cách sử dụng:

```
string kq = <chuoi1>.Remove(<Vị trí>, [<Số ký tự>]);
```

<Vị trí>: Là vị trí của ký tự trong chuoi1 bắt đầu xóa.

<Số ký tự>: Là số lượng ký tự sẽ xóa trong chuoi1

Ví dụ 1.40:

```
public static void Main(string []args)
{
    string str = "mothaibabonnam";
    string str1 = str.Remove(3,8);
}
//str1 = "motnam"
```

Ngoài ra, Remove còn có thể sử dụng để xóa từ một vị trí chỉ định đến cuối chuỗi.

Cách sử dụng:

```
string kq = <chuoi1>.Remove(<Vị trí>);
```

<Vị trí>: Vị trí ký tự bắt đầu xóa trong chuoi1.

Ví dụ 1.41:

```
public static void Main(string []args)
{
    string str = "mothaibabonnam";
    string str1 = str.Remove(8);
}
//str1 = "motnamba"
```

➤ **Nhóm phương thức sao chép chuỗi:** Copy, ToCharArray, CopyTo

- Phương thức Copy: Phương thức tĩnh giúp trả về 1 chuỗi nằm trên 1 vùng nhớ khác nhưng cùng nội dung với chuỗi ban đầu.

Cách sử dụng:

```
string kq = String.Copy(<chuoi>);
```

Ví dụ 1.42:

```
public static void Main(string []args)
{
    string str1 = "Chao ngay moi";
    string str = String.Copy(str1);
}
//Tạo ra vùng nhớ mới của str1 đồng thời sao chép nội dung chuỗi str
//vào str1. Kết quả str = "Chao ngay moi"
```

- Phương thức ToCharArray: Phương thức thuộc lớp, chuyển 1 chuỗi về 1 mảng ký tự và trả giá trị về là mảng ký tự đó.

Cách sử dụng:

```
char[] <tên mảng ký tự> = <chuoi>.ToCharArray();
```

<chuoi>: Là chuỗi ký tự cần chuyển đổi về mảng các ký tự.

<Tên mảng ký tự>: chứa các ký tự chuyển từ <chuoi>

Ví dụ 1.43:

```
public static void Main(string []args)
{
    string str1 = "Chao";
    char []chars =str1.ToCharArray();
}
//mảng ký tự chars = {'C', 'h', 'a', 'o'}
```

- Phương thức CopyTo: Phương thức thuộc lớp, giúp sao chép các ký tự trong một chuỗi vào một mảng ký tự, có thể quy định được vị trí và số lượng ký tự trong chuỗi sẽ copy vào mảng ký tự, đồng thời chỉ định copy vào vị trí thứ mấy trong mảng ký tự.

Cách sử dụng:

```
void CopyTo(int sourceIndex, char[] destination, int destinationindex,
           int count)
```

Ví dụ 1.44:

```
public static void Main(string []args)
{
    string str= “chao mung mot ngay moi”;
    char []chars = new char[10];
    str.CopyTo( 5, chars, 0, 8);
}
//mảng ký tự chars = {'m', 'u', 'n', 'g', ' ', 'm', 'o', 't'}
```

➤ Nhóm các phương thức định dạng chuỗi: PadLeft, PadRight, ToLower, ToUpper, Trim, TrimStart, TrimEnd.

- Phương thức PadLeft: Phương thức thuộc lớp, canh lề phải đồng thời thêm vào bên trái chuỗi một số ký tự đặc biệt.

Cách sử dụng:

```
string <chuoi> = <chuoi> . PadLeft(len, [ký tự])
```

[ký tự]: Ký tự sẽ thêm, nếu trong hàm PadLeft không có tham số [ký tự] thì mặc định sẽ thêm khoảng trắng vào.

Ví dụ 1.45:

```
public static void Main(string []args)
{
    string str1 = "Chao";
    str1= str1.PadLeft(7);      //str1 = “ Chao”
    string str2="hello";
    str2=str2.PadLeft(8, '*'); //str2 = “***hello”
}
```

- Phương thức PadRight: Phương thức thuộc lớp, canh lề trái đồng thời thêm vào bên phải chuỗi một số ký tự đặc biệt.

Cách sử dụng:

```
string <chuoi> = <chuoi> . PadRight(len, [ký tự])
```

[ký tự]: Ký tự sẽ thêm, nếu trong hàm PadRight không có tham số [ký tự] thì mặc định sẽ thêm khoảng trắng vào.

Ví dụ 1.46:

```
public static void Main(string []args)
{
    string str1 = "Chao";
    str1= str1.PadRight(7);      //str1 = “Chao ”
    string str2="hello";
    str2=str2.PadRight(8, '*'); //str2 = “hello***”
}
```

- Phương thức ToUpper: Phương thức thuộc lớp, trả về một chuỗi in hoa

Cách sử dụng:

```
string <chuoi1> = <chuoi2> . ToUpper()
```

Ví dụ 1.47:

```
public static void Main(string []args)
{
    string str1 = "Chao";
    string str2= str1.ToUpper(); //str2 = “CHAO”
}
```

- Phương thức ToLower: Phương thức thuộc lớp, trả về một chuỗi in thường

Cách sử dụng:

```
string <chuoi1> = <chuoi2> . ToLower()
```

Ví dụ 1.48:

```
public static void Main(string []args)
{
    string str1 = "Chao";
    string str2= str1.ToLower(); //str2 = “chao”
}
```

- Phương thức Trim: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng đầu chuỗi và cuối chuỗi.

Cách sử dụng:

```
string <chuoi1> = <chuoi2> . Trim()
```

Ví dụ 1.49:

```
public static void Main(string []args)
{
    string str1 = " Chao ngay moi ";
    string str2= str1.Trim(); //str2 = “Chao ngay moi”
}
```

- Phương thức TrimStart: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng đầu chuỗi.

Cách sử dụng:

```
string <chuoi1> = <chuoi2> . TrimStart()
```

Ví dụ 1.50:

```
public static void Main(string []args)
{
    string str1 = " Chao ngay moi ";
    string str2= str1.TrimStart(); //str2 = “Chao ngay moi ”
}
```

- Phương thức TrimEnd: Phương thức thuộc lớp, trả về 1 chuỗi đã bỏ khoảng trắng cuối chuỗi.

Cách sử dụng:

```
string <chuoi1> = <chuoi2> . TrimEnd()
```

Ví dụ 1.51:

```
public static void Main(string []args)
{
    string str1 = " Chao ngay moi ";
    string str2= str1.TrimEnd(); //str2 = “ Chao ngay moi ”
}
```

## 1.8. Mảng

Mảng là một tập hợp có thứ tự của những đối tượng, các đối tượng này cùng một kiểu. Cú pháp khai báo mảng được kết hợp giữa cú pháp khai báo mảng của C và định nghĩa lớp của C#, do đó các đối tượng của mảng có thể truy cập những phương thức và thuộc tính của lớp System.Array.

### 1.8.1. Mảng một chiều

➤ Cú pháp:

```
<kiểu dữ liệu> [] <tên mảng> = new <kiểu dữ liệu>[<số pt>];
```

Ví dụ 1.52: Khai báo mảng số nguyên có 100 phần tử

```
int [] mang = new int[100];
```

Hoặc có thể khai báo mảng đồng thời khởi tạo giá trị cho các phần tử mang:

➤ Cú pháp:

```
<kiểu dữ liệu> [] <tên mảng> = {<gt1>, <gt2>, <gt3>, ...., <gtn>};
```

Lưu ý: khi khởi tạo mảng bằng toán tử new thì các phần tử của mảng sẽ mang giá trị mặc định như bảng 1.5 sau:

Bảng 1.5: Bảng mô tả các giá trị mặc định của kiểu dữ liệu

Kiểu dữ liệu	Giá trị mặc định
int, long, byte, ...	0
bool	false
char	'\0' (null)
enum	0
reference	null

➤ Truy cập các phần tử của mảng:

Để truy cập các thành phần trong mảng phải sử dụng toán tử chỉ mục []. Với phần tử đầu tiên trong mảng mang chỉ số số 0.

Ngoài ra C# hỗ trợ thuộc tính Length để xác định chiều dài của mang.

Ví dụ 1.53: Tính tổng các phần tử của mảng A = {1, 2, 3, 4, 5}

```
public static void Main(string []args)
{
    int[] mang={1,2,3,4,5,6};
    int tong = 0;
    for(int i=0; i<mang.Length; i++) {
        tong = tong + mang[i];
    }
}
```

### 1.8.2. Mảng hai chiều

Trong C#, ngoài việc khai báo mảng một chiều như mục 1.8.1 thì lập trình viên có thể khai mảng hai chiều hoặc nhiều chiều tùy thuộc vào mục đích sử dụng. Và mỗi mảng khi khai báo lại có thể khai báo theo hai dạng khác nhau là mảng đa chiều cùng kích thước hoặc mảng đa chiều không cùng kích thước.

➤ Mảng hai chiều cùng kích thước: Là mảng được tổ chức thành các dòng và các cột, trong đó dòng được tính theo hàng ngang và cột được tính theo hàng dọc của mảng. Trong mảng hai chiều cùng kích thước thì số lượng phần tử trên mỗi dòng là bằng nhau.

- Cú pháp:

```
<kieu du lieu> [,] <ten mang> = new <kieu du lieu>[<dòng>,<cột>];
```

Ví dụ 1.54: Khai báo mảng hai chiều cùng kích thước kiểu số nguyên có 3 dòng 4 cột.

```
int [,] M = new int[3,4];
```

4 cột			
M[0,0]	M[0,1]	M[0,2]	M[0,3]
M[1,0]	M[1,1]	M[1,2]	M[1,3]
M[2,0]	M[2,1]	M[2,2]	M[2,3]

Ví dụ 1.55: Khai báo mảng M có kích thước 2 dòng, 3 cột với các phần tử tương ứng như sau:

1	2	3
4	5	6

Tính tổng các phần tử trong mảng M

```
public static void Main(string []args)
{
    int[,] M = new int[2,3];
    int tong = 0, i, j, k=1;
    for(i=0; i<2; i++)
        for(j=0; j<3; j++) {
            M[i, j] = k;
            k = k+1;
        }
    for(i=0; i<2; i++)
        for(j=0; j<3; j++)
            tong = tong + M[i, j];
}
```

- Mảng hai chiều không cùng kích thước: Là mảng được tổ chức thành các dòng và các cột, trong đó dòng được tính theo hàng ngang và cột được tính theo hàng dọc của mảng. Trong mảng hai chiều không cùng kích thước thì số lượng phần tử trên mỗi dòng có thể khác nhau.

- Cú pháp:

Khi khai báo mảng hai chiều cùng kích thước phải khai báo số dòng của mảng trước

```
<kieu du lieu> [][] <Ten mang> = new <kieu du lieu>[<dòng>][];
```

Sau khi đã khởi tạo số dòng của mảng, cần khai báo số cột trên từng dòng của mảng:

```
<ten mang> [0] = new <kieu du lieu>[<Số cột 0>];
<ten mang> [1] = new <kieu du lieu>[<Số cột 1>];
.....
<ten mang> [dòng] = new <kieu du lieu>[< Số cột n>];
```

Ví dụ 1.56: Khai báo mảng hai chiều có 3 dòng, dòng 1 có 3 cột, dòng hai có 4 cột, dòng 3 có 5 cột.

Gán phần tử vị trí dòng 0 và cột 1 giá trị 10

Gán phần tử vị trí dòng 1 và cột 0 giá trị 5

Gán phần tử vị trí dòng 2 và cột 4 giá trị 15

```
public static void Main(string []args)
{
    int[][] M = new int[3][];
    M[0] = new int[3];
    M[1] = new int[4];
    M[2] = new int[5];
    M[0][1] = 10;
    M[1][0] = 5;
    M[2][4] = 15;
}
```

0	10	0		
5	0	0	0	
0	0	0	0	15

## 1.9. Tạo và sử dụng DLL trong C#

DLL là từ viết tắt của Dynamic linking library, gọi là thư viện liên kết động, là các module chứa các phương thức và dữ liệu.các phương thức và dữ liệu trong Module này sẽ được đóng gói vào 1 tập tin có đuôi .DLL.

### 1.9.1. Ưu điểm và nhược điểm khi sử dụng DLL

Một số ưu điểm khi sử dụng DLL:

- Giảm không gian sử dụng bộ nhớ: Điều này có nghĩa là sử dụng DLL sẽ làm giảm kích thước của ứng dụng: Do mã lệnh của các hàm trong DLL sẽ không được nhúng vào trong file chương trình của ứng dụng. Khi đó ứng dụng chỉ cần lưu thông tin của hàm trong DLL, và khi cần hệ điều hành sẽ tải các hàm này vào bộ nhớ để ứng dụng sử dụng. Khi không còn sử dụng, có thể giải phóng DLL khỏi bộ nhớ, khi cần nâng cấp, chỉ cần thay thế file DLL, các file chương trình khác không bị ảnh hưởng
- Tạo ra khả năng tương tác giữa các ngôn ngữ lập trình: Một ứng dụng có thể sử dụng các DLL viết bằng bất cứ ngôn ngữ lập trình nào. Các nhà phát triển phần mềm chỉ việc đóng gói các module của mình vào trong một DLL với ngôn ngữ ưa thích, sau đó module này có thể được sử dụng trong các ứng dụng viết bằng C++ hay Visual Basic

*Một số nhược điểm khi sử dụng DLL:*

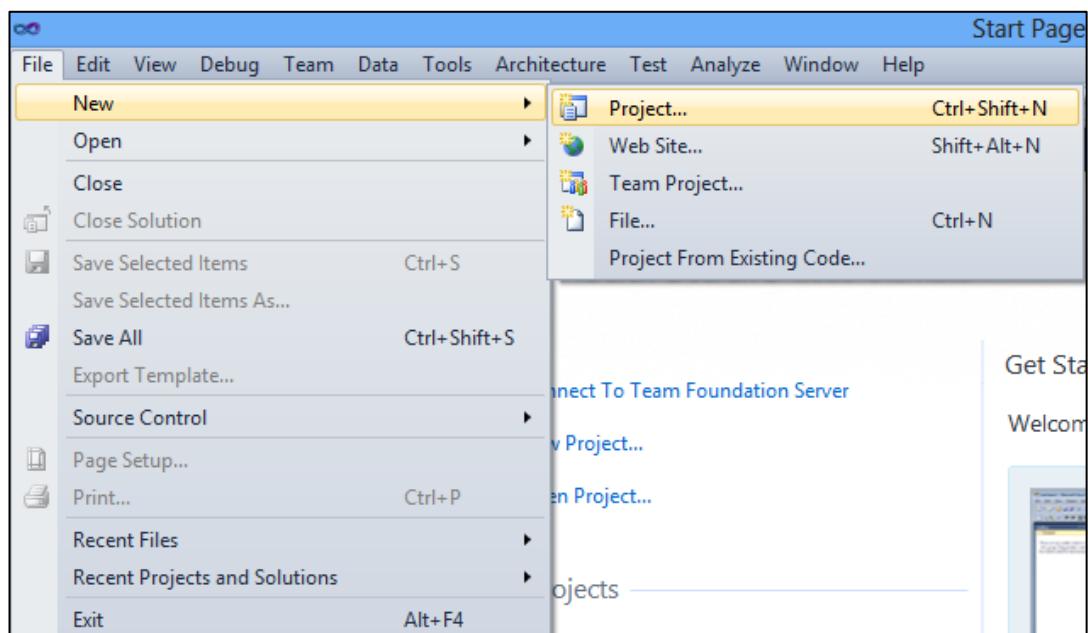
Tuy nhiên khi sử dụng DLL thỉnh thoảng sẽ gặp một lỗi sau:

The ordinal abc could not be located in the dynamic-link library xyz.dll

Lỗi này phát sinh do chương trình cài đặt không kiểm tra phiên bản của các DLL trước khi sao lưu nó vào trong thư mục hệ thống. Khi một DLL mới thay thế một DLL cũ có sẵn, và nếu DLL mới này có một số thay đổi lớn làm cho nó không thể tương thích ngược lại với các chương trình sử dụng phiên bản cũ, nó sẽ làm rối loạn chương trình đó.

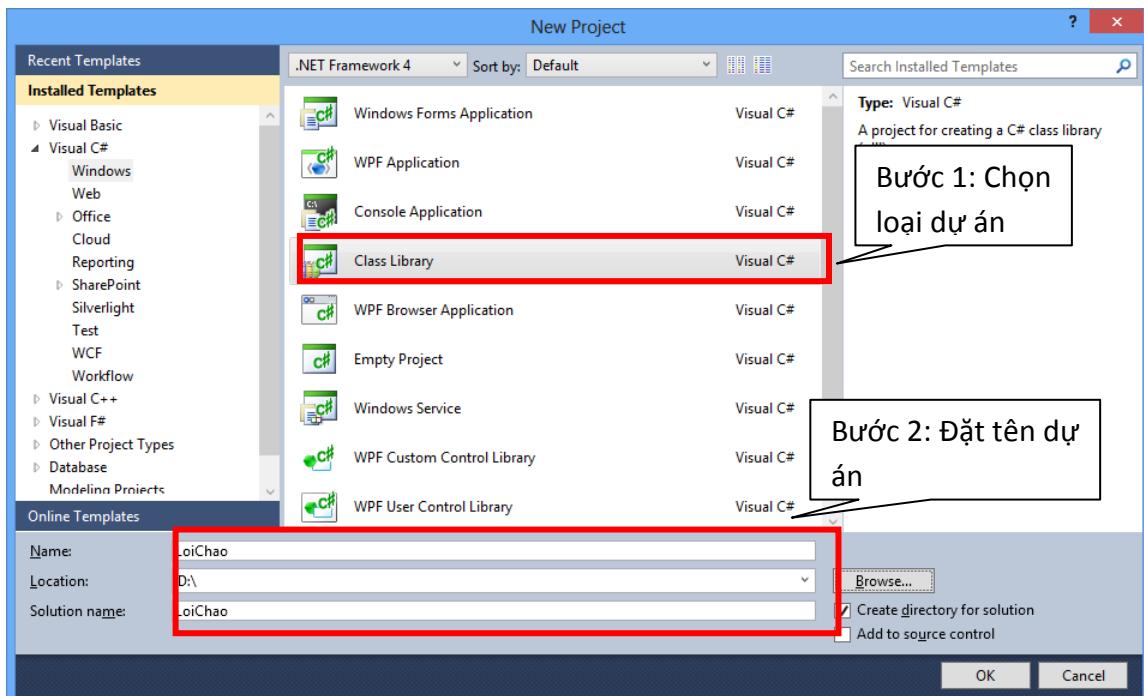
### 1.9.2. Các bước để tạo tập tin DLL

Bước 1: Tạo mới dự án: Chọn thực đơn File > new > Project như hình 1.8.



Hình 1.8: Tạo mới dự án

Bước 2: Chọn loại dự án là Class Library và đặt tên “LoiChao” cho tập tin DLL.



Hình 1.9: Chọn loại dự án và đặt tên dự án

Bước 3: Giao diện soạn thảo mã lệnh được hiển thị, tiến hành viết mã lệnh như hình 1.10.

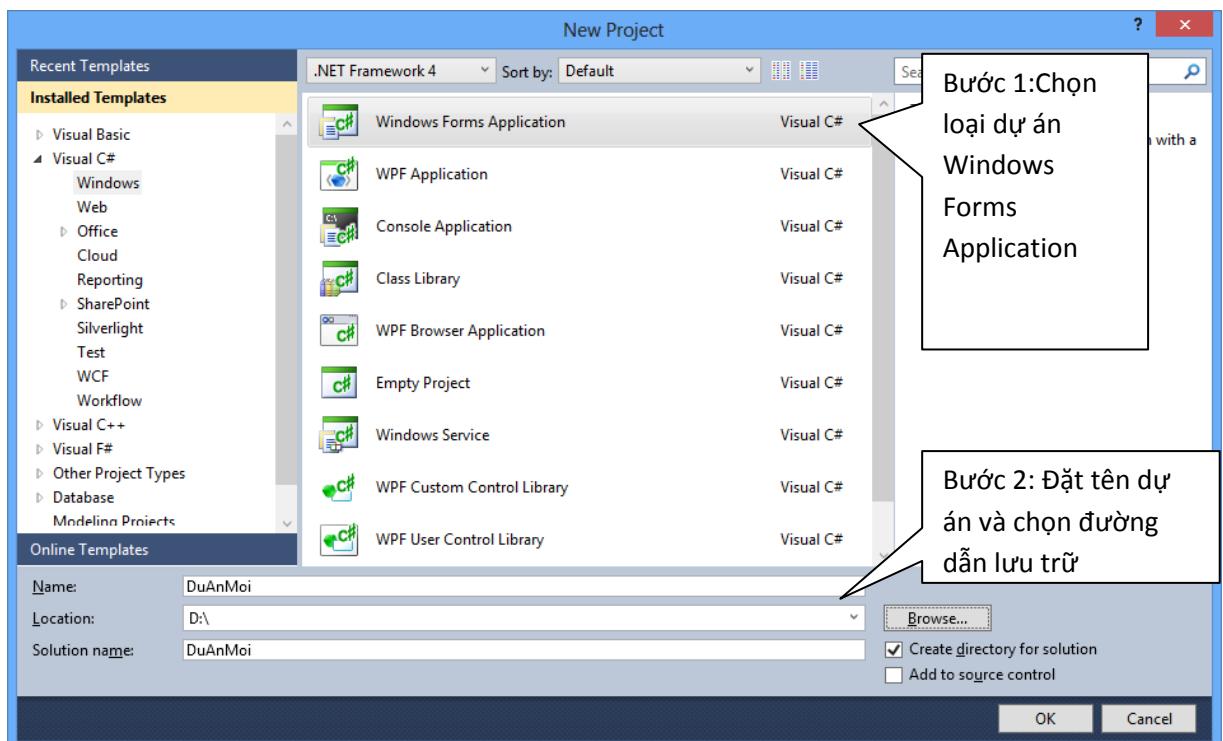
```
Class1.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace LoiChao
{
    public class LoiNoiDau
    {
        public void LoiDauTien()
        {
            MessageBox.Show("Chào mừng đến với ngôn ngữ C#");
        }
        public void LoiKetThuc()
        {
            MessageBox.Show("Tạm biệt, hẹn gặp lại");
        }
    }
}
```

Hình 1.10: Giao diện soạn thảo mã lệnh

Bước 4: Nhấn F5 biên dịch để dự án tạo tập tin DLL. Thông thường tập tin DLL được tạo sẽ nằm trong thư mục Bin của dự án → hoàn thành công việc tạo tập tin DLL

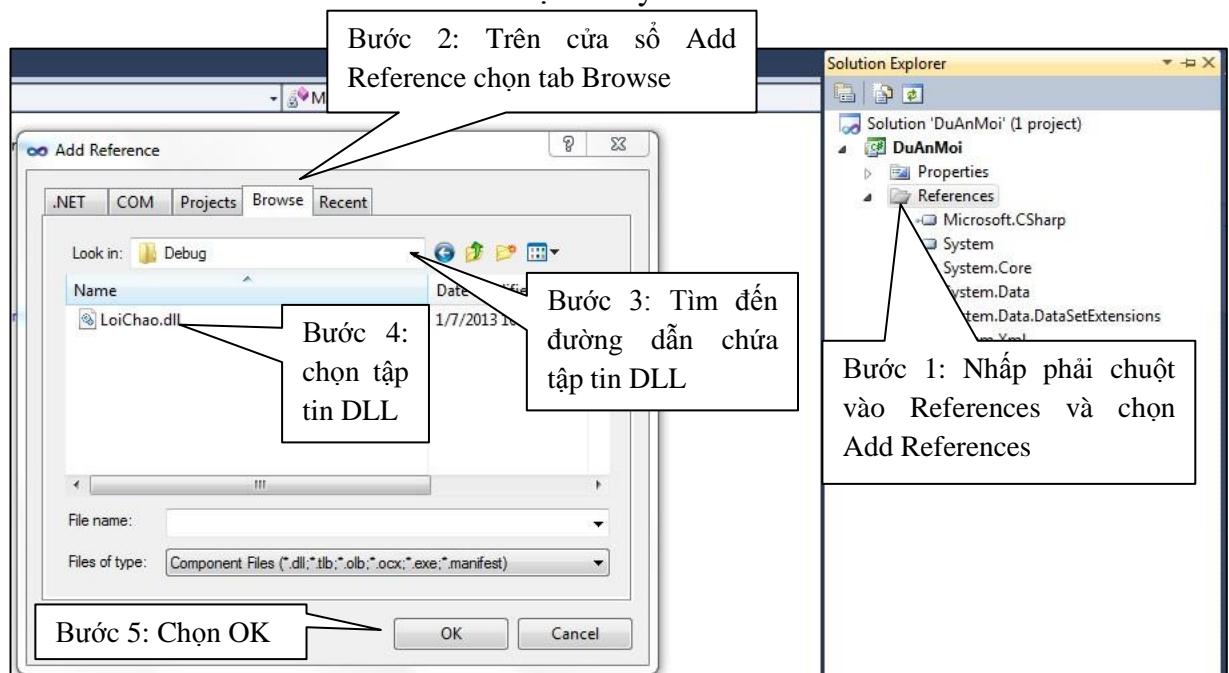
### 1.9.3. Các bước để sử dụng tập tin DLL

Bước 1: Tạo dự án mới như hình 1.11.



Hình 1.11: Tạo dự án mới

Bước 2: Bên cửa sổ Solution Explorer, nhấp phải chuột vào References và chọn Add References để thêm \*.DLL vào dự án này như hình 1.12.



Hình 1.12: Thêm tập tin DLL vào dự án

Bước 3: Sử dụng tập tin DLL.

```

Form1.cs [Design] Form1.cs [Design]
DuanMoi.Form1
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using LoiChao;
namespace DuanMoi
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

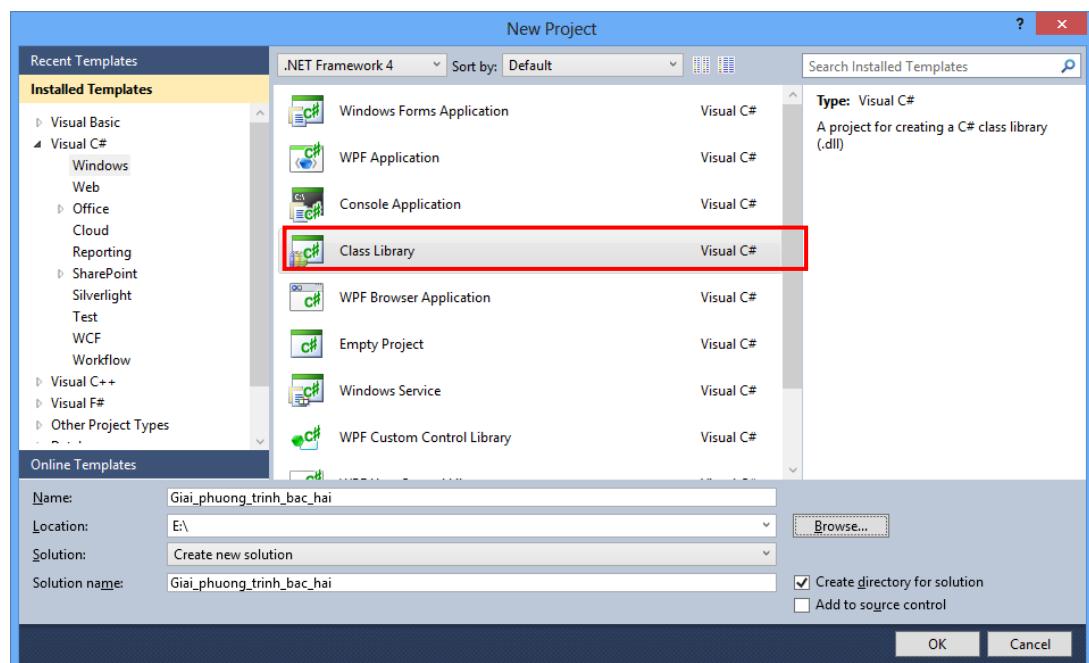
        private void Form1_Load(object sender, EventArgs e)
        {
            LoiNoiDau obj = new LoiNoiDau();
            obj.LoiDauTien();
            obj.LoiKetThuc();
        }
    }
}

```

Hình 1.13: Sử dụng DLL

Ví dụ: Xây dựng tập tin DLL hỗ trợ giải phương trình bậc hai.

Bước 1: Tao dự án loại Class Library và đặt tên là Giai\_phuong\_trinh\_bac\_hai như hình 1.14.



Hình 1.14: Tạo dự án Class Library

## Bước 2: Viết mã lệnh cho điều khiển

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Giai_phuong_trinh_bac_hai
{
    public class PhuongTrinhBacHai
    {
        int a, b, c;
        public int C {
            get { return c; } set { c = value; }
        }
        public int B {
            get { return b; } set { b = value; }
        }
        public int A {
            get { return a; } set { a = value; }
        }
        public string TimNghiem()
        {
            string ketqua = "";
            double x1=0, x2=0;
            double delta = b * b - 4 * a * c;
            if (a != 0)
            {
                if (delta < 0)
                    ketqua = "Phương trình vô nghiệm";
                else
                    if (delta == 0)
                    {
                        x1 = -b / (2.0 * a);
                        ketqua = "Phương trình có nghiệm kép: " + x1;
                    }
                    else
                    {
                        x1 = (-b - Math.Sqrt(delta)) / (2.0 * a);
                        x2 = (-b + Math.Sqrt(delta)) / (2.0 * a);
                        ketqua = "x1 = " + x1 + " và x2 = " + x2;
                    }
                }
                else
                    ketqua = "Đây không phải là phương trình bậc hai";
            return ketqua;
        }
    }
}
```

## **1.10. Bài tập cuối chương**

Bài 1: Tạo tập tin thư viện tên ToaDo.DLL, hỗ trợ cộng và trừ hai tọa độ

Bài 2: Tạo tập tin thư viện tên PhanSo.DLL, hỗ trợ cộng, trừ, nhân, chia hai phân số

Bài 3: Tạo tập tin thư viện tên PTB1.DLL, hỗ trợ người sử dụng giải phương trình bậc nhất:  $ax + b = 0$

Bài 4: Tạo tập tin thư viện tênPTB4.DLL, hỗ trợ người sử dụng giải phương trình trùng phương:  $ax^4 + bx^2 + c = 0$

Bài 5: Tạo tập tin thư viện tên HePTB1.DLL, hỗ trợ người sử dụng giải hệ phương trình bậc nhất hai ẩn số:

$$\begin{cases} ax + by = c \\ a_1x + b_1y = c_1 \end{cases}$$

## CHƯƠNG 2: GIỚI THIỆU VỀ WINDOWS FORMS

### 2.1. Giới thiệu

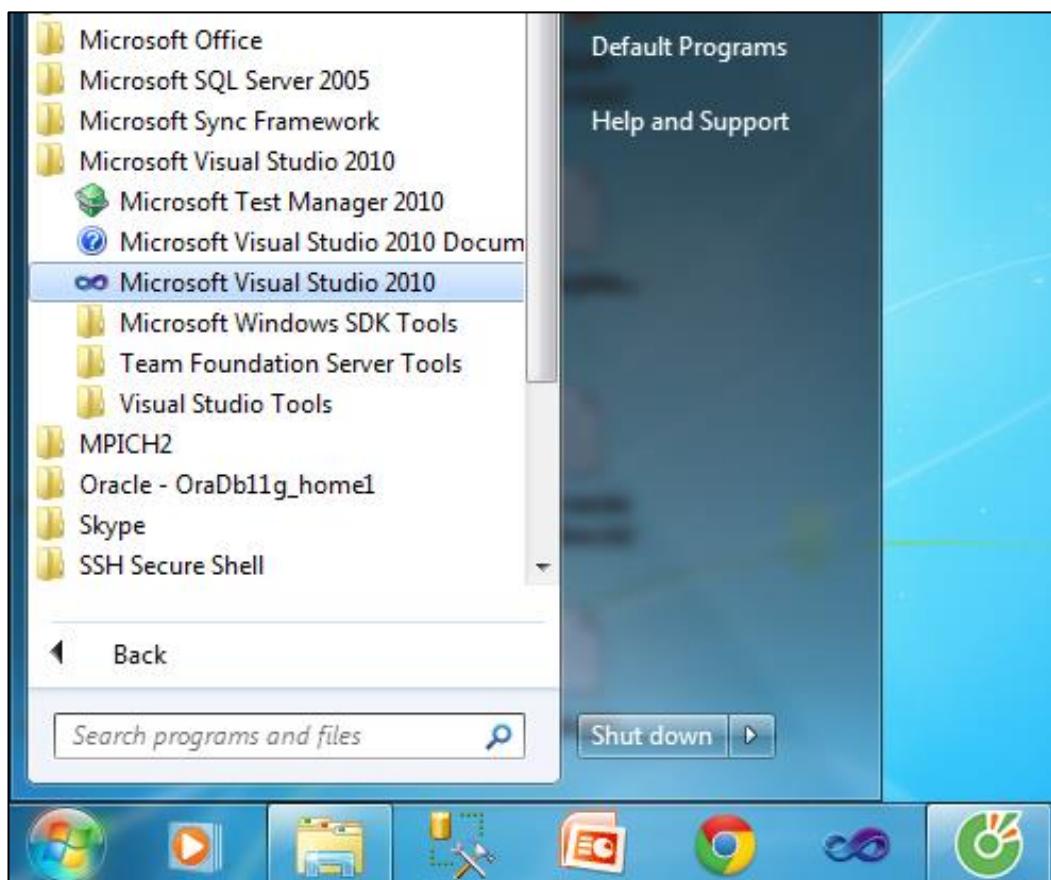
#### 2.1.1. Ứng dụng Windows Forms

Windows Forms là ứng dụng cơ bản của Microsoft; Windows Forms cho phép lập trình viên tạo các form một cách dễ dàng, đồng thời hỗ trợ nhiều điều khiển (Controls) hoặc thành phần (Components) giúp lập trình viên có thể xây dựng, tùy chỉnh các giao diện form một cách nhanh chóng: Tạo ra các form có kích thước và hình dạng khác nhau, bố trí các điều khiển và thành phần trên form phù hợp với nhu cầu, mục đích của người sử dụng.

Ngoài ra, trên mỗi điều khiển thì Windows Forms cung cấp nhiều sự kiện giúp tương tác với thiết bị. Lập trình viên có thể sử dụng một cách đa dạng các sự kiện này trên các thiết bị như: sự kiện ấn nút bàn phím, sự kiện nhập chuột, hoặc sự kiện ấn tổ hợp phím.

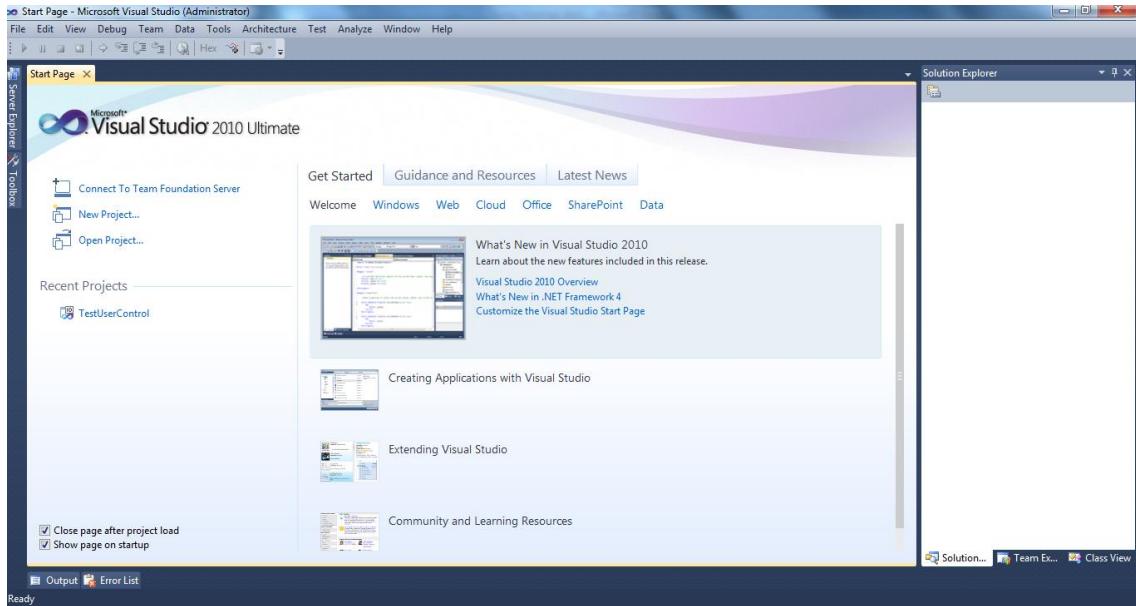
- Tạo một dự án Windows Forms: Để tạo một dự án Windows Forms với C#, cần thực hiện một số bước sau :

Bước 1: Sau khi cài xong Microsoft Visual Studio 2010, tiến hành bật ứng dụng Visual Studio 2010 lên bằng cách nhấp vào biểu tượng như hình 2.1:



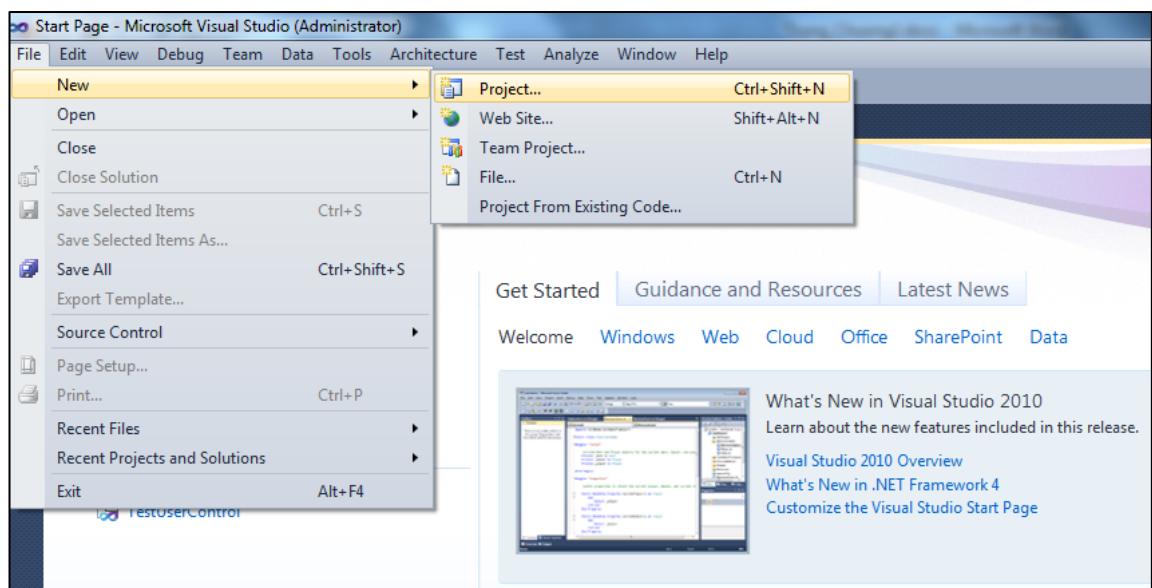
Hình 2.1: Mở ứng dụng Visual Studio 2010

Bước 2: Sau khi đã chọn biểu tượng Visual Studio 2010 giao diện như hình 2.2 sẽ được hiển thị:



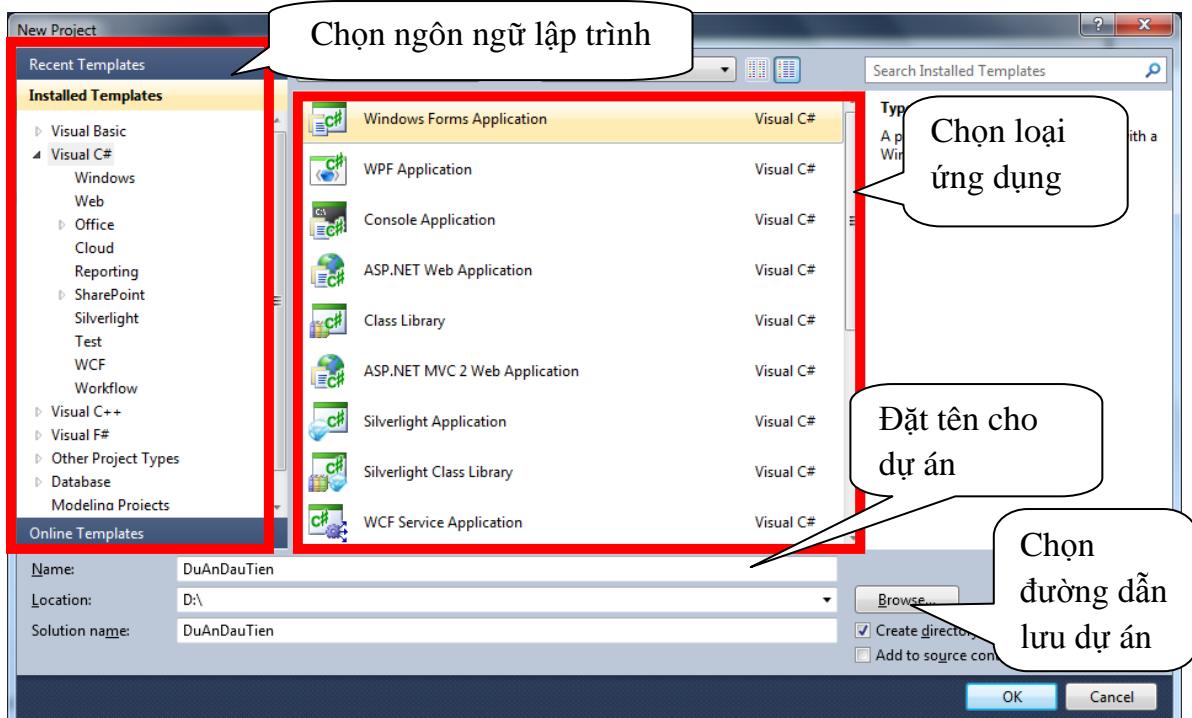
Hình 2.2: Giao diện ứng dụng Visual Studio 2010

Bước 3: Chọn Menu File > New > Project



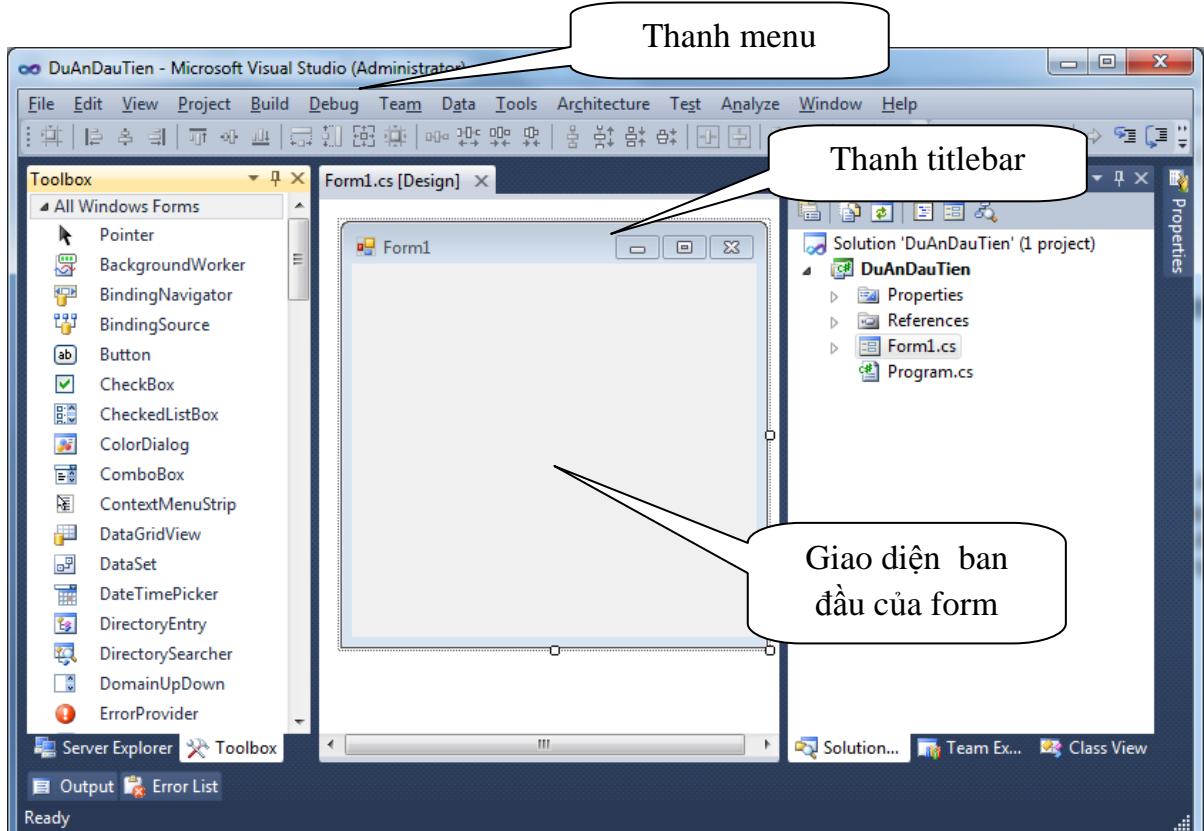
Hình 2.3: Giao diện tạo dự án mới

Bước 4: Sau khi thực hiện xong bước 3, cửa sổ New Project sẽ xuất hiện, tại cửa sổ này lập trình viên có thể chọn ngôn ngữ lập trình sử dụng và chọn loại ứng dụng muốn viết. Cụ thể ở đây chọn ngôn ngữ Visual C#, và loại ứng dụng là Windows Forms. Sau khi đã chọn xong loại ứng dụng đã viết, lập trình viên cần đặt tên cho dự án (nếu không muốn sử dụng tên mặc định của Visual Studio) và đường dẫn sẽ lưu dự án như hình 2.4:



Hình 2.4: Cửa sổ New Project

Sau khi đã tạo xong dự án, một form mới có tên Form1 mặc định sẽ được thêm vào dự án vừa tạo như hình 2.5. Lúc này lập trình viên có thể thiết kế giao diện của Form1 bằng cách thêm các thành phần cần thiết phù hợp với nhu cầu của phần mềm.

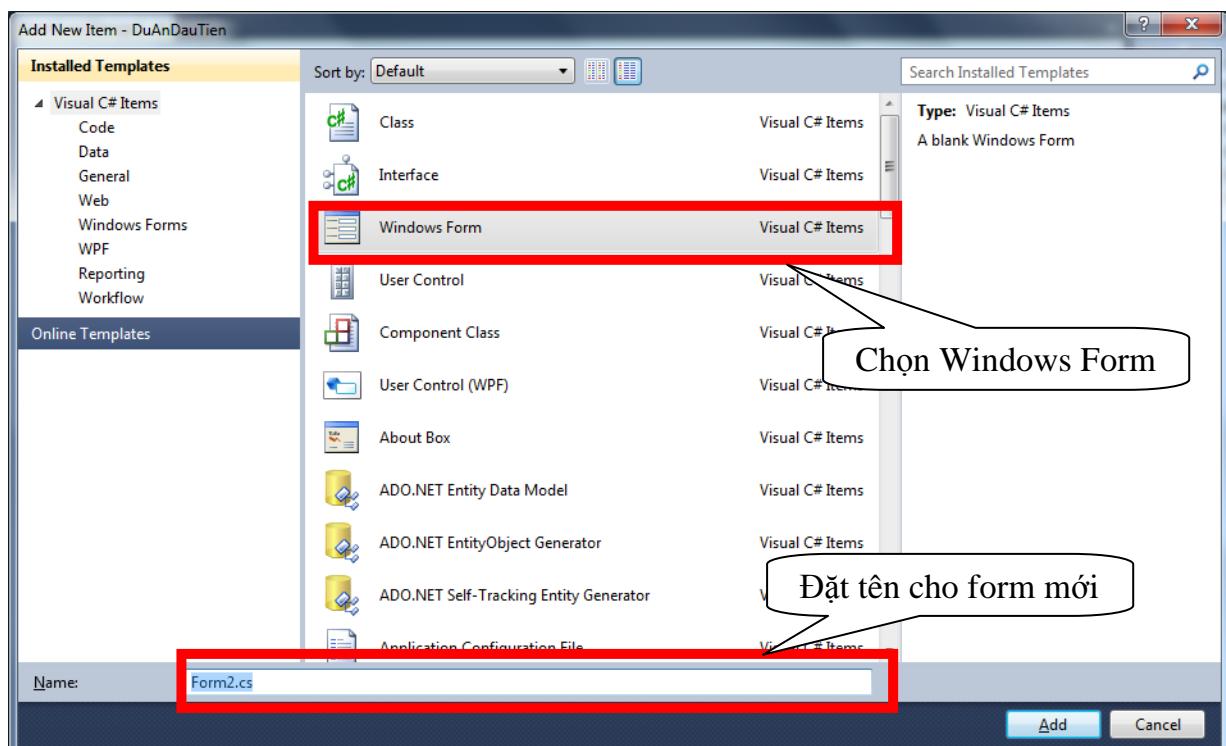


Hình 2.5: Giao diện chính của dự án Windows Forms

➤ Thêm form mới vào dự án:

Phần lớn các dự án phần mềm đều chứa rất nhiều form . Do đó lập trình viên có thể thêm một form mới vào dự án và thiết kế form đó tại thời điểm thiết kế hoặc lập trình viên có thể thêm form mới bằng cách viết các đoạn mã tạo form và form mới sẽ được tạo ra tại thời điểm thực thi của chương trình.

Để thêm một form mới vào dự án tại thời điểm thiết kế, trên menu Project, chọn Add Windows Form. Một hộp thoại Add New Item được mở ra. Sau đó chọn mục Windows Form và đặt tên cho form mới như hình 2.6. Nhập Add để hoàn tất việc thêm form mới vào dự án phần mềm.



Hình 2.6: Cửa sổ Add New Item

Để thêm một Form mới sau khi đã hoàn thành việc thiết kế giao diện, lập trình viên sẽ phải viết các đoạn mã chương trình tạo form và sau đó form mới sẽ được tạo và hiển thị trong quá trình chương trình được thực thi.

Khái báo đối tượng thuộc lớp Form như sau:

```
Form form1;  
form1 = new Form();
```

Hiển thị Form khi tại thời điểm chương trình thực thi cần sử dụng đến phương thức Show() hoặc ShowDialog():

```
form1.Show();
```

### 2.1.2. Không gian tên (namespace)

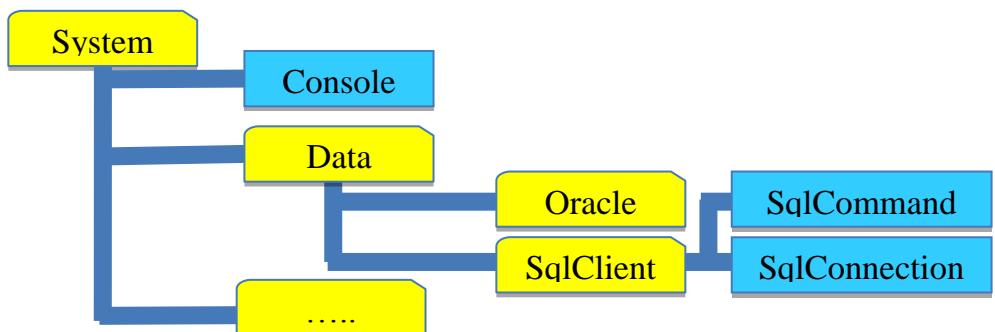
Namespace được gọi là không gian tên. Ý nghĩa lớn của việc sử dụng không gian tên là tránh sự xung đột tên lớp (class) trong một dự án (project). Ngoài ra, không gian tên còn giúp phân nhánh những lớp có cùng chức năng nhằm giúp dễ dàng quản lý mã nguồn. Cụ thể, để làm việc với màn hình Console, C# đã tạo ra một lớp Console, lớp này chứa những thuộc tính và phương thức chỉ dành riêng làm việc với môi trường Console, lớp Console này đặt trong không gian tên System;

Không gian tên tổ chức dạng phân cấp, y như việc phân cấp của cây thư mục. Trong một không gian tên có thể chứa nhiều lớp, các lớp trong cùng một không gian tên không được trùng tên. Hình 2.7 cho thấy sự phân cấp của không gian tên:

Không gian tên System chứa không gian tên Data;

Không gian tên Data chứa không gian tên Oracle và SqlCommand;

Trong không gian tên SqlCommand chứa nhiều lớp khác tên nhau như: lớp SqlCommand, lớp SqlConnection, ...



Hình 2.7: Tổ chức phân cấp của không gian tên trong C#

➤ Sử dụng không gian tên:

C# xây dựng sẵn nhiều lớp đối tượng đặt trong các không gian tên khác nhau phục vụ cho việc lập trình. Để sử dụng không gian tên trong C#, cần sử dụng từ khóa using.

Ví dụ 2.1:

```
using System;  
using System.Windows.Forms;
```

Khi tạo một dự án kiểu Windows Application, mặc định dự án sẽ sử dụng 8 không gian tên sau:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

Ý nghĩa của các không gian tên như sau:

- Không gian tên System: Chứa các lớp và các kiểu dữ liệu cơ sở (int, double, byte, short, ...).
- Không gian tên System.Data: Chứa các lớp của ADO.NET, các lớp giúp kết nối hoặc thao tác với cơ sở dữ liệu (SQL Server, Oracle, Microsoft Access).
- Không gian tên System.Drawing: Chứa các lớp làm việc với đồ họa.
- Không gian tên System.ComponentModel: Chứa các lớp cơ sở và giao diện giúp thực thi chuyển đổi kiểu, thuộc tính, liên kết với các dữ liệu nguồn và các thành phần cấp phép
- Không gian tên System.Collections.Generic: Chứa các lớp hỗ trợ cho việc thiết lập các tập hợp dạng chung mẫu (template) như:
  - ArrayList List<T>: Tập hợp chung mẫu có thể thay đổi kích thước khi vận hành.
  - Queue Queue<T>: Tập hợp chung mẫu theo mô hình vào trước ra trước(FIFO).
  - Stack Stack<T>: Tập hợp chung mẫu theo mô hình vào sau ra trước (LIFO)
  - ...
- Không gian tên System.Text: Cung cấp các lớp giúp mã hóa các ký tự Ascii hoặc Unicode.
- Không gian tên System.Linq: Cung cấp các lớp hỗ trợ cho việc sử dụng ngôn ngữ truy vấn tích hợp (Language-Integrated Query – LinQ).
- Không gian tên System.Windows.Forms: Là không gian tên chính cung cấp các lớp để xây dựng ứng dụng kiểu Windows Application. Không gian tên System.Windows.Forms chia thành các nhóm sau:
  - Control, UserControl, Form
  - Menu, toolbar: ToolStrip, ToolStrip, ContextMenuStrip, StatusStrip
  - Controls: TextBox, ComboBox, ListBox, ListView, WebBrowser, Label, ...
  - Layout: FlowLayoutPanel, TableLayoutPanel, ... Giúp trình bày các điều khiển trên form.

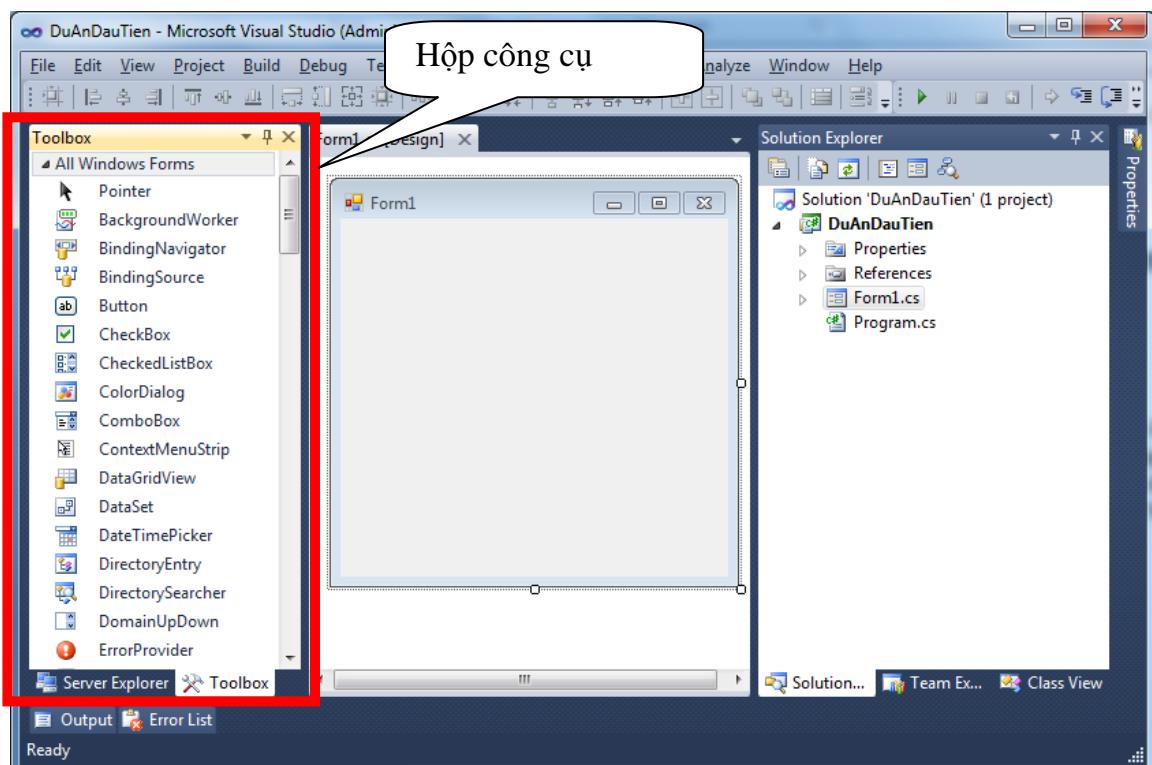
- Data và DataBinding: Gồm các công cụ BindingNavigator, BindingSource, ... giúp liên kết dữ liệu từ cơ sở dữ liệu đến các điều khiển Dataset hoặc DataGridView.
- Command Dialog Boxes: Gồm các điều khiển OpenFileDialog, SaveFileDialog, ColorDialog, FontDialog, PrintDialog làm việc với tập tin, màu sắc, phông chữ, in ấn.

### 2.1.3. Thanh công cụ (Toolbox)

Cung cấp danh sách các Component/ Control được liệt kê theo nhóm, cho phép lập trình viên sử dụng thao tác kéo thả vào form để thiết kế giao diện chương trình.

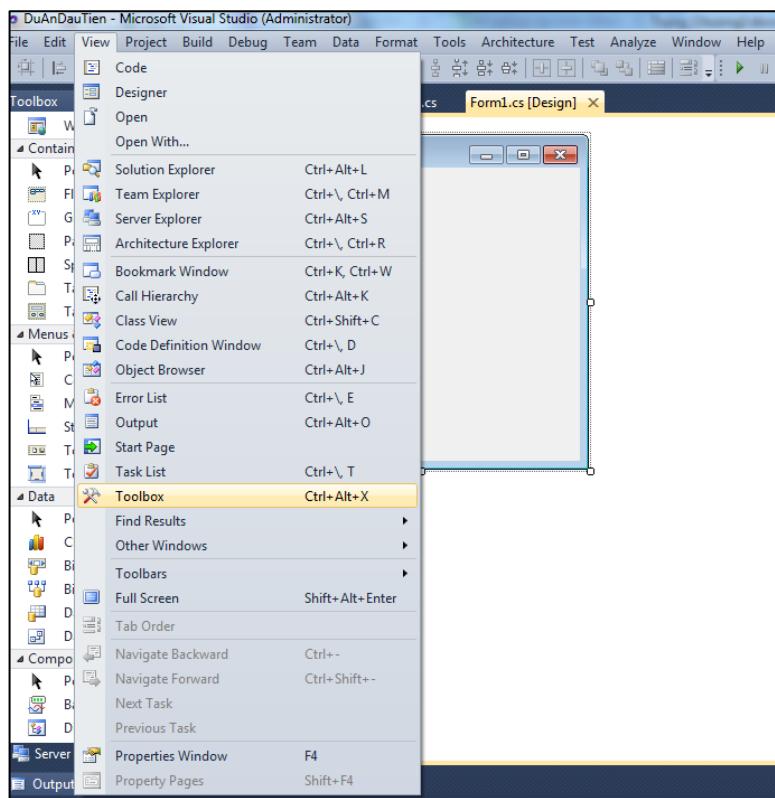
Khi tạo xong dự án, thì giao diện chính của dự án được hiển thị như hình 2.8. Phía bên tay trái hình 2.8 là thanh công cụ.

Ngoài những điều khiển hiển thị mặc định trong thanh công cụ, lập trình viên cũng có thể thêm các thành phần mới vào bằng cách chọn Project/AddReference.



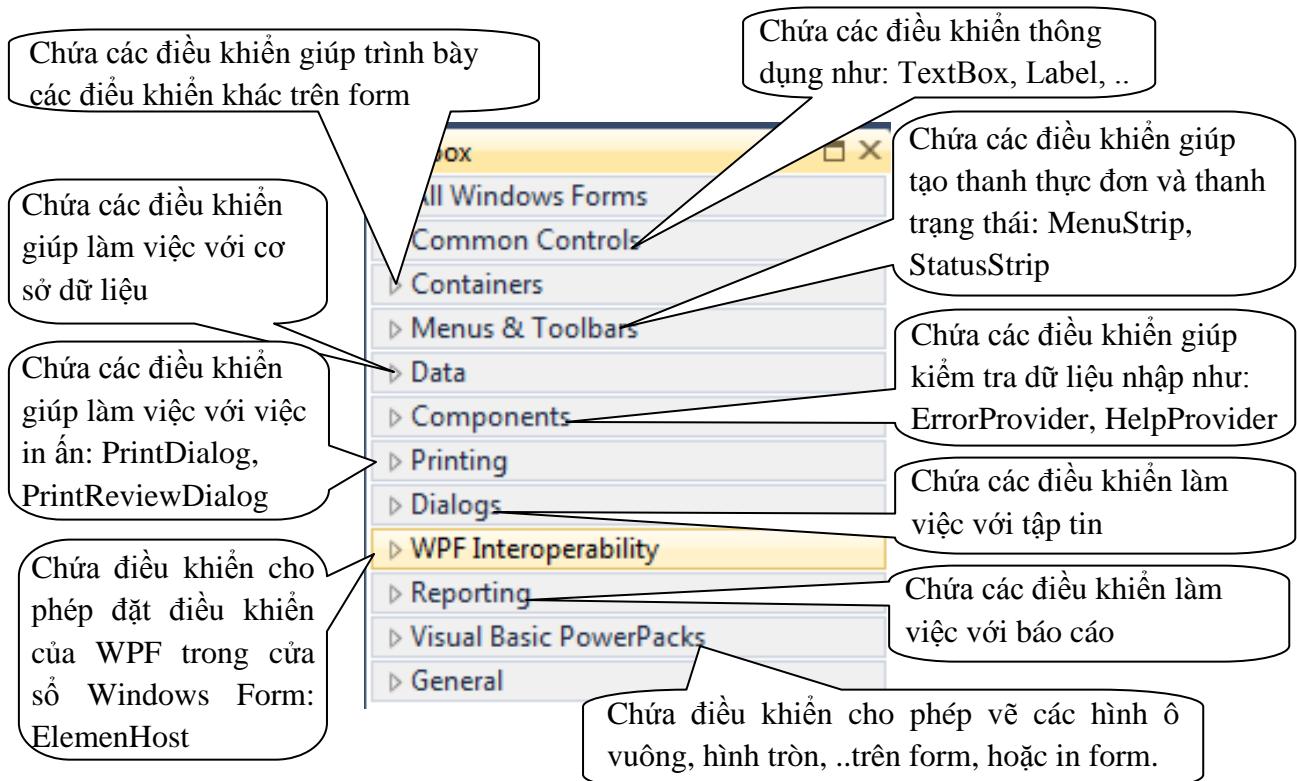
Hình 2.8: Giao diện chính với thanh công cụ bên trái

Nếu giao diện chính không hiển thị thanh công cụ, lập trình viên có thể hiển thị bằng cách chọn View/ Toolbox như hình 2.9.



Hình 2.9: Hiển thị thanh công cụ

Thanh công cụ bố trí các điều khiển thành những nhóm riêng biệt như hình 2.10:

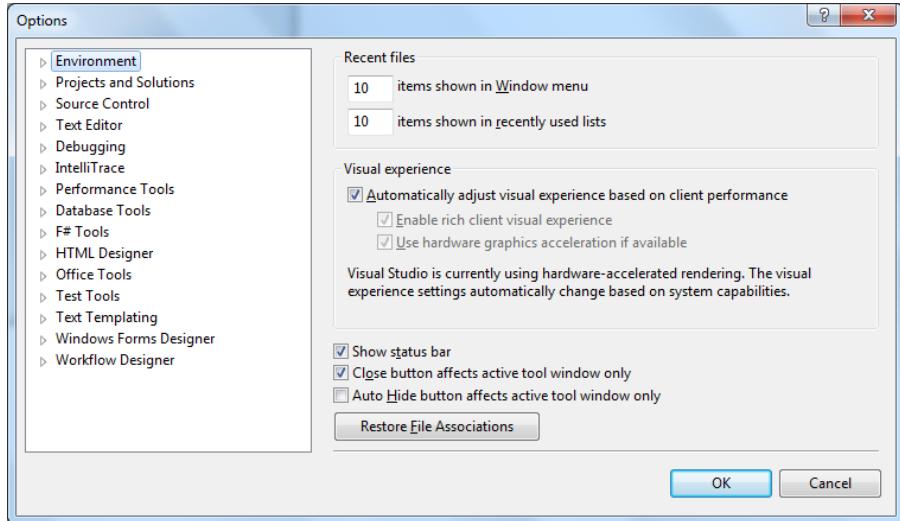


Hình 2.10: Thanh công cụ

Với việc phân loại các điều khiển theo nhóm giúp cho lập trình viên dễ dàng tìm kiếm điều khiển cần sử dụng hơn.

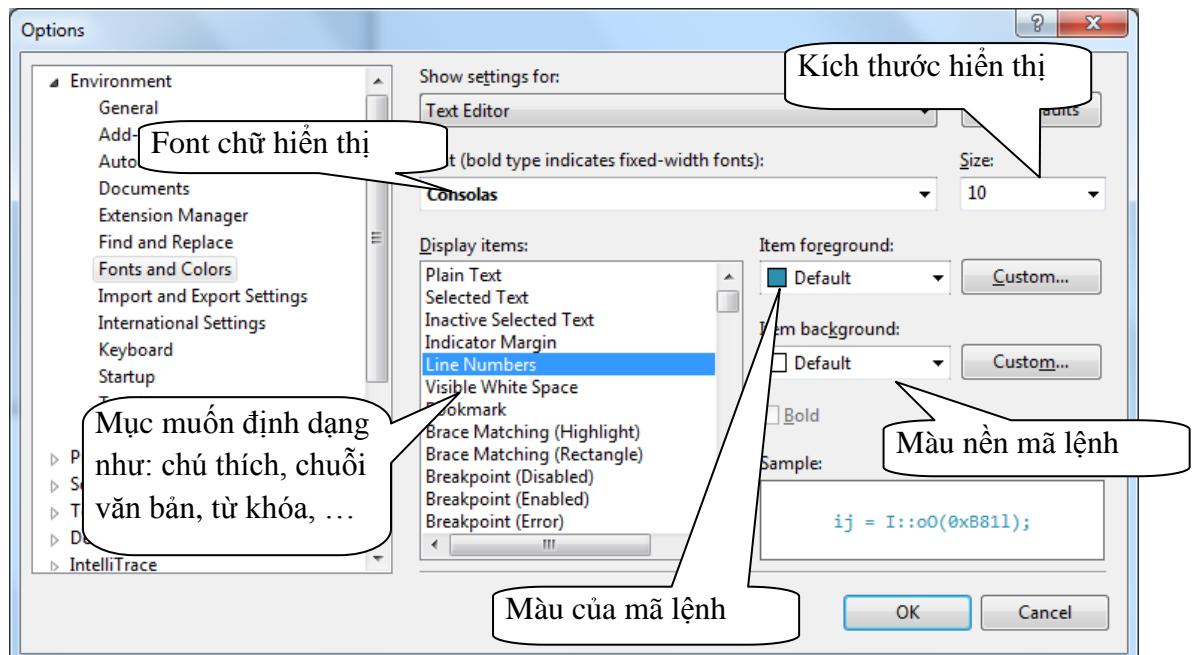
## 2.1.4. Định dạng mã C#

Để định dạng mã lệnh khi lập trình. Lập trình viên của thẻ tùy chỉnh lại định dạng mã trong cửa sổ Option. Đầu tiên bật cửa sổ Option bằng cách chọn *Tools/ Options* như hình 2.11:



Hình 2.11: Giao diện cửa sổ Option

Sau khi đã mở cửa sổ Option, chọn mục Evironment/ Fonts and Colors như hình 2.12 để định dạng lại mã lệnh:



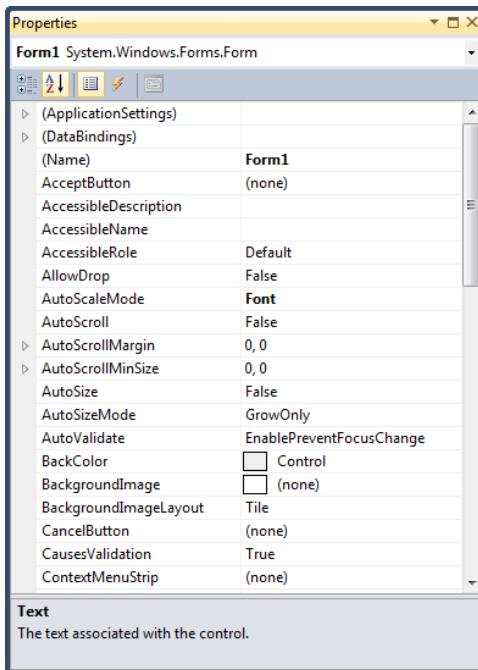
Hình 2.12: Giao diện định dạng mã lệnh

## 2.2. Các loại Form

### 2.2.1. Thuộc tính Form

- Thiết lập giá trị thuộc tính form trong cửa sổ Properties:

Windows Forms hỗ trợ nhiều thuộc tính cho việc tùy chỉnh form. Lập trình viên có thể thay đổi giá trị các thuộc tính này trong cửa sổ Properties như hình 2.13 để điều chỉnh kích thước, màu sắc, font chữ, ... của form cho phù hợp.



Hình 2.13: Giao diện cửa sổ Properties

Bảng 2.1 mô tả một số thuộc tính thường sử dụng trong việc thiết kế form

Bảng 2.1: Bảng mô tả thuộc tính Form

Thuộc tính	Mô tả
Name	Đặt tên form
AcceptButton	Giá trị thuộc tính này nhận là tên của một button trên form. Khi đó thay vì nhấp chuột vào button để thực thi thì người dùng có thể ấn phím Enter trên bàn phím
BackColor	Thiết lập màu nền của form
BackgroundImage	Thiết lập hình nền cho form
BackgroundImageLayout	Thiết lập việc hiển thị hình vừa thêm trong thuộc tính BackgroundImage sẽ hiển thị trên form ở dạng: bình thường (None), giữa (Center), ...
CancelButton	Giá trị thuộc tính này nhận là tên của một button trên form. Khi đó thay vì nhấp chuột vào button để thực thi thì người dùng có thể ấn phím Escape trên bàn phím
ControlBox	Mang giá trị true hoặc false. Nếu thiết lập

	thuộc tính là false thì sẽ loại bỏ các nút minimize và nút maximize trên form
<i>Cusor</i>	Thiết lập hình dạng con trỏ khi di chuyển con trỏ vào form
<i>Enable</i>	Mang giá trị true hoặc false; Nếu thiết lập thuộc tính là false thì điều khiển trong form sẽ không cho phép người dùng thao tác.
<i>Font</i>	Thiết lập văn bản hiển thị trên điều khiển
<i>ForeColor</i>	Thiết lập màu mặc định cho chuỗi của các điều khiển trên form
<i>FormBorderStyle</i>	Thiết lập đường viền của form và hành vi của form khi chạy chương trình
<i>HelpButton</i>	Mang giá trị true hoặc false; Nếu thiết lập thuộc tính là true thì trên thanh titlebar sẽ hiện 1 nút có dấu ? (nút này chỉ hiện khi hai thuộc tính MinimizeBox và MaximizeBox được thiết lập là false)
<i>Icon</i>	Biểu tượng hiển thị bên trái trên thanh titlebar của form
<i>KeyReview</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true cho phép các sự kiện bàn phím của form có hiệu lực
<i>Location</i>	Khi thuộc tính StartPosition được thiết lập là Manual, thì thuộc tính Location có tác dụng thiết lập vị trí hiển thị của form trên màn hình
<i>MaximizeBox</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là false thì nút maximize form trên thanh titlebar sẽ mất đi
<i>MaximumSize</i>	Thiết lập kích thước lớn nhất của form (chiều rộng x chiều cao)
<i>MinimizeBox</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là false thì nút minimize form trên thanh titlebar sẽ mất đi
<i>MinimumSize</i>	Thiết lập kích thước nhỏ nhất của form (chiều rộng x chiều cao)

<i>Opacity</i>	Thiết lập độ trong suốt cho form
<i>Size</i>	Kích thước form
<i>StartPosition</i>	Vị trí hiển thị của form trên màn hình
<i>Text</i>	Chuỗi văn bản hiển thị trên titlebar của form
<i>TopMost</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true thì form sẽ luôn hiển thị trên các cửa sổ khác
<i>Visible</i>	Mang giá trị true hoặc false: nếu thiết lập thuộc tính là true thì form sẽ được hiển thị trên màn hình, nếu là false sẽ không hiển thị trên màn hình
<i>WindowState</i>	Có 3 giá trị: Normal: hiển thị form bình thường; Minimized: khi chạy chương trình form sẽ bị thu nhỏ dưới thanh taskbar; Maximized: form hiển thị có kích thước đầy màn hình
<i>IsMDIContainer</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Form ở dạng MDI Form</li> <li>- Nếu là False: Form ở dạng bình thường</li> </ul>
<i>MdiParent</i>	mang giá trị là đối tượng MDI Form. Khi thiết lập giá trị cho thuộc tính MdiParent thì form sẽ trở thành Child Form

➤ Thiết lập giá trị thuộc tính form bằng mã lệnh:

Ngoài việc thiết kế form bằng việt thiết lập các giá trị trong cửa sổ properties.

Lập trình viên cũng có thể thiết lập giá trị thuộc tính của form bằng mã lệnh như sau:

- Thay đổi chuỗi văn bản hiển thị trên titlebar bằng cách thiết lập giá trị cho thuộc tính *Text*:

```
Form1.Text = "Đây là Form1";
```

- Thiết lập kích thước form: Có thể thiết lập giá trị cho thuộc tính *Width* và thuộc tính *Height* để quy định chiều rộng và chiều cao cho form.

```
Form1.Width = 300;
Form1.Height = 400;
```

hoặc:

```
Form1.Size = new Size(300, 400);
```

Lưu ý: Nếu thuộc tính *StartPosition* được thiết lập là *WindowsDefaultBounds*, thì form sẽ được thiết lập kích thước mặc định. *StartPosition* được thiết lập giá trị khác thì kích thước form sẽ được thiết lập như quy định ở thuộc tính *Size*.

- Thiết lập độ trong suốt cho form:

```
Form1.Opacity = 0.5;
```

- Thiết lập vị trí hiển thị của form: Sử dụng thuộc tính *StartPosition* để thiết lập vị trí hiển thị ban đầu của form trên màn hình. Giá trị thiết lập của thuộc tính *StartPosition* là một trong các giá trị quy định sẵn trong biến kiểu liệt kê *FormStartPosition*:

Bảng 2.2: Mô tả các giá trị của *FormStartPosition*

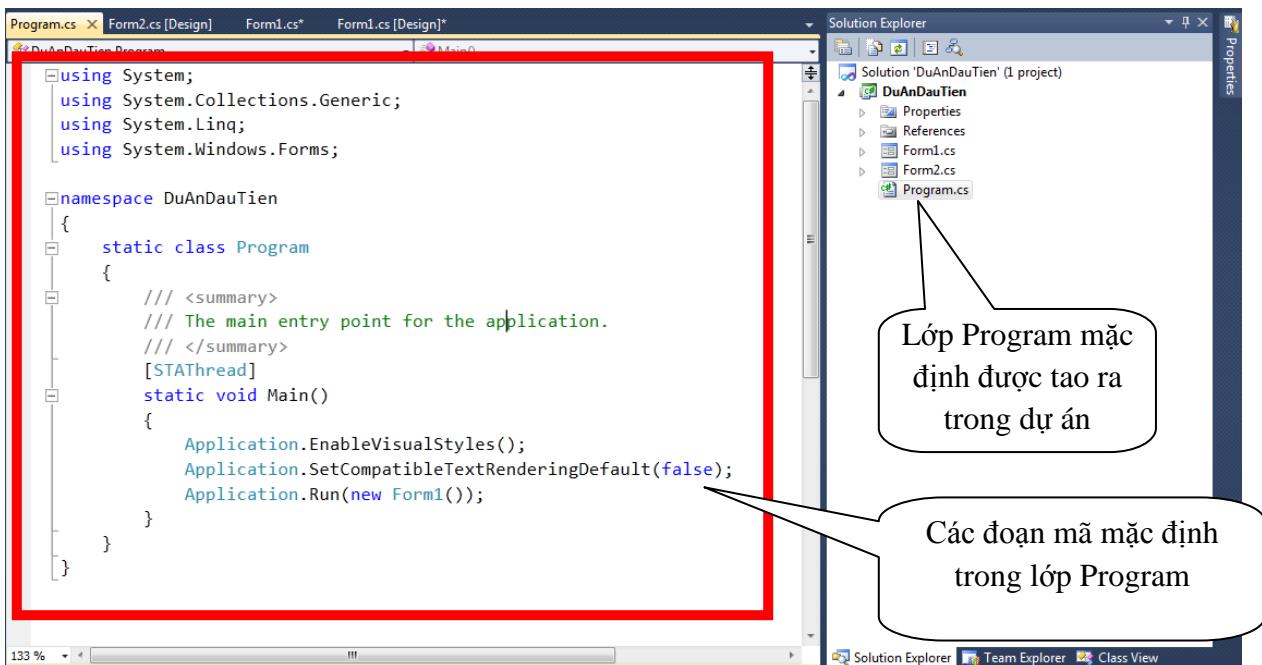
Giá trị	Mô tả
Manual	Nếu <i>StartPosition</i> mang giá trị Manual thì vị trí form hiển thị sẽ là vị trí thiết lập tại thuộc tính <i>Size</i> .
CenterScreen	Form hiển thị giữa màn hình
WindowsDefaultLocation	Form hiển thị tại vị trí mặc định với kích thước form sẽ là kích thước form được thiết lập tại thuộc tính <i>Size</i>
WindowsDefaultBounds	Form hiển thị tại vị trí mặc định với kích thước form sẽ là kích thước mặc định
CenterParent	Form hiển thị ở vị trí giữa form cha

Ví dụ 2.2: Hiển thị Form1 ở vị trí giữa màn hình

```
Form1.StartPosition = FormStartPosition.WindowsDefaultLocation;
```

- Thiết lập form hiển thị:

Trong C#, lập trình viên có thể thiết lập form hiển thị đầu tiên khi dự án được thực thi bằng cách sửa lại mã lệnh trong hàm main của lớp Program như hình 2.14:



Hình 2.14: Lớp Program

Lớp Program chứa một hàm bên trong là hàm Main. Khi dự án được thực thi, thì mã lệnh trong hàm Main sẽ thực thi trước. Cụ thể tại dòng:

`Application.Run(new Form1());`

Dòng lệnh trên chỉ ra rằng form đầu tiên sẽ hiển thị là Form1. Do đó, lập trình viên chỉ cần sửa lại mã lệnh chỉ định một form khác muốn hiển thị bằng cách thay tên form đó cho Form1.

Ví dụ 2.3: Hiển thị form có tên Form2 trong dự án

`Application.Run(new Form2());`

## 2.2.2. Các loại Form

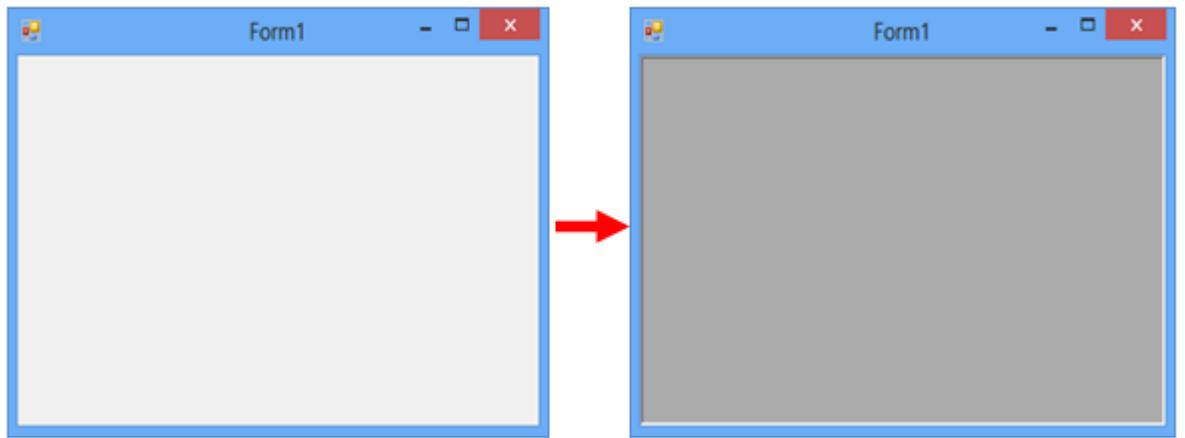
Trong ứng dụng Windows Forms, có 3 loại form: form bình thường (Normal Form), form cha (Mdi Form), form con (Child Form).

➤ Mdi Form:

Mdi Form là form có thể chứa các Child Form bên trong. Để làm được công việc đó thì form phải được thiết lập thuộc tính:

`IsMdiContainer = True;`

Lập trình viên có thể thiết lập thuộc tính `IsMdiContainer` trong cửa sổ Properties trên màn hình thiết kế form, hoặc bằng mã lệnh. Khi đã thiết lập thuộc tính `IsMdiContainer` là True thì hình dạng form sẽ thay đổi như hình 2.15.



*Hình 2.15: Hình dạng form khi chuyển từ Normal Form sang Mdi Form*

➤ Child Form:

Child Form là dạng form nằm bên trong vùng làm việc của Mdi Form hay nói cách khác là Child Form được chứa bên trong Mdi Form. Một form muốn trở thành Child Form cần phải khai báo thuộc tính MdiParent có giá trị là đối tượng Mdi Form.

Thuộc tính MdiParent không được biểu diễn trên cửa sổ Properties, do đó lập trình viên bắt buộc phải thiết lập giá trị MdiParent bằng mã lệnh.

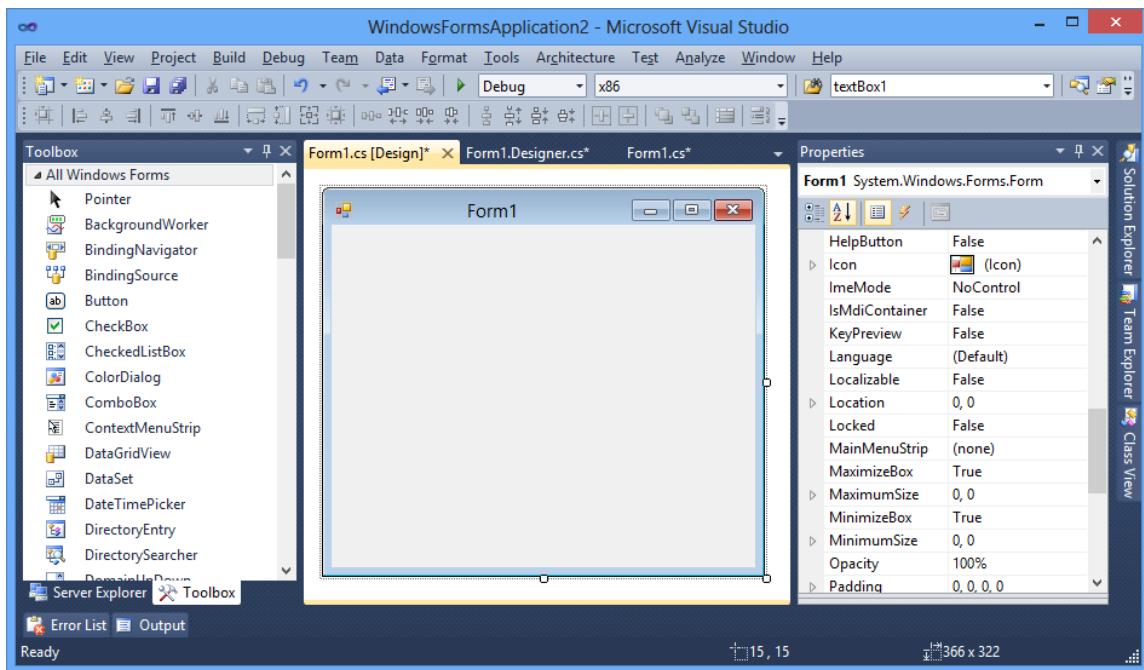
Ví dụ 2.4: Viết chương trình biểu diễn Mdi Form và Child Form như hình 2.16.



*Hình 2.16: Mdi Form và Child Form*

Hướng dẫn:

Bước 1: Tạo một dự án mới bằng C#, loại dự án là Windows Forms Application.  
Giao diện đầu tiên của chương trình hiển thị Form1 như hình 2.17.



Hình 2.17: Giao diện Form1 khi tạo dự án

Bước 2: Viết mã lệnh cho chương trình

Tại sự kiện *Load* của Form1 viết các dòng lệnh sau:

```
private void Form1_Load(object sender, EventArgs e)
{
    this.Text = "Mdi Form";
    this.IsMdiContainer = true;
    Form frm = new Form();
    frm.Text = "Child Form";
    frm.MdiParent = this;
    frm.Show();
}
```

Bước 3: Nhấn Ctrl + Shift + B để biên dịch mã nguồn và nhấn F5 để thực thi chương trình sẽ được Mdi Form và Child Form như hình 2.16.

➤ Normal Form:

Normal Form là form hoạt động độc lập với tất cả các form khác, nghĩa là form sẽ không chứa hoặc được chứa bởi một form nào khác. Thông thường khi tạo mới dự án Windows Forms Application thì form mặc định được tạo và hiển thị đầu tiên là Normal Form. Giao diện của Normal Form như hình 2.17.

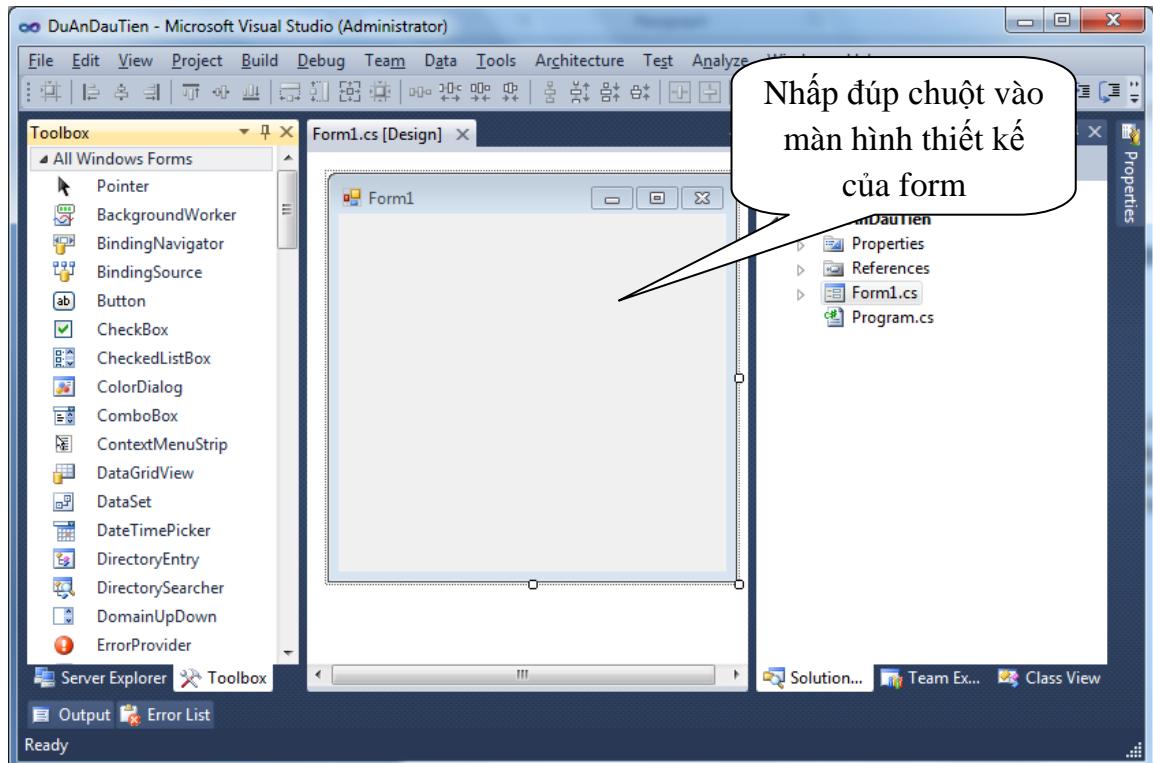
### 2.2.3. Các hình dạng của Form

Windows Forms mặc định các form được tạo ra đều ở dạng hình chữ nhật. Nếu lập trình viên muốn tạo form với hình dạng khác như hình tròn hoặc hình đa giác thì cần thiết lập lại thuộc tính *Region* của form trong sự kiện *Form\_Load*. Tuy nhiên, lập trình viên không thể thấy được hình dạng mới của form trong lúc thiết kế mà chỉ có thể

thấy được hình dạng khác của form khi chương trình được thực thi. Do đó sẽ cần phải thực thi chương trình nhiều lần để kích thước form phù hợp với nhu cầu thiết kế.

Ví dụ 2.5: Thiết kế form có hình ellipse có chiều rộng là 300 và chiều cao là 200

Bước 1: Tạo sự kiện Form\_Load bằng cách nhấp đúp chuột vào màn hình thiết kế form như hình 2.18:



Hình 2.18: Màn hình thiết kế form

Bước 2: Sau khi tạo ra sự kiện Form\_Load, tiến hành viết các dòng lệnh trong sự kiện Form\_Load như sau:

```
private void Form1_Load(object sender, EventArgs e)
{
    System.Drawing.Drawing2D.GraphicsPath path = new
        System.Drawing.Drawing2D.GraphicsPath();
    path.AddEllipse(0, 0, 300, 200);
    Region myRegion = new Region(myPath);
    this.Region = myRegion;
}
```

Bước 3: Án F5 để thực thi chương trình, khi chương trình được thực thi sẽ được form như hình 2.19:



Hình 2.19: Form hình Ellipse

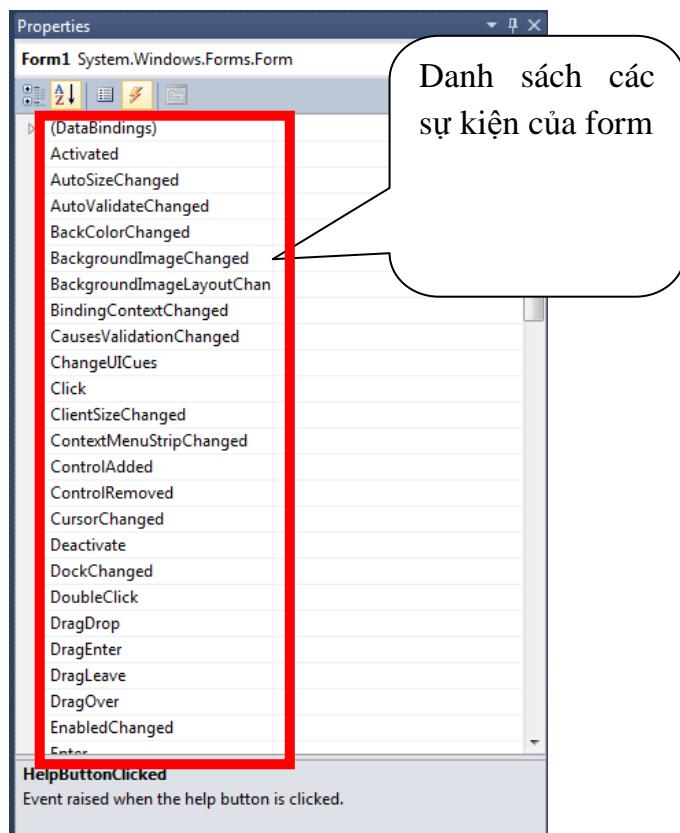
#### 2.2.4. Biến cố của Form

Biến cố của form hay có thể gọi là các hành động hoặc sự kiện liên quan đến form. Ứng dụng Windows Forms được hỗ trợ nhiều sự kiện và các đoạn mã tạo ra các sự kiện đều được sinh ra một cách tự động. Để sử dụng các sự kiện của form lập trình viên cần thực hiện các thao tác sau:

Bước 1: Nhấp chuột trái vào form

Bước 2: Trên cửa sổ Properties tương ứng, nhấp chuột trái vào biểu tượng để mở hộp thoại các sự kiện như hình 2.20.

Bước 3: Hộp thoại hình 2.20 chứa nhiều sự kiện liên quan đến form như: đóng form, mở form, di chuyển chuột vào form, ... Lập trình viên muốn sử dụng sự kiện nào chỉ cần nhấp đúp chuột trái vào sự kiện đó.



Hình 2.20: Danh sách sự kiện form

Ví dụ 2.6: Muốn sử dụng sự kiện nhấp chuột của form chỉ cần nhấp đôi chuột vào sự kiện *Click* thì một phương thức nhấp chuột của form sẽ được tự động phát sinh mã lệnh:

```
private void Form1_Click(object sender, EventArgs e)
{
    //Mã lệnh khi nhấp chuột vào form
}
```

- Bảng mô tả các sự kiện thường sử dụng của form:

Bảng 2.3: Bảng mô tả các sự kiện của form

Sự kiện	Mô tả
AutoSizeChanged	Xảy ra khi thuộc tính Autosize của Form chuyển từ True sang False hay ngược lại là False sang True
BackColorChanged	Xảy ra khi thuộc tính BackColor của Form thay đổi
Click	Xảy ra khi người dùng Click chuột vào vùng làm việc thuộc Form
ControlAdded	Xảy ra khi một điều khiển được Add vào Form
ControlRemoved	Xảy ra khi một điều khiển bị xóa khỏi Form
CursorChanged	Xảy ra khi thuộc tính Cursor của Form thay đổi
DoubleClick	Xảy ra khi người dùng DoubleClick vào vùng làm việc của Form
FontChanged	Xảy ra khi thuộc tính Font của Form có sự thay đổi
ForeColorChanged	Xảy ra khi thuộc tính ForeColor của Form có sự thay đổi
FormClosed	Xảy ra khi Form đã đóng (Nhấn vào nút X màu đỏ trên titlebar)
FormClosing	Xảy ra khi Form đang đóng (2 sự kiện FormClosed và FormClosing thường dùng trong lập trình CSDL: khi xảy ra sự kiện này thì đóng kết nối CSDL)
KeyDown	Xảy ra khi người dùng nhấn một phím hay một tổ hợp phím
KeyPress	Xảy ra khi người dùng nhấn một phím
KeyUp	Xảy ra khi người dùng nhả một phím
MouseClick	Xảy ra khi người dùng nhấn chuột (một trong 3 lựa chọn: Trái, giữa, phải)
MouseDoubleClick	Xảy ra khi người dùng nhấp đúp chuột vào một vùng làm việc của Form (một trong 3 lựa chọn: Trái,

	giữa, phải)
MouseDown	Xảy ra khi người dùng nhấn chuột
MouseHover	Xảy ra khi người dùng di chuyển vào các vùng làm việc Form
MouseLeave	Xảy ra khi di chuyển chuột ra khỏi vùng làm việc của Form
MouseMove	Xảy ra khi di chuyển chuột trên một vùng làm việc thuộc Form (nếu Form có chứa một điều khiển nào đó, khi di chuyển chuột trên điều khiển này thì không xảy ra sự kiện MouseMove của Form)
MouseUp	Xảy ra khi người dùng nhả nhấn chuột (có thể là chuột trái, chuột phải, chuột giữa - chuột cuộn)
Move	Xảy ra khi di chuyển Form (có sự thay đổi vị trí của Form)
StyleChanged	Xảy ra khi thuộc tính FormBorderStyle của Form thay đổi
TextChanged	Xảy ra khi thuộc tính Text của Form thay đổi.

Sự kiện FormClosed: Sự kiện này được gọi khi Form đã đóng

```
private void frmForm_FormClosed(object sender,
                               FormClosedEventArgs e)
{
    MessageBox.Show("Sự kiện FormClosed được gọi");
}
```

Sự kiện FormClosing: Xảy ra khi Form đang đóng

```
private void frmForm_FormClosing(object sender,
                                 FormClosingEventArgs e)
{
    DialogResult kq = MessageBox.Show("Bạn muốn đóng
                                      Form lại không?", "FormClosing",
                                      MessageBoxButtons.YesNo,
                                      MessageBoxIcon.Information);
    if (kq == DialogResult.Yes)
        e.Cancel = false; // Đóng Form lại
    else
        e.Cancel = true; // Không đóng Form nữa
}
```

Sự kiện KeyPress: Xảy ra khi ấn phím, nếu không chỉ rõ phím nào được nhấn thì khi nhấn bất cứ phím nào của sự kiện *KeyPress* của form đều xảy ra.

```
private void frmForm_KeyPress(object sender,
                               KeyPressEventArgs e)
{
    if (e.KeyChar == 'a') //nhấn phím a trên bàn phím
        MessageBox.Show("Sự kiện KeyPress được gọi");
}
```

Sự kiện KeyDown:

```
private void frmForm_KeyDown(object sender,
                             KeyEventArgs e)
{
    // Nhấn Ctrl+F gọi sự kiện KeyDown
    if (e.KeyCode == Keys.F &&
        e.Modifiers == Keys.Control)
        MessageBox.Show("Sự kiện KeyDown được gọi");
}
```

Sự kiện MouseClick: Xảy ra khi nhấn một trong 3 nút: chuột trái, giữa hoặc chuột phải.

```
private void frmForm_MouseClick(object sender,
                                MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left) //nhấn chuột trái
        MessageBox.Show("Sự kiện MouseClick được gọi");
    else
        if (e.Button == MouseButtons.Middle) //nhấn chuột giữa
            MessageBox.Show("Sự kiện MouseClick được gọi");
        else
            if (e.Button == MouseButtons.Right) //nhấn chuột phải
                MessageBox.Show("Sự kiện MouseClick được gọi");
}
```

## 2.2.5. Phương thức

Một số phương thức thường sử dụng của form: *Show()*, *ShowDialog()*, *Hide()*, *Close()*.

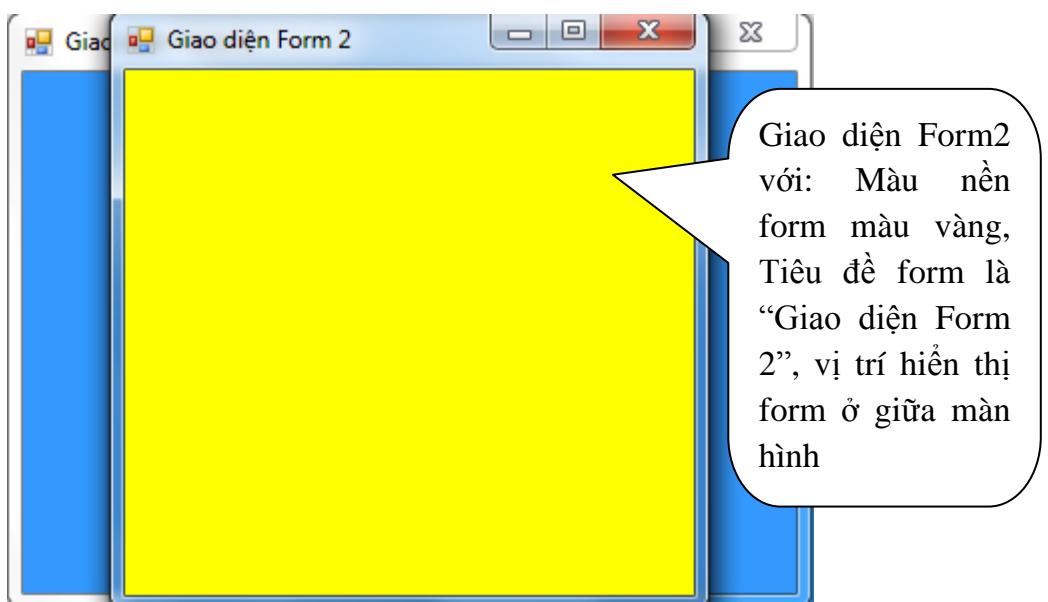
- Phương thức *Show()*: Phương thức *Show()* sử dụng để hiển thị form trên màn hình.

Ví dụ 2.7: Thiết kế chương trình như hình 2.21:



Hình 2.21: Giao diện Form1

Yêu cầu khi nhấp chuột vào Form1 như hình 2.21 thì sẽ hiển thị Form2 ở giữa màn hình như hình 2.22:



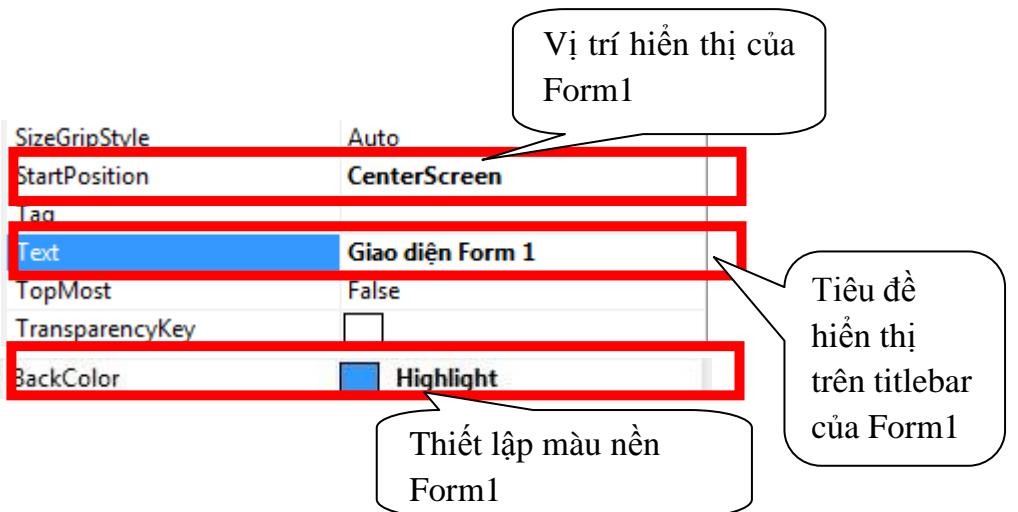
Hình 2.22: Giao diện Form2

Hướng dẫn:

Bước 1: Tạo Dự án Windows Forms mới với form mặc định ban đầu tên Form1.

Bước 2: Thiết lập các thuộc tính cho Form1 như sau:

- Thiết lập tiêu đề, màu nền và vị trí hiển thị cho Form1: Trên cửa sổ Properties thiết lập thuộc tính *Text*, thuộc tính *BackColor* và thuộc tính *StartPosition* như hình 2.23:



Hình 2.23: Thiết lập thuộc tính Form1

Sau khi thiết lập xong 3 thuộc tính như hình 2.23 sẽ được Form1 có giao diện như hình 2.21.

Bước 3: Tạo sự kiện *Click* của Form1 và thêm các mã lệnh sau cho sự kiện Click:

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Phương thức Show() giúp hiển thị Form2
    Form2.Show();
}
```

Khi bước 3 thực hiện xong và nhấn F5 để thực thi chương trình và *Click* chuột vào Form1 thì sẽ được Form2 như hình 2.22.

- Phương thức *ShowDialog()*: Phương thức *ShowDialog()* cũng có tác dụng hiển thị form như phương thức *Show()*. Nhưng khác biệt cơ bản là phương thức *Show()* giúp hiển thị form mới lên nhưng người dùng vẫn có thể quay lại thao tác trên các form đang hiển thị trước đó; Còn phương thức *ShowDialog()* sẽ hiển thị form mới lên và người dùng sẽ chỉ có thể thao tác trên form vừa hiển thị mà không thao tác được trên các form trước đó, do vậy người dùng muốn thao tác lên form hiển thị trước thì phải đóng form hiện hành bằng phương thức

`ShowDialog()`. Một đặc điểm khác nữa là phương thức `ShowDialog()` là phương thức trả về hai giá trị là `DialogResult.OK` và `DialogResult.Cancel`.

- Phương thức `ShowDialog()` trả về `DialogResult.OK` khi form đang hiển thị.
- Phương thức `ShowDialog()` trả về `DialogResult.Cancel` khi form đóng.

Ví dụ 2.8: Trong sự kiện `Click` của Form1 như bước 3 trên, tiến hành thay phương thức `Show()` thành phương thức `ShowDialog()`. Khi chương trình được thực thi thì người dùng chỉ có thể thao tác với Form2 mà không thể quay trở về Form1 được.

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Phương thức ShowDialog() giúp hiển thị Form2
    Form2.ShowDialog();
}
```

➤ Phương thức `Hide()`: Phương thức giúp ẩn một form để form đó không hiển thị trên màn hình. Việc ẩn form của phương thức `Hide()` thực chất là thiết lập thuộc tính `Visible = false`.

Ví dụ 2.9: Trong sự kiện `Click` của Form1 như bước 3, thực hiện yêu cầu là khi nhập chuột vào Form1 thì Form1 sẽ ẩn đi và Form2 sẽ hiện lên.

```
private void Form1_Click(object sender, EventArgs e)
{
    //this là con trỏ đại diện cho Form hiện hành, Form
    //hiện hành ở đây là Form1
    this.Hide();
    Form Form2 = new Form();
    Form2.Text = "Giao diện Form 2";
    Form2.StartPosition = FormStartPosition.CenterScreen;
    Form2.BackColor = Color.CadetBlue;
    Form2.ShowDialog();
}
```

- Phương thức *Close()*: Sử dụng để đóng form.

Ví dụ 2.10: Trong sự kiện *Click* của Form1 như bước 3, thực hiện yêu cầu là khi đóng Form2 thì Form1 cũng đóng.

```
private void Form1_Click(object sender, EventArgs e)
{
    //Tạo đối tượng lớp Form
    Form Form2 = new Form();
    //Thiết lập tiêu đề trên titlebar của form
    Form2.Text = "Giao diện Form 2";
    //Thiết lập vị trí hiển thị form
    Form2.StartPosition = FormStartPosition.CenterScreen;
    //Thiết lập màu nền cho form
    Form2.BackColor = Color.CadetBlue;
    //Kiểm tra giá trị trả về của phương thức ShowDialog()
    //nếu giá trị trả về là DialogResult.Cancel thì Form2
    //đã đóng, tiến hành đóng Form1 bằng phương thức
    //Close()
    if (Form2.ShowDialog() == DialogResult.Cancel)
    {
        this.Close();
    }
}
```

### 2.3. Bài tập cuối chương

Câu 1: Trong các dòng mã lệnh sau đây, mã lệnh nào cho phép tạo và hiển thị một đối tượng Windows Form mới có tên là Form1

- Form1 frm = new Form1;  
frm.Show();
- Form Form1 = new Form();  
Form1.Show();
- Form1 frm ;  
frm.Show();
- Form frm;  
frm.Show();
- Form Form1 = new Form();  
Form1.ShowDialog();

Câu 2: Trong các thuộc tính sau, thuộc tính nào dùng để thiết lập nội dung hiển thị trên thanh title bar và thuộc tính nào dùng để thiết lập màu nền của form.

- Thuộc tính Text và ForeColor
- Thuộc tính Display và BackColor
- Thuộc tính Text và BackColor

- d) Thuộc tính Display và ForeColor

Câu 3: Các thuộc tính sau, thuộc tính nào cho phép thiết lập form trở thành Mdi Form

- a) IsMdiContainer
- b) MdiParent
- c) MdiContainer
- d) ParentForm

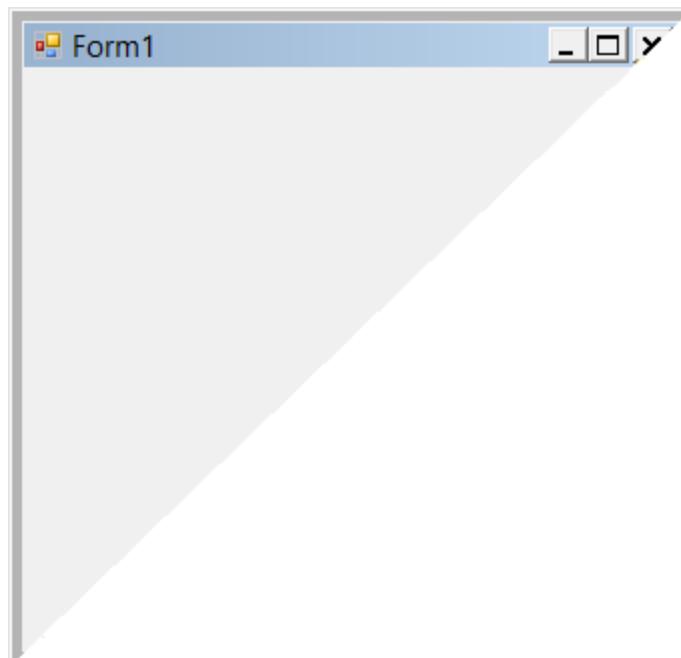
Câu 4: Các thuộc tính sau, thuộc tính nào cho phép thiết lập form trở thành Child Form

- a) IsMdiContainer
- b) MdiParent
- c) MdiContainer
- d) ParentForm

Câu 5: Trong các sự kiện sau, sự kiện nào sẽ phát sinh khi form đã đóng

- a) FormClosed
- b) FormClosing
- c) ClosedForm
- d) ClosingForm
- e) Load
- f) Click

Câu 6: Xây dựng form có dạng hình tam giác như hình 2.21:

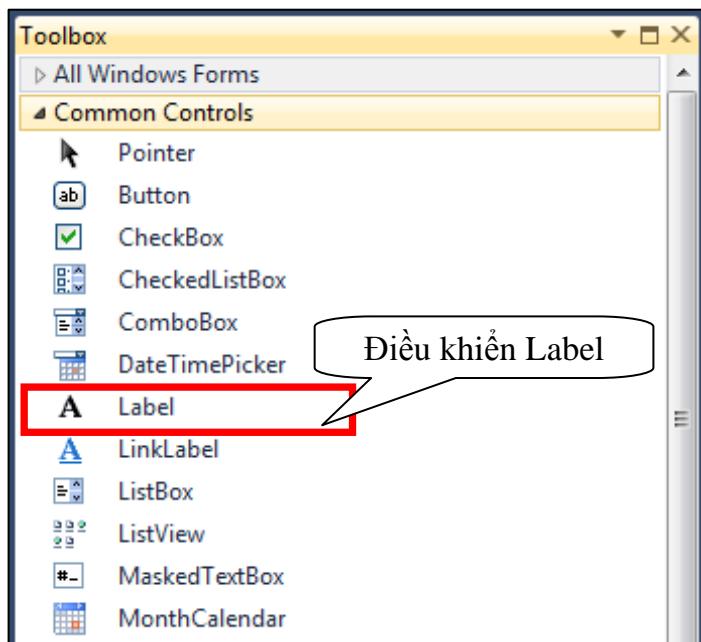


Hình 2.21: Form hình dạng tam giác

## CHƯƠNG 3: CÁC ĐIỀU KHIỂN THÔNG THƯỜNG

### 3.1. Điều khiển Label

*Label* thường dùng hiển thị thông tin chỉ đọc và thường sử dụng kèm với các điều khiển khác để mô tả chức năng. *Label* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.1.



Hình 3.1: Điều khiển Label trong cửa sổ Properties

- Một số thuộc tính thường dùng của *Label*:

Bảng 3.1: Bảng mô tả thuộc tính *Label*

Thuộc tính	Mô tả
<i>BorderStyle</i>	Thiết lập đường viền
<i>Font</i>	Kích thước và kiểu chữ hiển thị trên <i>Label</i>
<i>Text</i>	Nội dung hiển thị trên <i>Label</i>
<i>Backcolor</i>	Màu nền của <i>Label</i>
<i>Name</i>	Tên của <i>Label</i>
<i>Locked</i>	Khóa không cho di chuyển
<i>Enabled</i>	Đánh dấu tích là đối tượng sẽ ở trạng thái hoạt động
<i>Cursor</i>	Đổi hình trỏ chuột khi đưa vào <i>Label</i>
<i>ForeColor</i>	Thiết lập màu chữ hiển thị trên <i>Label</i>
<i>TextAlign</i>	Căn lề chữ hiển thị trên <i>Label</i>
<i>Visible</i>	Ẩn hoặc hiện <i>Label</i>

Ví dụ 1: Thiết kế form hiển thị thông tin sinh viên như hình 3.2:

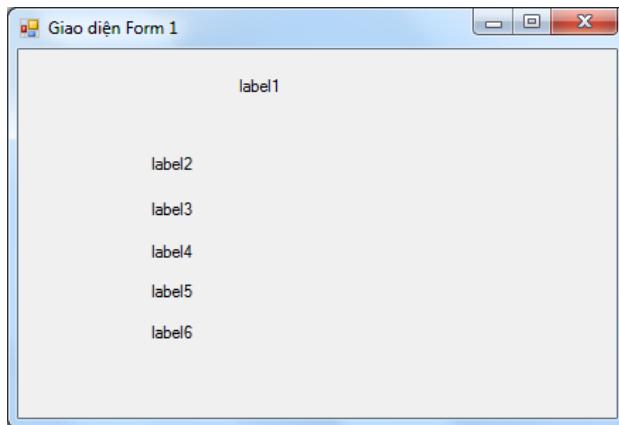


Hình 3.2: Giao diện hiển thị thông tin sinh viên

Hướng dẫn:

Bước 1: Tạo dự án Windows Forms mới, Form1 mặc định tạo sẵn trong dự án đặt tên tiêu đề trên titlebar như sau: “Giao diện Form1”

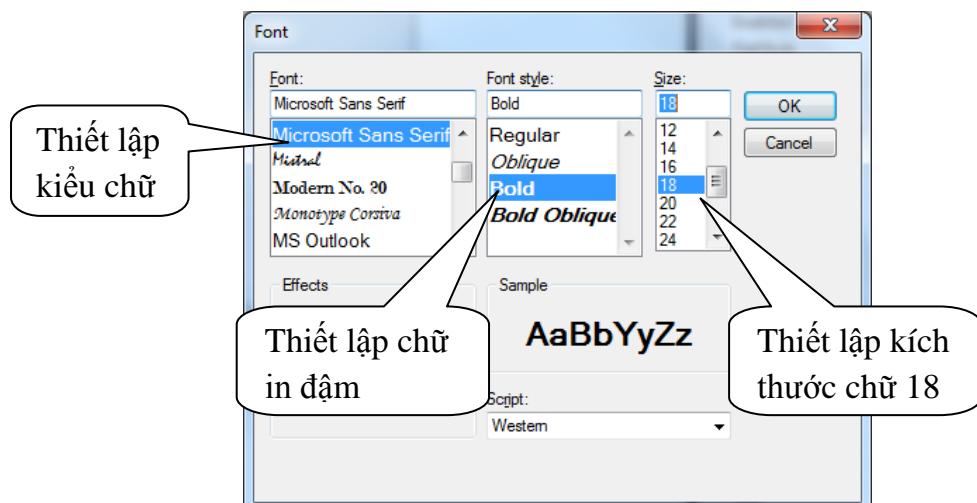
Bước 2: Kéo thả các *Label* trong cửa sổ *Toolbox* vào Form1 như hình 3.3 sau:



Hình 3.3: Giao diện sau khi kéo Label vào Form1

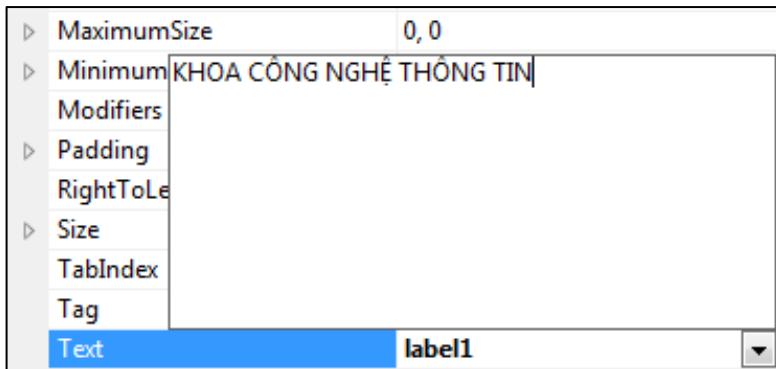
Bước 3: Thiết lập giá trị thuộc tính cho label1

Nhấp chuột trái vào Form1 và thiết lập thuộc tính *Font* trong cửa sổ *Properties* như hình 3.4:



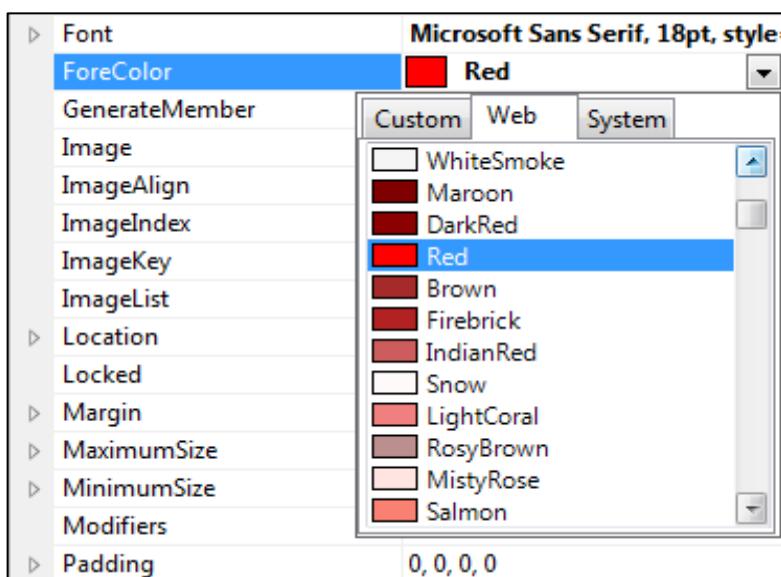
Hình 3.4: Cửa sổ thiết lập thuộc tính Font

Thiết lập thuộc tính *Text* cho label1 như hình 3.5:



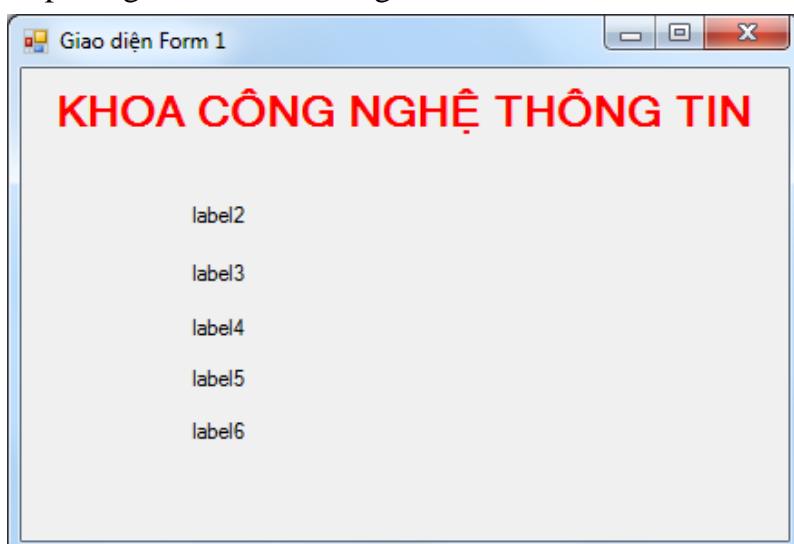
Hình 3.5: Thiết lập thuộc tính *Text* cho label1

Thiết lập thuộc tính *ForeColor* như hình 3.6:



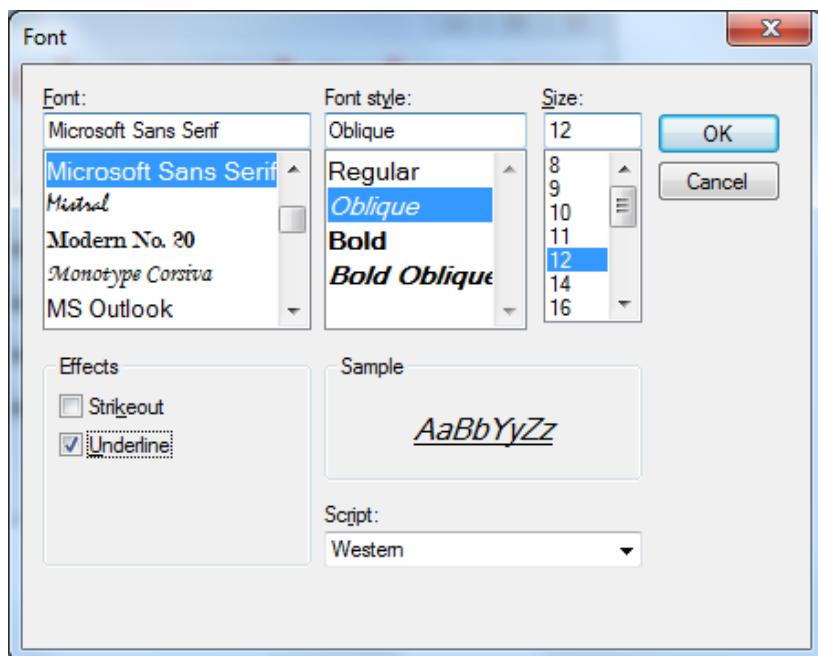
Hình 3.6: Thiết lập thuộc tính *ForeColor*

Sau khi thiết lập xong thuộc tính *Text*, giao diện Form1 được như hình 3.7:



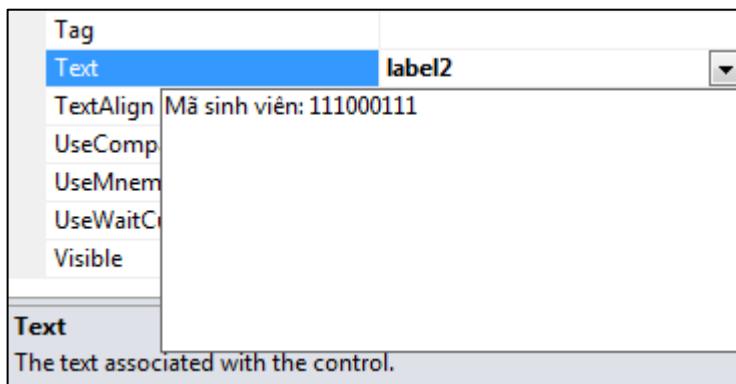
Hình 3.7: Giao diện Form1 sau khi thiết lập xong thuộc tính *Text* cho label1

Bước 4: Thiết lập các thuộc tính *Font* cho label2, label3, label4, label5, label6 như hình 3.8:



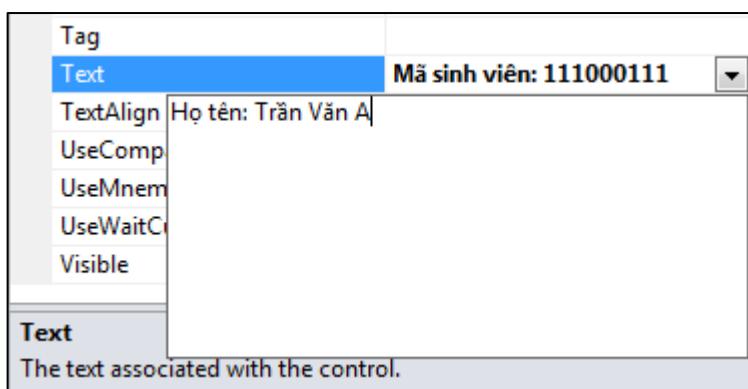
Hình 3.8: Thiết lập thuộc tính *Font* cho label2, label3, label4, label5, label6

Thiết lập thuộc tính *Text* cho label2 như hình 3.9:



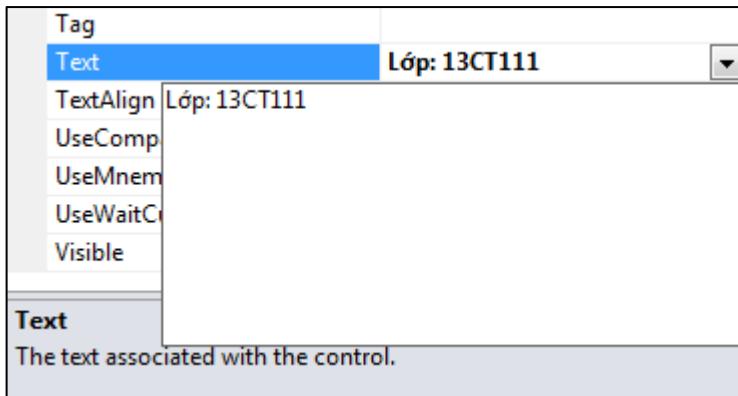
Hình 3.9: Thiết lập thuộc tính *Text* cho label2

Thiết lập thuộc tính *Text* cho label3 như hình 3.10:



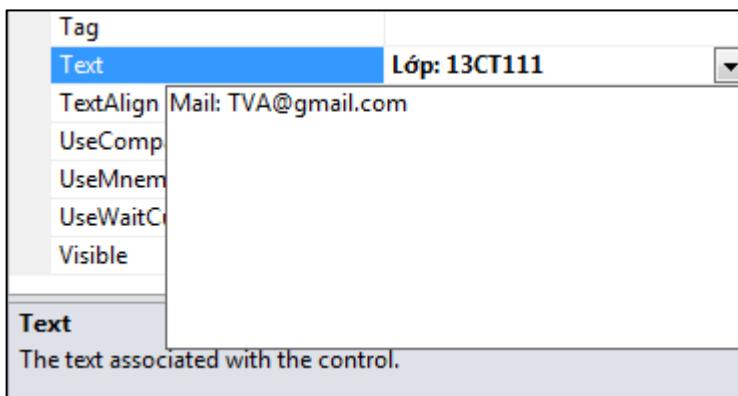
Hình 3.10: Thiết lập thuộc tính *Text* cho label3

Thiết lập thuộc tính *Text* cho label4 như hình 3.11:



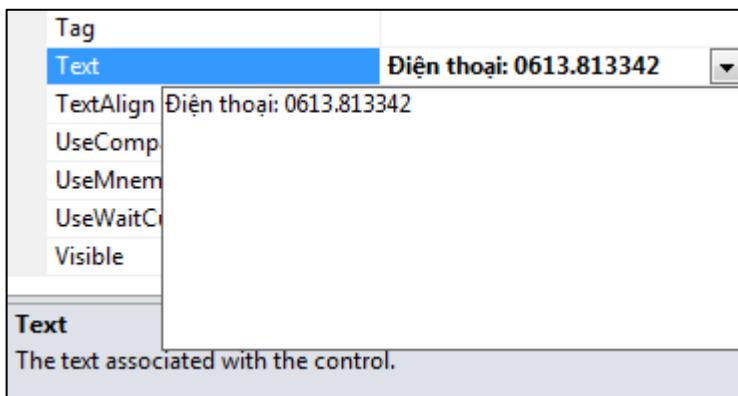
Hình 3.11: Thiết lập thuộc tính *Text* cho label4

Thiết lập thuộc tính *Text* cho label5 như hình 3.12:



Hình 3.12: Thiết lập thuộc tính *Text* cho label5

Thiết lập thuộc tính *Text* cho label6 như hình 3.13:



Hình 3.13: Thiết lập thuộc tính *Text* cho label5

Bước 5: Nhấn F5 thực thi chương trình sẽ được form như hình 3.2.

### 3.2. Điều khiển Button

*Button* là điều khiển tạo giao diện nút lệnh trên form, khi người dùng nhấp chuột vào nút lệnh thì chương trình sẽ thực hiện một hành động nào đó. *Button* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.14.



Hình 3.14: Điều khiển Button trong cửa sổ Properties

- Một số thuộc tính thường dùng của *Button*:

Bảng 3.2: Bảng mô tả thuộc tính *Button*

Thuộc tính	Mô tả
<i>Name</i>	Đặt tên cho nút lệnh
<i>Text</i>	Nội dung hiển thị trên nút nhấn
<i>Visible</i>	Ẩn/ hiện nút nhấn
<i>Enable</i>	Cho phép/ không cho phép tương tác với nút lệnh
<i>Font</i>	Chỉ định kiểu chữ, kích cỡ chữ hiển thị
<i>ForeColor</i>	Màu chữ hiển thị trên nút lệnh
<i>Image</i>	Hình ảnh hiển thị trên nút lệnh
<i>BackColor</i>	Màu của nút lệnh
<i>TabIndex</i>	Chỉ định thứ tự tab của các <i>Button</i> trên form

- Một số sự kiện thường dùng của *Button*:

Bảng 3.3: Bảng mô tả sự kiện *Button*

Sự kiện	Mô tả
<i>Click</i>	Sự kiện nhấn chuột vào <i>Button</i>
<i>MouseEnter</i>	Chuột nằm trong vùng thay đổi của <i>Button</i>
<i>MouseHover</i>	Rê chuột vào vùng của <i>Button</i>
<i>MouseLeave</i>	Rê chuột ra khỏi vùng của <i>Button</i>
<i>MouseMove</i>	Chuột được di chuyển trên <i>Button</i> .

Ví dụ 2: Thiết kế form xử lý nút lệnh như hình 3.15. Yêu cầu: Khi nhấp chuột vào nút lệnh “**Hiển thị**” thì in sẽ hiển thị chữ “**Xin chào**” tại vị trí “---nội dung hiển thị---”, khi nhấp chuột vào nút lệnh “**Thoát**” sẽ đóng chương trình.



*Hình 3.15: Giao diện form xử lý nút lệnh*

Hướng dẫn:

Bước 1: Kéo các điều khiển từ cửa sổ *Toolbox* vào form như hình 3.16



*Hình 3.16: Giao diện ban đầu của form xử lý nút lệnh*

Bước 2: Thiết lập thuộc tính cho điều khiển trong cửa sổ *Properties*:

- label1:
  - Thuộc tính *Font*: kích cỡ chữ 18;
  - Thuộc tính *ForeColor*: đỏ;
  - Thuộc tính *Text*: “KHOA CÔNG NGHỆ THÔNG TIN”
- label2:
  - Thuộc tính *Name*: lblNoiDung;
  - Thuộc tính *Font*: kích cỡ chữ 20;
  - Thuộc tính *Text*: “-----nội dung hiển thị-----”
- button1:
  - Thuộc tính *Name*: btnHienThi
  - Thuộc tính *Text*: “Hiển thị”
- button2:
  - Thuộc tính *Name*: btnThoat
  - Thuộc tính *Text*: “Thoát”

Bước 3: Viết mã lệnh cho các nút lệnh:

- Sự kiện *Click* của nút lệnh btnHienThi:

```

private void btnHienThi_Click(object sender, EventArgs e)
{
    LblNoiDung.Text = "Xin chao"
}

```

- Sự kiện *Click* của nút lệnh *btnThoat*:

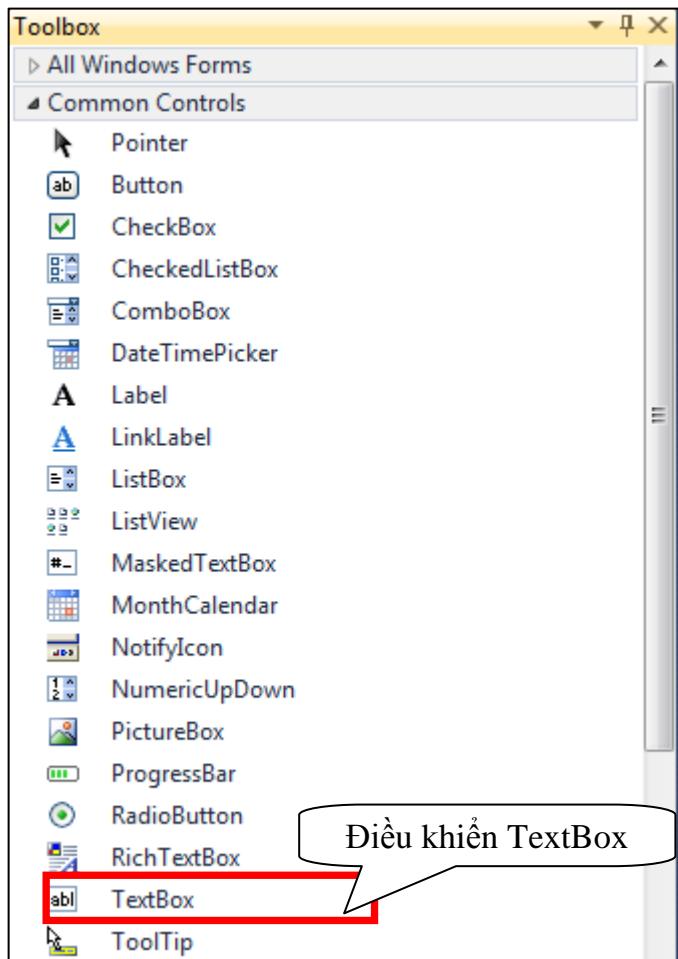
```

private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}

```

### 3.3. Điều khiển TextBox

*TextBox* là điều khiển dùng để nhập chuỗi làm dữ liệu đầu vào cho ứng dụng hoặc hiển thị chuỗi. Người dùng có thể nhập nhiều dòng, hoặc tạo mặt nạ để nhập mật khẩu. *TextBox* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.14.



Hình 3.14: Điều khiển *TextBox* trong cửa sổ *Properties*

- Một số phương thức thường dùng của *TextBox*:

*Bảng 3.4 : Bảng mô tả các phương thức của TextBox*

Phương thức	Mô tả
<i>Clear()</i>	Xóa tất cả chuỗi hiển thị trong <i>TextBox</i>
<i>Cut()</i>	Di chuyển phần nội dung bôi đen của chuỗi
<i>Paste()</i>	Dán phần nội dung được chọn của chuỗi
<i>Copy()</i>	Sao chép phần nội dung bôi đen của chuỗi
<i>Undo()</i>	Khôi phục thao tác trước
<i>Select()</i>	Chọn một phần nội dung của chuỗi trong <i>TextBox</i>
<i>SelectAll()</i>	Chọn tất cả nội dung của chuỗi trong <i>TextBox</i>
<i>DeselectAll()</i>	Bỏ chọn chuỗi trong <i>TextBox</i>

- Một số thuộc tính thường dùng của *TextBox*:

*Bảng 3.5: Bảng mô tả các thuộc tính của TextBox*

Thuộc tính	Mô tả
<i>Name</i>	Tên của <i>TextBox</i> để gọi viết lệnh
<i>Text</i>	Nội dung hiển thị ban đầu khi chương trình chạy
<i>Font</i>	Chọn kiểu chữ, kích thước chữ cho <i>TextBox</i>
<i>ForeColor</i>	Chọn màu chữ cho <i>TextBox</i>
<i>BackColor</i>	Chọn màu nền cho <i>TextBox</i>
<i>Enable</i>	Cho phép/Không cho phép thao tác trên <i>TextBox</i>
<i>Multiline</i>	Cho phép <i>TextBox</i> có nhiều dòng hay không
<i>PasswordChar</i>	Ký tự thay thế khi nhập vào <i>TextBox</i>
<i>ReadOnly</i>	Cho phép/Không cho phép sửa dữ liệu trên <i>TextBox</i>
<i>Visible</i>	Ẩn/ hiện <i>TextBox</i>
<i> TextAlign</i>	Canh lề dữ liệu trong <i>TextBox</i>
<i>MaxLength</i>	Quy định chuỗi Max sẽ nhập vào <i>TextBox</i> , mặc định là 32767
<i>ScrollBars</i>	Các loại thanh cuộn (dọc, ngang, cả hai)
<i>WordWrap</i>	Cho phép văn bản tự động xuống dòng khi độ dài vượt quá chiều ngang của <i>TextBox</i>
<i>BorderStyle</i>	định kiểu đường viền cho <i>TextBox</i>
<i>TabIndex</i>	Chỉ định thứ tự tab của các <i>TextBox</i> trên form

<i>Focus</i>	<i>TextBox</i> sẵn sàng được tương tác bởi người sử dụng
<i>CanUndo</i>	Mang giá trị True hoặc False, nếu là True thì cho phép thực hiện phương thức <i>Undo()</i> , mang giá trị True khi <i>TextBox</i> đã thực hiện thao tác thêm, sửa hay xóa nội dung.
<i>SelectedText</i>	Lấy ra phần nội dung chuỗi được bôi đen
<i>SelectionStart</i>	Vị trí bắt đầu chọn nội dung của chuỗi
<i>SelectionLength</i>	Chiều dài chuỗi sẽ chọn trong <i>TextBox</i>
<i>HideSelection</i>	Mang giá trị True hoặc False, nếu là True thì không cho phép sử dụng thuộc tính <i>SelectionStart</i> , nếu là giá trị False thì cho phép sử dụng <i>SelectionStart</i>

- Một số sự kiện thường dùng của *TextBox*:

Bảng 3.6: Bảng mô tả các sự kiện của *TextBox*

Sự kiện	Mô tả
<i>KeyDown</i>	Thực hiện công việc nào đó khi một phím được nhấn xuống
<i>KeyUp</i>	Thực hiện công việc nào đó khi một phím được nhả ra
<i>KeyPress</i>	Xảy ra khi người sử dụng nhấn một phím và nhả ra, ta dùng sự kiện này để lọc các phím không muốn nhận như cho nhập số (0 đến 9) không cho nhập chuỗi. Mỗi sự kiện <i>KeyPress</i> cho ta một cặp sự kiện <i>KeyDown</i> và <i>KeyUp</i>
<i>Click</i>	Nhấp chuột vào <i>TextBox</i>
<i>DoubleClick</i>	Nhấp đúp chuột vào <i>TextBox</i>
<i>MouseEnter</i>	Chuột nằm trong vùng thấy được của <i>TextBox</i>
<i>MouseHover</i>	Chuột nằm trong vùng hiển thị một quãng thời gian
<i>MouseLeave</i>	Chuột ra khỏi vùng nhập liệu của <i>TextBox</i>
<i>MouseMove</i>	Chuột được di chuyển trên <i>TextBox</i>
<i>TextChanged</i>	Giá trị của thuộc tính <i>Text</i> bị thay đổi

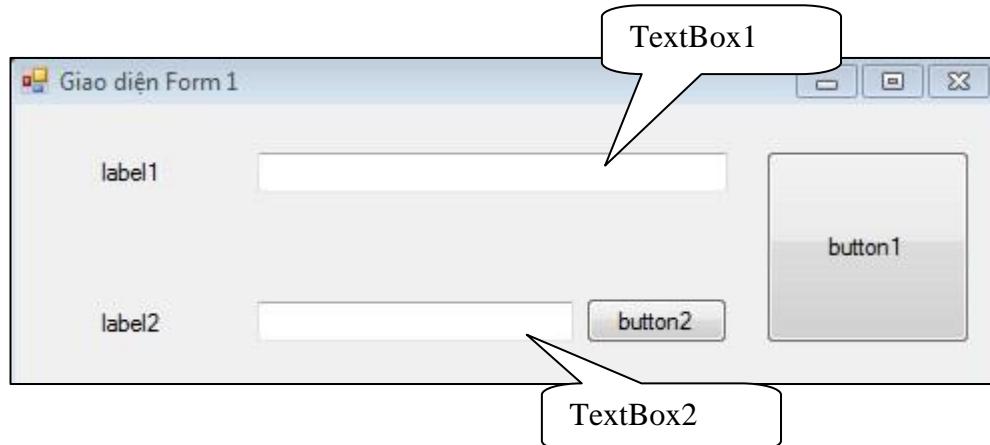
Ví dụ 3: Thiết kế form và thực hiện các yêu cầu như hình 3.15:



Hình 3.15: Giao diện form xử lý TextBox

Hướng dẫn:

Bước 1: Thiết kế form ban đầu như hình 3.16



Hình 3.16: Giao diện ban đầu của form xử lý TextBox

Bước 2: Thiết lập các thuộc tính của điều khiển trong cửa sổ Properties:

- label1:  
Thuộc tính *Text* = “Nội dung.”
- label2:  
Thuộc tính *Text* = “Nhập chuỗi.”
- textbox1:  
Thuộc tính *Name* = txtNoiDung  
Thuộc tính *Text* = “Quả cau nho nhỏ miếng trầu cay”

Này của Xuân Hương đã quêt rồi  
Có phải duyên nhau thì thăm lại  
Đừng xanh như lá bạc như vôi”

Thuộc tính *Multiline* = true; Sử dụng chuột điều chỉnh kích thước textbox1 như hình 3.15.

- textbox2:  
Thuộc tính *Name* = txtTimKiem
- button1:  
Thuộc tính *Name* = btnTimKiem  
Thuộc tính *Text* = “Tìm kiếm”
- button2:  
Thuộc tính *Name* = btnThoat  
Thuộc tính *Text* = “Thoát”

Bước 3: Viết mã lệnh cho các nút lệnh:

- Sự kiện *Click* của nút lệnh btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Click* của nút lệnh btnTimKiem:

```
private void btnTimKiem_Click(object sender, EventArgs e)
{
    int i;
    i = txtNoiDung.Text.IndexOf(txtChuoitk.Text);
    if (i >= 0)
    {
        txtNoiDung.SelectionStart = i;
        txtNoiDung.SelectionLength =
            txtChuoitk.Text.Length;
        MessageBox.Show(txtNoiDung.SelectedText);
    }
    else
        MessageBox.Show("Không tìm thấy");
}
```

### 3.4. Điều khiển ComboBox và ListBox

ComboBox và ListBox là hai điều khiển có nhiều điểm tương đồng, đều sử dụng để chứa dữ liệu cho phép người dùng lựa chọn.

### 3.4.1. ListBox

Tại mỗi thời điểm có thể chọn một hoặc nhiều dòng dữ liệu, không cho nhập mới. Điều khiển *ListBox* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.17



Hình 3.17: Điều khiển *ListBox* trong cửa sổ *Properties*

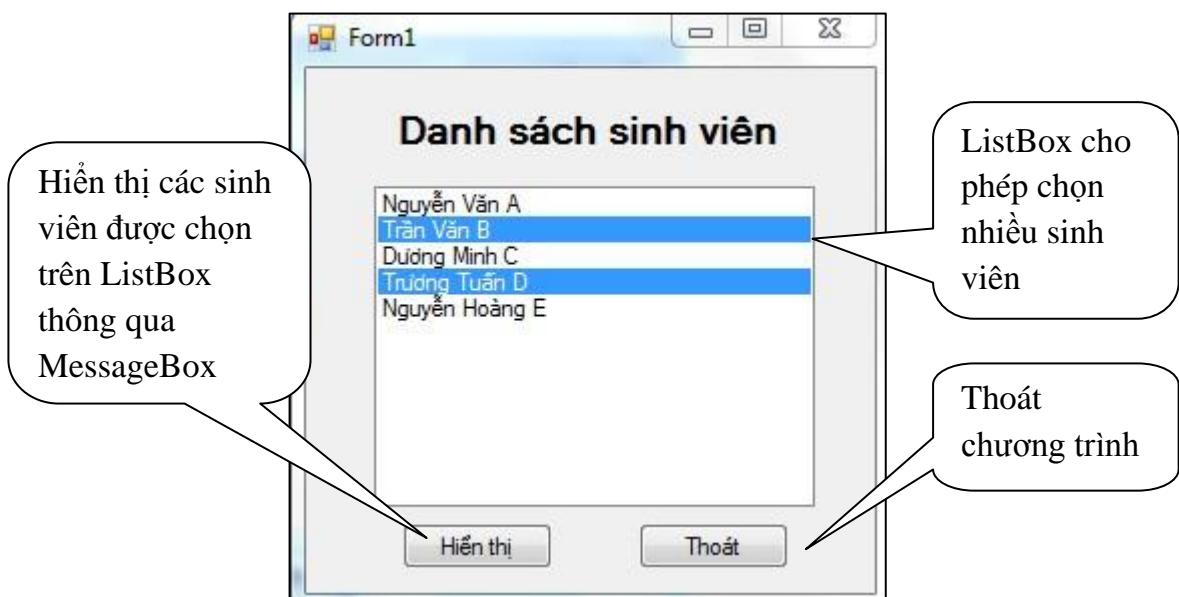
- Một số thuộc tính thường dùng của *ComboBox*:

Bảng 3.7: Bảng mô tả các thuộc tính của *ComboBox*

Thuộc tính	Mô tả
<i>SelectionMode</i>	Cho phép chọn một hoặc nhiều dòng dữ liệu trên <i>ListBox</i> , bao gồm: <ul style="list-style-type: none"> <li>▪ One: chỉ chọn một giá trị.</li> <li>▪ MultiSimple: cho phép chọn nhiều, chọn bằng cách click vào mục chọn, bỏ chọn bằng cách click vào mục đã chọn.</li> <li>▪ MultiExtended: chọn nhiều bằng cách nhấn kết hợp với Shift hoặc Ctrl</li> </ul>
<i>SelectedItems</i>	Được sử dụng khi <i>SelectionMode</i> là <i>MultiSimple</i> hoặc <i>MultiExtended</i> . Thuộc tính <i>SelectedItems</i> chứa các dòng dữ liệu được chọn.
<i>SelectedIndices</i>	Được sử dụng khi <i>SelectionMode</i> là <i>MultiSimple</i> hoặc <i>MultiExtended</i> . Thuộc tính <i>SelectedIndices</i> chứa các chỉ số của các dòng dữ liệu được chọn.
<i>FormatString</i>	Chuỗi định dạng. Tất cả các mục chọn trong <i>ListBox</i> sẽ được định dạng theo chuỗi này. Việc

	định dạng này chỉ thực hiện khi thuộc tính <i>FormattingEnabled</i> được thiết lập là <i>True</i>
<i>FormatingEnable</i>	Mang hai giá trị <i>True</i> và <i>False</i> . Nếu là <i>True</i> thì cho phép các mục chọn trong <i>ListBox</i> có thể định dạng lại. Nếu là <i>False</i> thì không thể định dạng.
<i>Items</i>	Trả về các mục chứa trong <i>ListBox</i>
<i>DataSource</i>	Chọn tập dữ liệu điền vào <i>ListBox</i> . Tập dữ liệu có thể là mảng chuỗi, <i>ArrayList</i> , ..
<i>SelectedIndex</i>	Lấy chỉ số mục được chọn, chỉ số mục chọn đầu tiên là 0
<i>SelectedItem</i>	Trả về mục được chọn
<i>SelectedValue</i>	Trả về giá trị của mục chọn nếu <i>ListBox</i> có liên kết dữ liệu. Nếu không liên kết dữ liệu hoặc thuộc tính <i>ValueMember</i> không được thiết lập thì giá trị thuộc tính <i>SelectedValue</i> là giá trị chuỗi của thuộc tính <i>SelectedItem</i>
<i>ValueMember</i>	Thuộc tính này chỉ định dữ liệu thành viên sẽ cung cấp giá trị cho <i>ListBox</i>

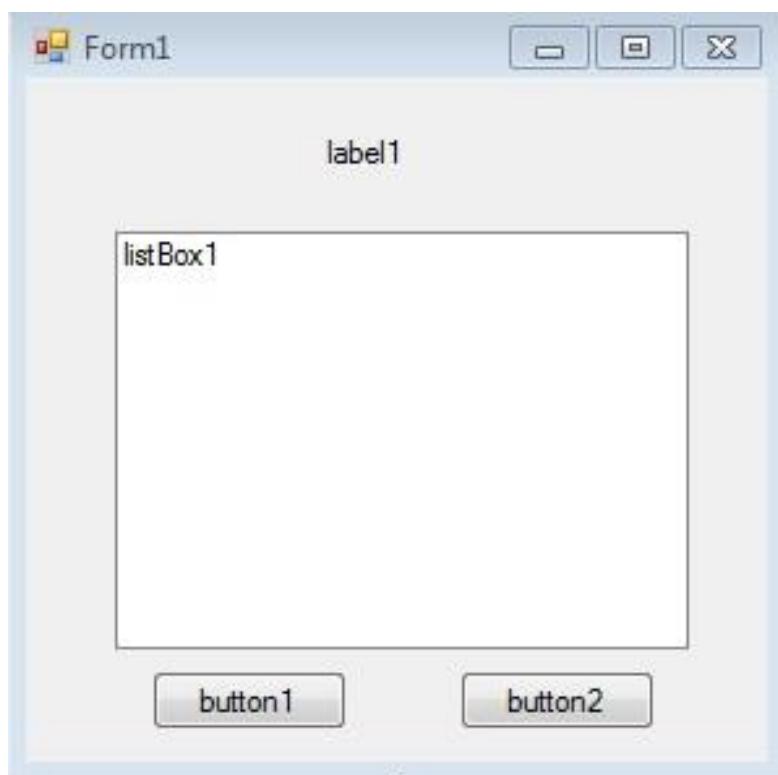
Ví dụ 4: Xây dựng chương trình với *ListBox* chứa danh sách tên của các sinh viên như hình 3.18



Hình 3.18: Giao diện ví dụ 4

Hướng dẫn:

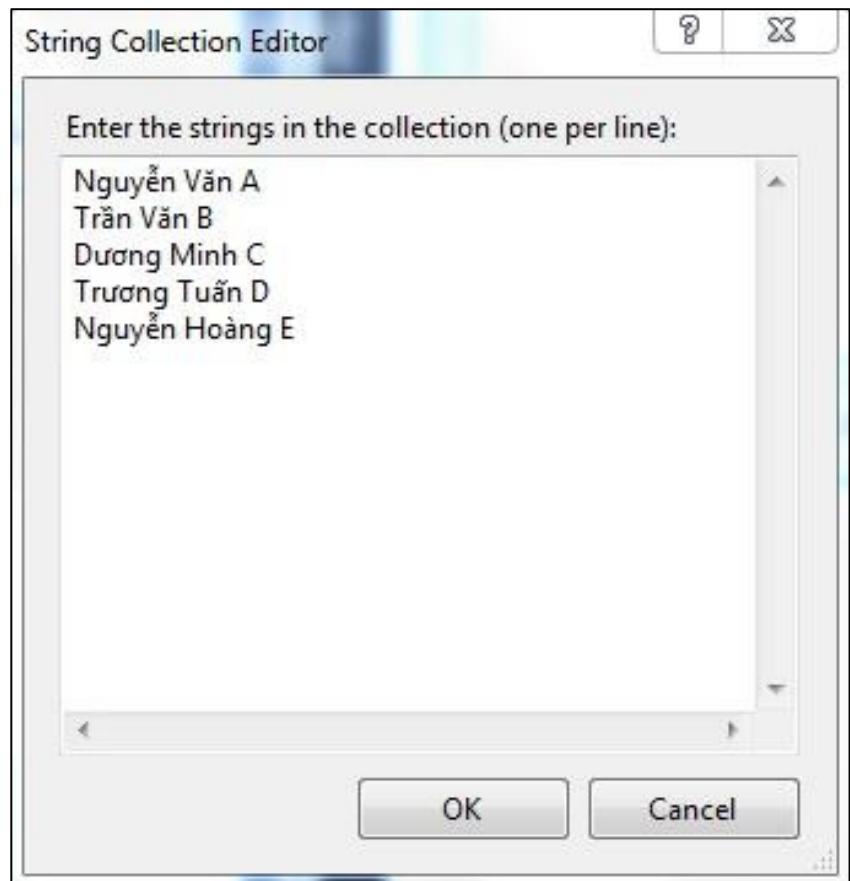
Bước 1: Thiết kế giao diện ban đầu như hình 3.19



Hình 3.19: Giao diện ban đầu ví dụ 4

Bước 2: Thiết lập giá trị các thuộc tính cho điều khiển

- label1:
  - Thuộc tính *Text* = “Danh sách sinh viên”
  - Thuộc tính *Size* = 16
- button1:
  - Thuộc tính *Text* = “Hiển thị”
  - Thuộc tính *Name* = btnHienThi
- button2:
  - Thuộc tính *Text* = “Thoát”
  - Thuộc tính *Name* = btnThoat
- listBox1: listBox1 chứa danh sách tên sinh viên, danh sách tên sinh viên có thể đưa vào listBox1 bằng cách thiết lập thuộc tính *Items* của listBox1 trong cửa sổ *Properties* như hình 3.20



Hình 3.20: Thiết lập thuộc tính Items cho listBox1

Bước 3: Viết mã lệnh cho các nút lệnh:

- Sự kiện Click của nút lệnh btnHienThi:

```
private void btnHienThi_Click(object sender, EventArgs e)
{
    string str="";
    foreach(string item in listBox1.SelectedItems)
    {
        str=str+item+" ";
    }
    MessageBox.Show(str);
}
```

- Sự kiện Click của nút lệnh btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

### 3.4.2. ComboBox

Mỗi lần chỉ có thể chọn một giá trị, có thể nhập mới dữ liệu vào. Điều khiển *ComboBox* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.21



Hình 3.21: Điều khiển *ComboBox* trong cửa sổ *Properties*

- Một số thuộc tính thường dùng của *ComboBox*:

Bảng 3.8: Bảng mô tả thuộc tính của *ComboBox*

Thuộc tính	Mô tả
<i>Text</i>	Trả về nội dung dòng dữ liệu đang hiển thị trên <i>ComboBox</i>
<i>DropdownStyle</i>	Quy định dạng của <i>ComboBox</i> , nhận một trong các giá trị: <ul style="list-style-type: none"> <li>▪ <i>DropDown</i>: giá trị mặc định, có thể chọn hoặc nhập mới mục dữ liệu vào <i>ComboBox</i>.</li> <li>▪ <i>Simple</i>: hiển thị theo dạng <i>ListBox</i> + <i>TextBox</i> có thể chọn dữ liệu từ <i>ListBox</i> hoặc nhập mới vào <i>TextBox</i>.</li> <li>▪ <i>DropDownList</i>: chỉ cho phép chọn dữ liệu trong <i>ComboBox</i></li> </ul>
<i>DropDownHeight</i>	Thiết lập chiều cao tối đa khi sổ xuống của <i>ComboBox</i>
<i>DropDownWidth</i>	Thiết lập độ rộng của mục chọn trong <i>ComboBox</i>
<i>FormatString</i>	Chuỗi định dạng. Tất cả các mục chọn trong <i>ComboBox</i> sẽ được định dạng theo chuỗi này. Việc định dạng này chỉ thực hiện khi thuộc tính <i>FormattingEnabled</i> được thiết lập là <i>True</i>
<i>FormattingEnabled</i>	Mang hai giá trị <i>True</i> và <i>False</i> . Nếu là <i>True</i> thì cho phép các mục chọn trong <i>ComboBox</i> có thể định dạng lại. Nếu là <i>False</i> thì không thể định dạng.
<i>Items</i>	Trả về các mục chứa trong <i>ComboBox</i>

<i>DataSource</i>	Chọn tập dữ liệu điền vào <i>ComboBox</i> . Tập dữ liệu có thể là mảng chuỗi, <i>ArrayList</i> , ..
<i>SelectedIndex</i>	Lấy chỉ số mục được chọn, chỉ số mục chọn đầu tiên là 0
<i>SelectedItem</i>	Trả về mục được chọn
<i>SelectedText</i>	Lấy chuỗi hiển thị của mục chọn
<i>SelectedValue</i>	Trả về giá trị của mục chọn nếu <i>ComboBox</i> có liên kết dữ liệu. Nếu không liên kết dữ liệu hoặc thuộc tính <i>ValueMember</i> không được thiết lập thì giá trị thuộc tính <i>SelectedValue</i> là giá trị chuỗi của thuộc tính <i>SelectedItem</i>
<i>ValueMember</i>	Thuộc tính này chỉ định dữ liệu thành viên sẽ cung cấp giá trị cho <i>ComboBox</i>

Ví dụ 5: Xây dựng chương trình với giao diện như hình 3.22 gồm *ComboBox* chứa danh sách 4 màu (vàng, xanh, đỏ, đen) và một Label hiển thị dòng chữ “**Hoàng Sa, Trường Sa là của Việt Nam**”. Yêu cầu khi chọn màu trong *ComboBox* thì màu của dòng chữ trên Label sẽ thay đổi tương ứng.



Hình 3.22: Giao diện ví dụ 5

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 3.23



Hình 3.23: Giao diện ban đầu ví dụ 5

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển

- label1:

- Thuộc tính Text = “Chọn màu:”
- button1:  
Thuộc tính Name = btnDoiMau  
Thuộc tính Text = “Đổi màu”
  - button2:  
Thuộc tính Name = btnThoat  
Thuộc tính Text = “Thoát”
  - label2:  
Thuộc tính Name = lblHienThi  
Thuộc tính Text = “Trường Sa, Hoàng Sa là của Việt Nam”
  - comboBox1: comboBox1 chứa danh sách 4 màu gồm: Vàng, xanh, đỏ, đen. Danh sách màu này có thể đưa vào comboBox1 bằng cách thiết lập thuộc tính *Items* của comboBox1 trong cửa sổ *Properties* như hình 3.24



Hình 3.24: Thiết lập thuộc tính *Items* của comboBox1

Bước 3: Viết mã lệnh cho các nút lệnh:

- Sự kiện *Click* của nút lệnh btnDoiMau:

```
private void btnDoiMau_Click(object sender, EventArgs e)
{
    if (comboBox1.Text == "Vàng")
        lblHienThi.ForeColor = Color.Yellow;
    if (comboBox1.Text == "Đỏ")
        lblHienThi.ForeColor = Color.Red;
    if (comboBox1.Text == "Đen")
        lblHienThi.ForeColor = Color.Black;
    if (comboBox1.Text == "Xanh")
        lblHienThi.ForeColor = Color.Blue;
}
```

- Sự kiện *Click* của nút lệnh *btnThoat*:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

### 3.4.3. Phương thức và sự kiện của ComboBox và ListBox

Một số phương thức và thuộc tính thường dùng của *ComboBox.Items* hoặc *ListBox.Items*:

Bảng 3.9: Bảng mô tả các phương thức và thuộc tính của *ComboBox.Item* và *ListBox.Item*

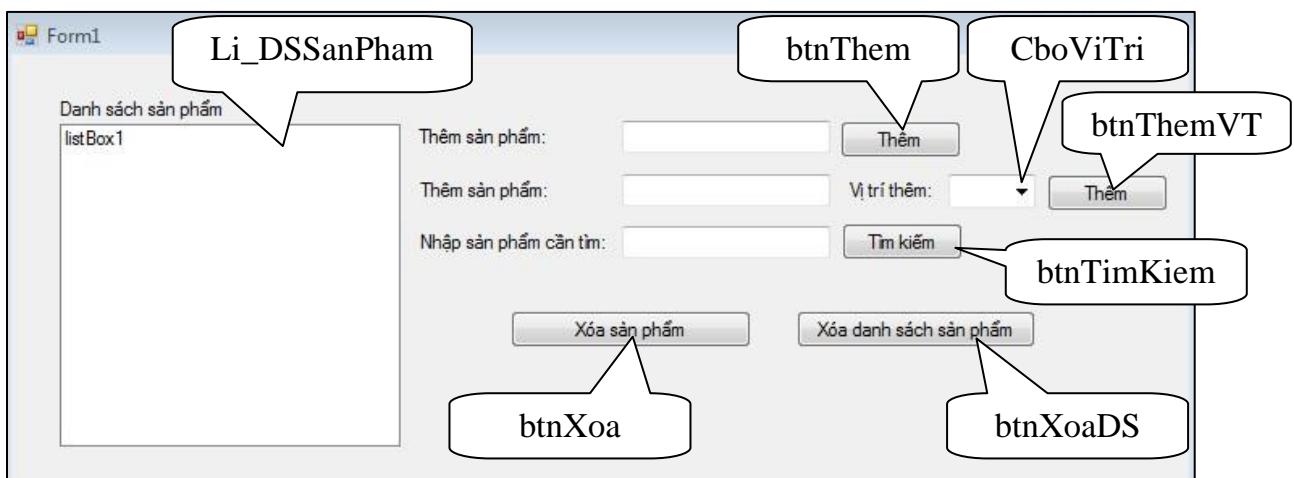
Phương thức	Mô tả
<i>Add(&lt;Mục mới&gt;)</i>	Thêm một mục <Mục mới> vào cuối danh sách <i>ComboBox</i> hoặc <i>ListBox</i>
<i>AddRange(&lt;Mảng mục chọn&gt;)</i>	Thêm một mảng các mục
<i>Insert( i, &lt;Mục mới&gt; )</i>	Chèn thêm mục chọn <Mục mới> vào vị trí <i>i</i>
<i>Count</i>	Trả về số mục chọn hiện đang có
<i>Item( i )</i>	Trả về mục chọn ở vị trí thứ <i>i</i>
<i>Remove(&lt;Mục cần xóa&gt;)</i>	Xóa mục chọn <Mục cần xóa> khỏi <i>ComboBox</i> hoặc <i>ListBox</i>
<i>RemoveAt( i )</i>	Xóa mục chọn có chỉ số <i>i</i> khỏi <i>ComboBox</i> hoặc <i>ListBox</i>
<i>Contains(&lt;Mục cần tìm&gt;)</i>	Trả về True nếu có mục chọn <Mục cần tìm> trong danh sách, trả về False nếu không có mục chọn trong <i>ComboBox</i> hoặc <i>ListBox</i>
<i>Clear()</i>	Xóa tất cả các mục chọn
<i>IndexOf(&lt;Mục cần tìm&gt;)</i>	Trả về chỉ số mục chọn <Mục cần tìm> trong <i>ComboBox</i> hoặc <i>ListBox</i> , nếu không tìm thấy sẽ trả về -1

- Sự kiện thường dùng của *ComboBox* và *ListBox*:

Cả *ComboBox* và *ListBox* đều sử dụng sự kiện *SelectedIndexChanged* để kiểm tra sự thay đổi mục chọn của người dùng.

Ví dụ 5: Thiết kế form quản lý danh sách sản phẩm như hình 3.25. Với CboViTri là điều khiển *ComboBox* chứa danh sách chỉ số của danh sách sản phẩm trong *ListBox* Li\_DSSanPham. Yêu cầu chức năng:

- Khi nhấn nút btnThem thì sẽ thêm sản phẩm mới vào cuối danh sách trong *ListBox* Li\_DSSanPham.
- Khi nhấn nút btnThemVT thì sẽ thêm sản phẩm mới vào danh sách tại vị trí như CboViTri chỉ định.
- Khi nhấn nút btnTimKiem sẽ hiển thị một *MessageBox* thông báo có tìm thấy sản phẩm trong danh sách không.
- Khi nhấn nút btnXoa sẽ xóa sản phẩm được chọn trong danh sách.
- Khi nhấn nút btnXoaDS sẽ xóa tất cả sản phẩm trong danh sách.



Hình 3.25: Giao diện form quản lý sản phẩm

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 3.26



Hình 3.26: Giao diện ban đầu form quản lý sản phẩm

Bước 2: Thiết lập thuộc tính cho điều khiển trong cửa sổ *Properties* như sau:

- listBox1:  
Thuộc tính *Name*: Li\_DSSanPham
- label1:  
Thuộc tính *Text*: “Danh sách sản phẩm.”

- label2:  
Thuộc tính *Text*: “Thêm sản phẩm:”
- label3:  
Thuộc tính *Text*: “Thêm sản phẩm:”
- label4:  
Thuộc tính *Text*: “Nhập sản phẩm cần tìm:”
- label5:  
Thuộc tính *Text*: “Vị trí thêm:”
- textbox1:  
Thuộc tính *Name*: txtThemSP
- textbox2:  
Thuộc tính *Name*: txtThemSPViTri
- textbox3:  
Thuộc tính *Name*: txtTimSP
- button1:  
Thuộc tính *Text*: “Thêm”  
Thuộc tính *Name*: btnThem
- button2:  
Thuộc tính *Text*: “Thêm”  
Thuộc tính *Name*: btnThemVT
- button3:  
Thuộc tính *Text*: “Tìm kiếm”  
Thuộc tính *Name*: btnTimKiem
- button4:  
Thuộc tính *Text*: “Xóa sản phẩm”  
Thuộc tính *Name*: btnXoa
- button5:  
Thuộc tính *Text*: “Xóa danh sách sản phẩm”  
Thuộc tính *Name*: btnXoaDS
- combobox1:  
Thuộc tính *Name*: cboViTri

### Bước 3: Viết mã lệnh

- Sự kiện *Form\_Load* cho Form1 như sau:

```
private void Form1_Load(object sender, EventArgs e)
{
    ThietLapViTriComboBox();
}
```

- Viết mã lệnh cho hàm ThietLapViTriComboBox:

```
private void ThietLapViTriComboBox()
{
    cboViTri.Items.Clear();
    int chiso = DSSanPham.Items.Count;
    for (int i = 0; i < chiso; i++)
        cboViTri.Items.Add(i.ToString());
}
```

- Sự kiện *Click* nút lệnh btnThem

```
private void btnThem_Click(object sender, EventArgs e)
{
    if (txtThemSP.Text.Trim() != "")
    {
        DSSanPham.Items.Add(txtThemSP.Text);
        txtThemSP.Text = "";
        ThietLapViTriComboBox();
    }
    else
        MessageBox.Show("Phải nhập tên sản phẩm");
}
```

- Sự kiện *Click* nút lệnh btnThemVT

```
private void btnThemVT_Click(object sender, EventArgs e)
{
    if (txtThemSPViTri.Text.Trim() != " ")
    {
        if (cboViTri.Text != "")
        {
            DSSanPham.Items.Insert(Convert.ToInt32(cboViTri.Text),
                txtThemSPViTri.Text);
            txtThemSPViTri.Text = "";
            ThietLapViTriComboBox();
        }
        else
            MessageBox.Show("Phải chọn vị trí thêm hợp lệ");
    }
    else
        MessageBox.Show("Phải nhập tên sản phẩm ");
}
```

- Sự kiện *Click* nút lệnh btnXoaDS

```
private void btnXoaDS_Click(object sender, EventArgs e)
{
    if (DSSanPham.Items.Count > 0)
        DSSanPham.Items.Clear();
    else
        MessageBox.Show("Danh sách sản phẩm chưa có gì");
}
```

- Sự kiện Click nút lệnh btnXoa

```
private void btnXoa_Click(object sender, EventArgs e)
{
    if (DSSanPham.SelectedIndex < 0)
        MessageBox.Show("Chọn sản phẩm muốn xóa ");
    else
    {
        DSSanPham.Items.Remove(DSSanPham.SelectedItem);
        MessageBox.Show("Xóa sản phẩm thành công");
    }
}
```

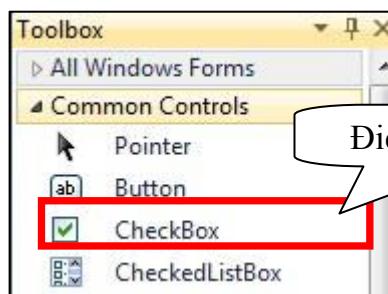
- Sự kiện Click nút lệnh btnTimKiem

```
private void btnTimKiem_Click(object sender, EventArgs e)
{
    if (txtTimSP.Text != "")
    {
        if (DSSanPham.Items.Contains(txtTimSP.Text) == true)
            MessageBox.Show("Tìm thấy sản phẩm");
        else
            MessageBox.Show("Không tìm thấy sản phẩm");
    }
    else
        MessageBox.Show("Nhập tên sản phẩm cần tìm");
}
```

### 3.5. Điều khiển CheckBox và RadioButton

#### 3.5.1. CheckBox

CheckBox là điều khiển cho phép trình bày các giá trị để lựa chọn trên form, người dùng có thể chọn một hoặc nhiều giá trị cùng lúc. Điều khiển CheckBox được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.27



Hình 3.27: Điều khiển CheckBox trên cửa sổ Toolbox

- Một số thuộc tính thường dùng của *CheckBox*:

Bảng 3.10: Bảng mô tả các thuộc tính của *CheckBox*

Thuộc tính	Mô tả
<i>AutoCheck</i>	Mang giá trị True hoặc False, nếu là giá trị True thì cho phép người dùng nhấp chuột để chọn, nếu là False thì không cho phép người dùng nhấp chuột chọn.
<i>Checked</i>	Thiết lập cho điều khiển <i>CheckBox</i> được lựa chọn hoặc không. Thiết lập giá trị True là điều khiển được chọn, nếu thiết lập False là điều khiển không được chọn.
<i>CheckState</i>	Thường dùng để kiểm tra tình trạng <i>CheckBox</i> có được chọn hay không. Mang 3 giá trị <i>Unchecked</i> , <i>Checked</i> , và <i>Indeterminate</i> . <i>Checked</i> : Điều khiển đang được chọn <i>UnChecked</i> : Điều khiển không được chọn <i>Indeterminate</i> : Điều khiển ở trạng thái không hoạt động
<i>Text</i>	Chuỗi văn bản hiển thị bên cạnh <i>CheckBox</i>
<i>ThreeState</i>	Mang giá trị True hoặc False; Nếu là True thì cho phép <i>CheckBox</i> có 3 trạng thái: <i>Checked</i> , <i>UnChecked</i> , <i>Indeterminate</i> .
<i>RightToLeft</i>	Mang giá trị Yes hoặc No; Cho biết chuỗi văn bản hiển thị (thuộc tính <i>Text</i> ) nằm bên trái hay bên phải của <i>CheckBox</i>

- Sự kiện thường sử dụng của *CheckBox*:

Sự kiện quan trọng và thường xuyên sử dụng của *CheckBox* là sự kiện *Click* và *CheckedChange*. Hai sự kiện này của *CheckBox* cho biết khi nhấp chuột chọn thì *CheckBox* sẽ ở trạng thái chọn (*Checked*) hay ở trạng thái không chọn (*UnChecked*).

Lập trình viên có thể kiểm tra *CheckBox* đang ở trạng thái nào bằng cách kiểm tra thuộc tính *Checked* hoặc *UnChecked* của *CheckBox*.

Ví dụ 6: Thiết kế chương trình như hình 3.28 và thực hiện các yêu cầu chức năng sau:

Hình 3.28: Giao diện form Thông tin sinh viên ví dụ 6

Yêu cầu:

- Khi nhấn nút “Lưu” sẽ xuất hiện MessageBox hiển thị thông tin gồm: Họ tên sinh viên, lớp, và các môn học sinh viên chọn.
- Khi nhấn nút “Thoát” sẽ đóng chương trình.

Hướng dẫn:

Bước 1: Thiết kế giao diện form như hình 3.29

Hình 3.29: Giao diện ban đầu form Thông tin sinh viên ví dụ 6

## Bước 2: Thiết lập thuộc tính điều khiển cho form trong cửa sổ *Properties*

- label1:
  - Thuộc tính Text: “Thông tin sinh viên”
  - Thuộc tính Size: 16
- label2:
  - Thuộc tính Text: “Nhập họ tên:”
- label3:
  - Thuộc tính Text: “Lớp:”
- label4:
  - Thuộc tính Text: “Danh sách môn học tự chọn:”
- textBox1:
  - Thuộc tính Name: txtHoTen
- textBox2:
  - Thuộc tính Name: txtLop
- checkBox1:
  - Thuộc tính Text: “Lập trình C#”
  - Thuộc tính Name: chkCSharp
- checkBox2:
  - Thuộc tính Text: “Mạng máy tính”
  - Thuộc tính Name: chkMang
- checkBox3:
  - Thuộc tính Text: “Xử lý ảnh”
  - Thuộc tính Name: chkXLAnh
- checkBox4:
  - Thuộc tính Text: “Lập trình Web”
  - Thuộc tính Name: chkWeb
- button1:
  - Thuộc tính Text: “Lưu”
  - Thuộc tính Name: btnLuu
- button2:
  - Thuộc tính Text: “Thoát”
  - Thuộc tính Name: btnThoat
- Sự kiện *Click* của nút lệnh btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện Click nút lệnh btnLuu

```
private void btnLuu_Click(object sender, EventArgs e)
{
    string str = "";
    str = str + "Họ tên: " + txtHoTen.Text + "\nLớp: "
        + txtLop.Text + "\nDanh sách môn: ";
    if (chkCSharp.Checked == true)
        str = str + "Lập trình C#, ";
    if (chkMang.Checked == true)
        str = str + "Mạng máy tính, ";
    if (chkWeb.Checked == true)
        str = str + "Lập trình Web, ";
    if (chkXLAnh.Checked == true)
        str = str + "Xử lý ảnh, ";
    MessageBox.Show(str);
}
```

### 3.5.2. RadioButton

*RadioButton* là điều khiển cho phép trình bày các giá trị để lựa chọn trên form. Điểm khác biệt của *RadioButton* với *CheckBox* là người dùng chỉ có thể có một sự lựa chọn với các *RadioButton* cùng nhóm, nghĩa là *RadioButton* có thể phân vào những nhóm khác nhau; Với những *RadioButton* cùng nhóm, khi người dùng nhập chọn một *RadioButton* thì đồng thời các *RadioButton* khác trong nhóm sẽ lập tức bỏ chọn. Điều khiển *RadioButton* được đặt trong nhóm *Common Controls* của cửa sổ *Toolbox* như hình 3.30



Hình 3.30: Điều khiển RadioButton trên cửa sổ Toolbox

- Một số thuộc tính thường dùng của *RadioButton*:

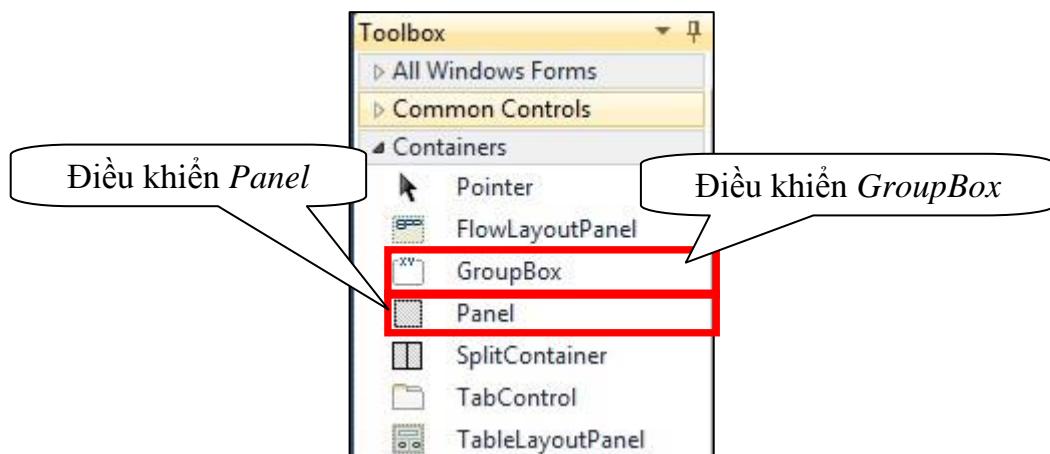
Bảng 3.11 Bảng mô tả các thuộc tính của *RadioButton*

Thuộc tính	Mô tả
Checked	Thiết lập cho điều khiển <i>RadioButton</i> được lựa

	chọn hoặc không. Thiết lập giá trị True là điều khiển được chọn, nếu thiết lập False là điều khiển không được chọn.
<i>Text</i>	Chuỗi văn bản hiển thị bên cạnh <i>RadioButton</i>
<i>RightToLeft</i>	Mang giá trị Yes hoặc No; Cho biết chuỗi văn bản hiển thị (thuộc tính <i>Text</i> ) nằm bên trái hay bên phải của <i>RadioButton</i>

Ngoài việc thiết lập thuộc tính *Checked* để thay đổi tình trạng của *RadioButton*, lập trình viên cũng có thể sử dụng để kiểm tra tình trạng của *RadioButton*. Nếu *RadioButton* được chọn thì thuộc tính *Checked* sẽ là *True*, nếu *RadioButton* không được chọn sẽ là *False*.

Tất cả *RadioButton* được chứa trong điều khiển thuộc nhóm *Containers* để tạo thành một nhóm, thông thường sử dụng điều khiển *Panel* hay *GroupBox* như hình 3.31. Điều này cho phép người dùng chỉ có thể chọn duy nhất một *RadioButton* trong nhóm.

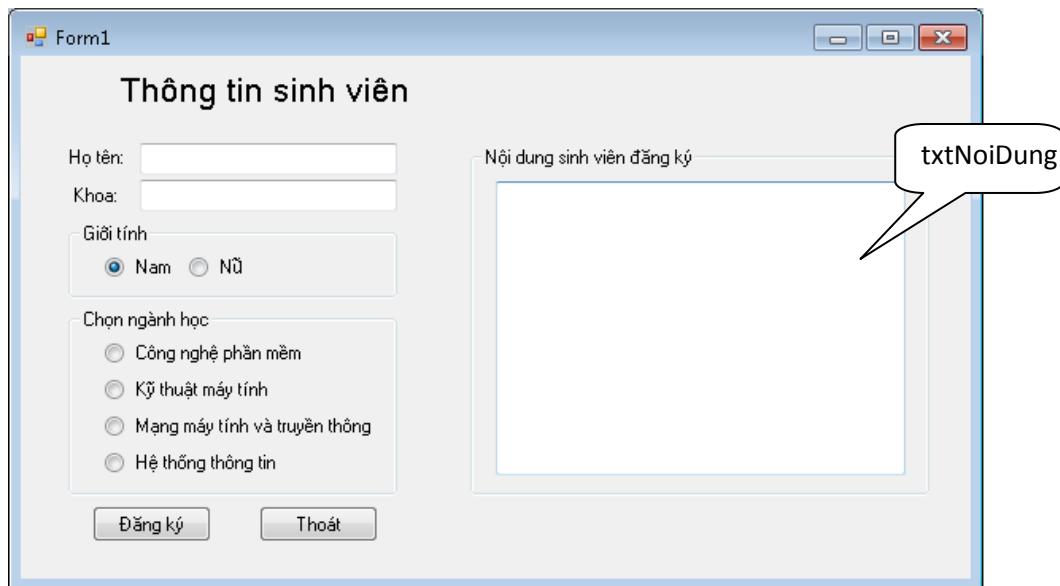


Hình 3.31: Điều khiển *GroupBox* trên cửa sổ *Toolbox*

➤ Sự kiện thường sử dụng của *RadioButton*:

Tương tự như *CheckBox*, sự kiện thường dùng của *RadioButton* cũng là hai sự kiện *Click* và *CheckedChange*. Sự kiện *Click* thực hiện khi người dùng nhấp chuột vào *RadioButton*, còn sự kiện *CheckedChange* thực hiện khi tình trạng của *RadioButton* bị thay đổi từ trạng thái chọn (*Checked*) sang trạng thái không chọn (*UnChecked*) và ngược lại.

Ví dụ 7: Thiết kế form Thông tin sinh viên như hình 3.32 và thực hiện yêu cầu chức năng



Hình 3.32: Giao diện form thông tin sinh viên ví dụ 7

Yêu cầu:

- Khi nhấn nút “Đăng ký” thì nội dung người dùng nhập sẽ hiển thị trong điều khiển *TextBox* có tên là txtNoiDung. Ví dụ: nhập họ tên là “Nguyễn Văn A”, Khoa là “Công nghệ thông tin”, giới tính chọn là “Nam”, chuyên ngành học là “Công nghệ phần mềm” thì sẽ hiển thị trên *TextBox* là:

Họ tên: Nguyễn Văn A

Khoa: Công nghệ thông tin

Giới tính: Nam

Chuyên ngành: Công nghệ phần mềm

- Khi nhấn nút “Thoát” chương trình sẽ đóng lại

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 3.33



Hình 3.33: Giao diện đầu tiên form Thông tin sinh viên ví dụ 7

Bước 2: Thiết lập thuộc tính cho các điều khiển trong cửa sổ *Properties*

- label1:
  - Thuộc tính *Text*: “Thông tin sinh viên”
  - Thuộc tính *Size*: 16
- label2:
  - Thuộc tính *Text*: “Họ tên:”
- label3:
  - Thuộc tính *Text*: “Khoa:”
- textBox1:
  - Thuộc tính *Name*: txtHoTen
  - Thuộc tính *Text*: “”
- textBox2:
  - Thuộc tính *Name*: txtKhoa
  - Thuộc tính *Text*: “”
- textBox3:
  - Thuộc tính *Name*: txtNoiDung
  - Thuộc tính *Text*: “”
  - Thuộc tính *Multiline*: True
- groupBox1:
  - Thuộc tính *Text*: “Giới tính”
- groupBox2:
  - Thuộc tính *Text*: “Chọn ngành học”
- groupBox3:
  - Thuộc tính *Text*: “Nội dung sinh viên đăng ký”
- button1:
  - Thuộc tính *Name*: btnDangKy
- button2:
  - Thuộc tính *Name*: btnThoat
- radioButton1:
  - Thuộc tính *Name*: radNam
  - Thuộc tính *Checked*: True
- radioButton2:
  - Thuộc tính *Name*: radNu
- radioButton3:
  - Thuộc tính *Name*: radCNPM
  - Thuộc tính *Checked*: True
- radioButton4:

Thuộc tính *Name*: radKTMT

- radioButton5:

Thuộc tính *Name*: radMMTTT

- radioButton6:

Thuộc tính *Name*: radHTTT

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* nút lệnh btnDangKy:

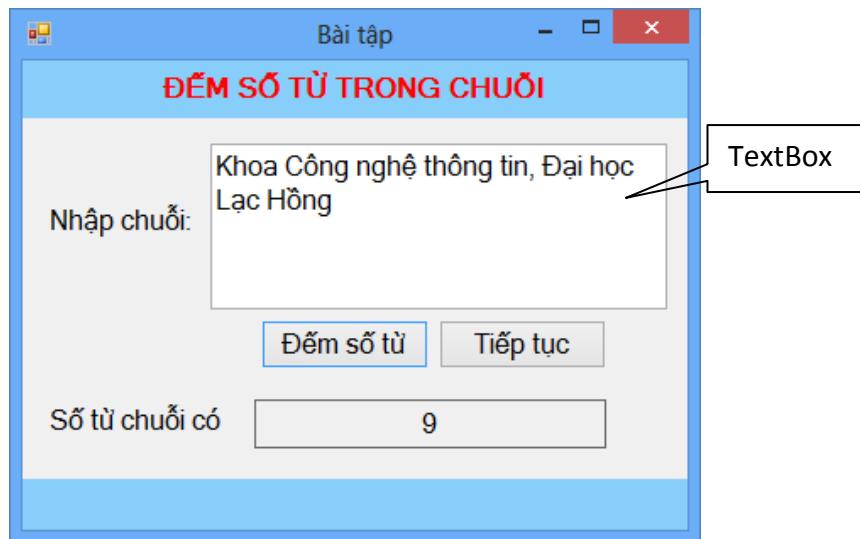
```
private void btnDangKy_Click(object sender, EventArgs e)
{
    txtNoiDung.Text = "";
    string str = "";
    str = "Họ tên: " + txtHoTen.Text + "\r\nKhoa: " +
          txtKhoa.Text + "\r\n";
    if (radNam.Checked == true)
        str = str + "Giới tính: Nam\r\n";
    if (radNu.Checked == true)
        str = str + "Giới tính: Nữ\r\n";
    str = str + "Chuyên ngành: ";
    if (radKTMT.Checked == true)
        str = str + " Kỹ thuật máy tính";
    if (radMMTTT.Checked == true)
        str = str + " Mạng máy tính và truyền thông";
    if (radHTTT.Checked == true)
        str = str + " Hệ thống thông tin";
    if (radCNPM.Checked == true)
        str = str + " Công nghệ phần mềm";
    txtNoiDung.Text = str;
}
```

- Sự kiện *Click* của nút lệnh btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

### 3.6. Bài tập cuối chương

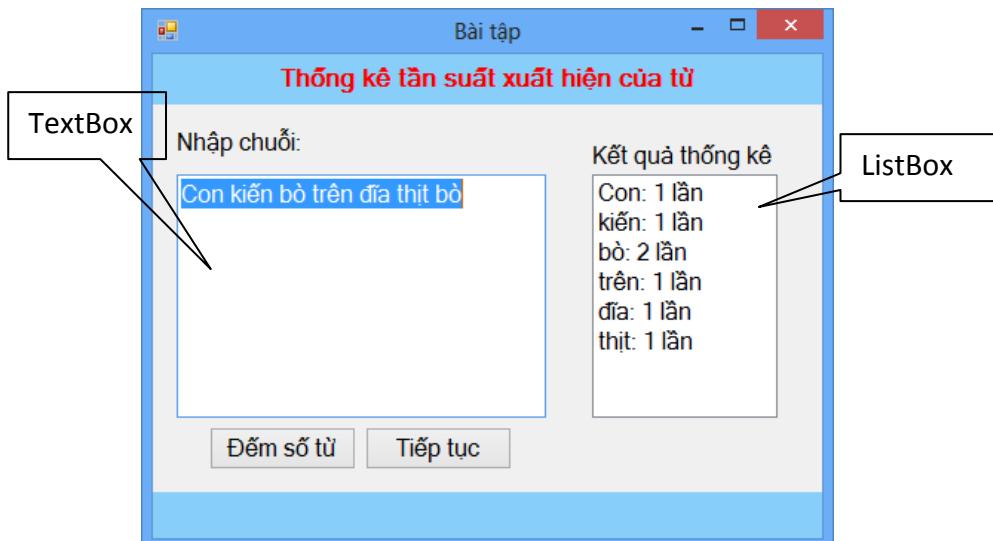
Câu 1: Viết chương trình đếm số từ của chuỗi trong TextBox có giao diện như hình 3.34.



Hình 3.34: Giao diện chương trình đếm từ

- Button “Đếm số từ” sẽ cho biết chuỗi người dùng nhập vào có bao nhiêu từ (xem các từ cách nhau bởi một khoảng trắng).
- Button “Tiếp tục” xóa trống các textbox về trạng thái như lúc đầu.

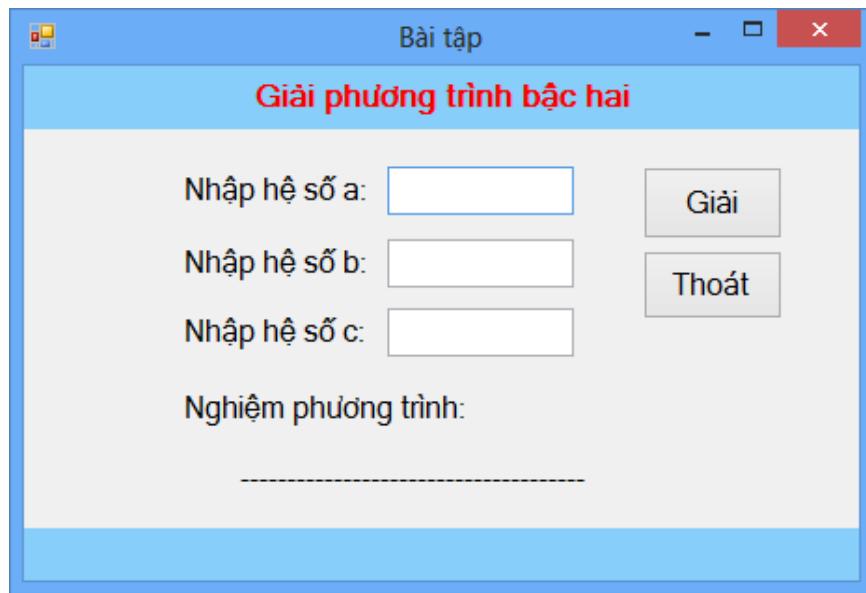
Câu 2: Viết chương trình có giao diện như hình 3.35, giúp thống kê tần suất xuất hiện các từ nhập vào.



Hình 3.35: Giao diện chương trình đếm tần suất xuất hiện của từ

- Button “Đếm số từ” sẽ hiển thị tần suất xuất hiện của từ bên ListBox (xem các từ cách nhau bởi một khoảng trắng).
- Button “Tiếp tục” xóa trống các TextBox và ListBox về trạng thái như lúc đầu.

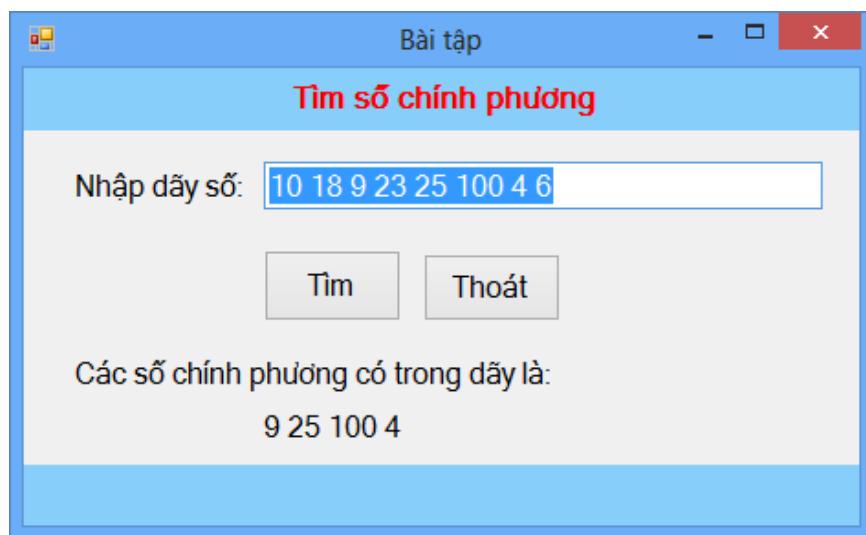
Câu 3: Viết chương trình giải phương trình bậc hai bao gồm các điều khiển: Label, TextBox và Button như giao diện như hình 3.36.



Hình 3.36: Giao diện chương trình giải phương trình bậc hai

- Button “Giải” sẽ hiển thị kết quả của phương trình tại Label “-----”. Sau khi hiển thị kết quả các TextBox sẽ được thiết lập rỗng để người dùng có thể nhập hệ số của phương trình mới.
- Button “Thoát” sẽ thoát chương trình.

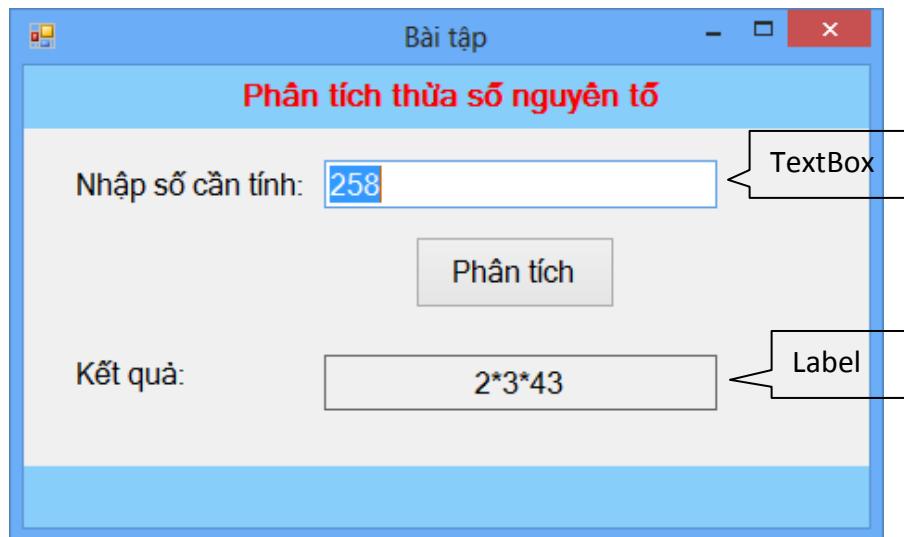
Câu 4: Viết chương trình tìm kiếm số chính phương trong một dãy số nhập trong TextBox (các số cách nhau bởi khoảng trắng). Giao diện gồm các điều khiển: TextBox, Label, Button như hình 3.37.



Hình 3.37: Giao diện chương trình tìm kiếm số chính phương

- Button “Tìm” sẽ hiển thị các số chính phương có trong dãy số đã nhập trong TextBox.
- Button “Thoát” sẽ thoát chương trình.

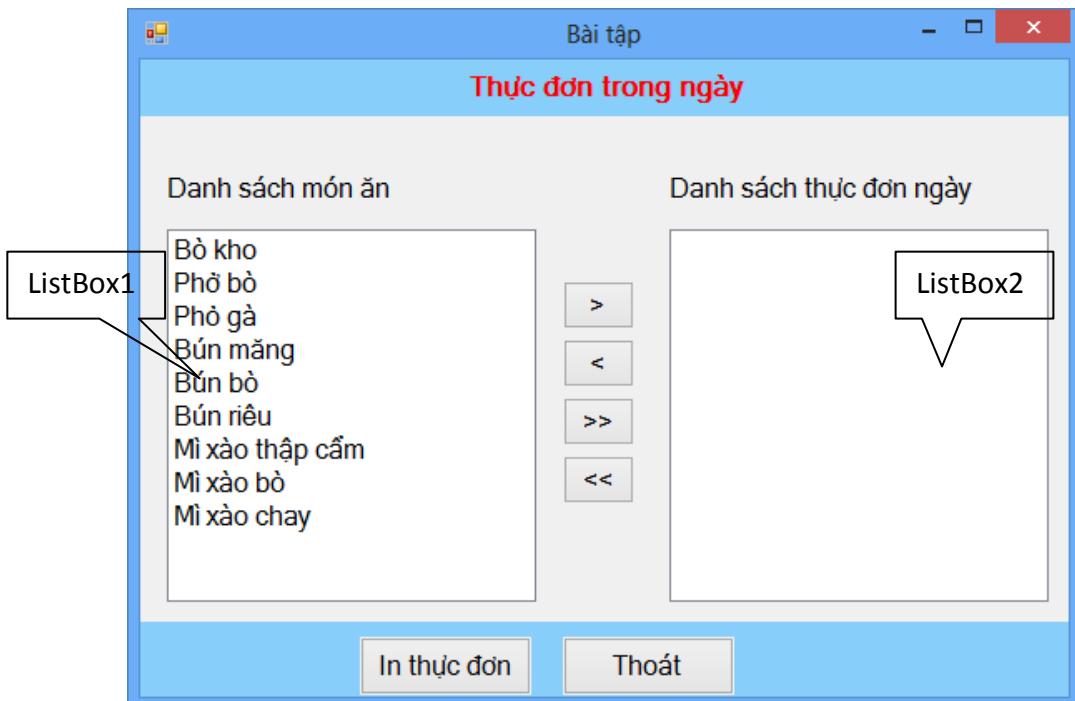
Câu 5: Viết chương trình phân tích một số thành các thừa số nguyên tố. Giao diện gồm các điều khiển: TextBox, Label, Button như hình 3.38.



Hình 3.38: Giao diện chương trình phân tích thừa số nguyên tố

- Button “Phân tích” sẽ phân tích số người dùng trong TextBox thành thừa số nguyên tố hiển thị trong Label.

Câu 6: Viết chương trình cho phép người dùng lựa chọn các món ăn để làm thành thực đơn các món ăn trong ngày mà cửa hàng thức ăn có bán. Giao diện gồm các điều khiển: ListBox, Label, Button như hình 3.39.

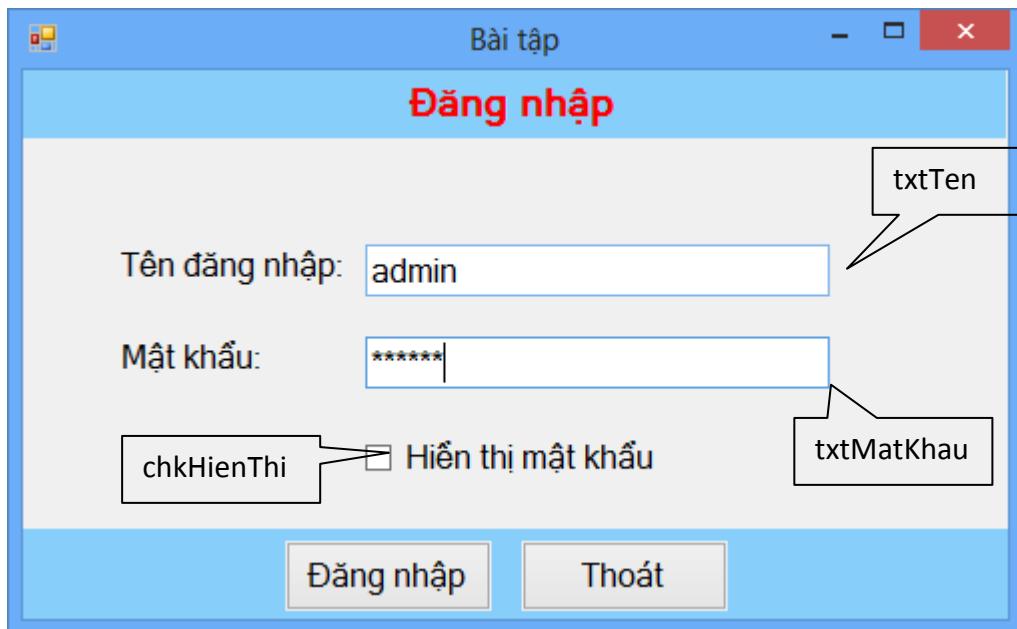


Hình 3.39: Giao diện chương trình chọn thực đơn thức ăn trong ngày

- ListBox1: Hiển thị danh sách tất cả các món ăn
- Button “>”: chuyển một món được chọn từ ListBox1 qua ListBox2
- Button “>>”: chuyển tất cả các món từ ListBox1 qua ListBox2
- Button “<”: xóa món ăn được chọn trong ListBox2
- Button “<<”: xóa tất cả món ăn trong ListBox2

- Button “In thực đơn” sẽ hiển thị hộp thoại MessageBox với nội dung là các món ăn đã chọn hiển thị trong ListBox2
- Button “Thoát” : đóng chương trình.

Câu 7: Viết chương trình tạo form đăng nhập có giao diện như hình 3.40.



Hình 3.40: Giao diện đăng nhập người dùng

Yêu cầu:

Người dùng nhập tên tài khoản và mật khẩu trên TextBox txtTen và txtMatkhau.

- Nếu CheckBox chkHienThi không được chọn thì ở TextBox txtMatkhau sẽ hiện dấu \* với mỗi ký tự người dùng gõ.
- Nếu CheckBox chkHienThi được chọn thì TextBox txtMatkhau sẽ hiển thị đúng các ký tự người dùng đã gõ.

Sau khi nhập xong tên đăng nhập và mật khẩu. Người dùng nhấn Button “Đăng nhập”:

- Nếu tên tài đăng nhập là “admin” và mật khẩu là “123456” thì xuất hiện MessageBox với nội dung: “Đăng nhập thành công”.
- Nếu tên tài khoản và mật khẩu không phải là “admin” và “123456” thì xuất hiện MessageBox với nội dung: “Không đăng nhập thành công”.

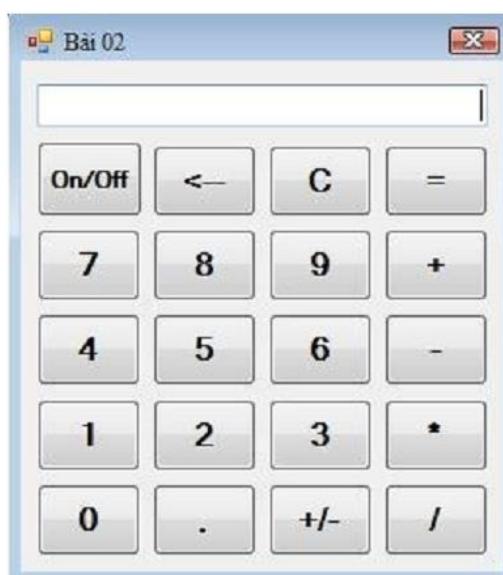
Người dùng nhấn Button “Thoát” để đóng chương trình

Câu 8: Viết chương trình tạo Calculator như hình 3.41.

Yêu cầu:

- Khi nhấn F5 để bắt đầu thực thi chương trình, hiển thị messagebox chứa thông tin: Họ tên sinh viên – mã số sinh viên. Khi nhấn chọn Button “OK” trên messagebox thì hiển thị form Máy tính. Khi nhấn chọn Button “Cancel” trên messagebox thì đóng chương trình.
- Xử lý Button “ON/OFF”

- Khi chưa nhấn chọn Button “ON/OFF” (hoặc số lần nhấn chọn là chẵn) thì tắt máy, nghĩa là, không cho tác động lên các thành phần còn lại trên form Máy tính.
- Nhấn chọn một lần Button “ON/OFF” (hoặc số lần nhấn chọn là lẻ) thì mở máy, nghĩa là cho tác động lên các thành phần còn lại trên form Máy tính, đồng thời xuất hiện cursor nháy ở lề phải khung hiển thị sẵn sàng làm toán.
- Xử lý khung hiển thị
  - Không hiển thị số được gõ từ bàn phím của máy tính mà chỉ hiển thị số khi nhấn chọn vào các Button số trên form Máy tính. Nhấn chọn Button nào thì hiển thị số tương ứng với nhãn của Button đó, trong đó, Button “.” để nhập dấu cách phân thập phân của số thực A và B; nút “– n” để gõ dấu âm đứng trước số thực A và B.
- Xử lý tính toán + , - , x , /
  - Thực hiện phép toán giữa 2 số thực A và B. Số A là số nhập trước khi nhấn chọn Button phép toán, nếu khung hiển thị đang để trống thì hiểu số A có giá trị 0. Số B là số nhập sau khi nhấn chọn Button phép toán.
- nhấn chọn Button “=” để xuất kết quả phép toán lên khung hiển thị.
- Mỗi lần nhấn chọn Button “←” để xoá một kí số, xóa từ bên phải sang.
- nhấn chọn Button “C: để xoá tất cả trong khung hiển thị.



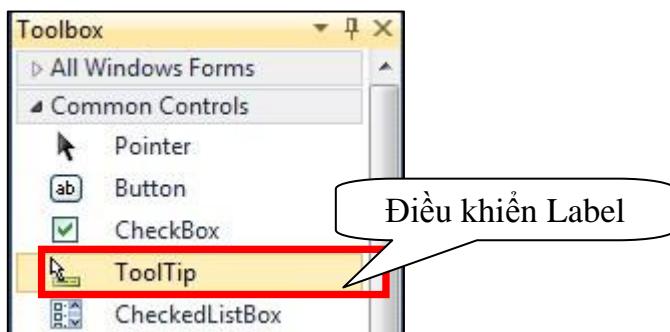
Hình 3.41: Giao diện Calculator

## CHƯƠNG 4: CÁC ĐIỀU KHIỂN ĐẶC BIỆT

### 4.1. Điều khiển Tooltip, HelpProvider, ErrorProvider

#### 4.1.1. Điều khiển Tooltip

Điều khiển *Tooltip* là điều khiển cho phép hiển thị các thông tin chú thích khi người dùng đưa chuột qua các điều khiển có thiết lập *Tooltip*. Điều khiển *Tooltip* được đặt trong nhóm Common Controls của cửa sổ Toolbox như hình 4.1.



Hình 4.1: Điều khiển *Tooltip*

- Một số thuộc tính thường dùng của *Tooltip*:

Bảng 4.1: Bảng mô tả các thuộc tính của *Tooltip*

Thuộc tính	Mô tả
Active	Mang giá trị True hoặc False, nếu thiết lập True thì <i>Tooltip</i> có hiệu lực hiển thị thông báo, nếu mang giá trị False thì <i>Tooltip</i> không hiển thị được thông báo.
AutomaticDelay	Thiết lập thời gian xuất hiện <i>Tooltip</i> khi vừa đưa chuột đến điều khiển, thời gian tính bằng mili giây
AutoPopDelay	Thời gian hiển thị <i>Tooltip</i> cho đến khi kết thúc khi người dùng đã đưa chuột đến điều khiển, thời gian tính bằng mili giây
<i>IsBalloon</i>	Quy định kiểu hiển thị của <i>Tooltip</i> . Nếu thiết lập False kiểu hiển thị <i>Tooltip</i> :  Nếu thiết lập True kiểu hiển thị <i>Tooltip</i> : 

<i>ReshowDelay</i>	Thời gian mà Tooltip tắt từ khi người dùng đưa chuột ra khỏi điều khiển, thời gian tính bằng mili giây
<i>ShowAlways</i>	
<i>ToolTipIcon</i>	Biểu tượng xuất hiện bên cạnh chuỗi khai báo trong thuộc tính <i>ToolTipTitle</i>
<i>ToolTipTitle</i>	Chuỗi hiển thị bên cạnh biểu tượng <i>ToolTipIcon</i>
<i>UseAnimation</i>	Thiết lập hiệu ứng ảnh động được biểu diễn khi Tooltip được hiển thị
<i>UseFading</i>	Thiết lập hiệu ứng mờ dần được biểu diễn khi Tooltip hiển thị

- Một số phương thức thường dùng của *Tooltip*:

Bảng 4.2: Bảng mô tả các phương thức của *Tooltip*

Phương thức	Mô tả
<i>SetTooltip()</i>	Thiết lập chuỗi hiển thị của Tooltip trên điều khiển
<i>GetTooltip()</i>	Lấy nội dung chuỗi hiển thị trên Tooltip
<i>Clear()</i>	Loại bỏ tất cả <i>ToolTipText</i> cho các điều khiển trên form

Ví dụ 4.1: Viết chương trình tạo giao diện form đăng nhập và thực hiện yêu cầu chức năng như hình 4.2.



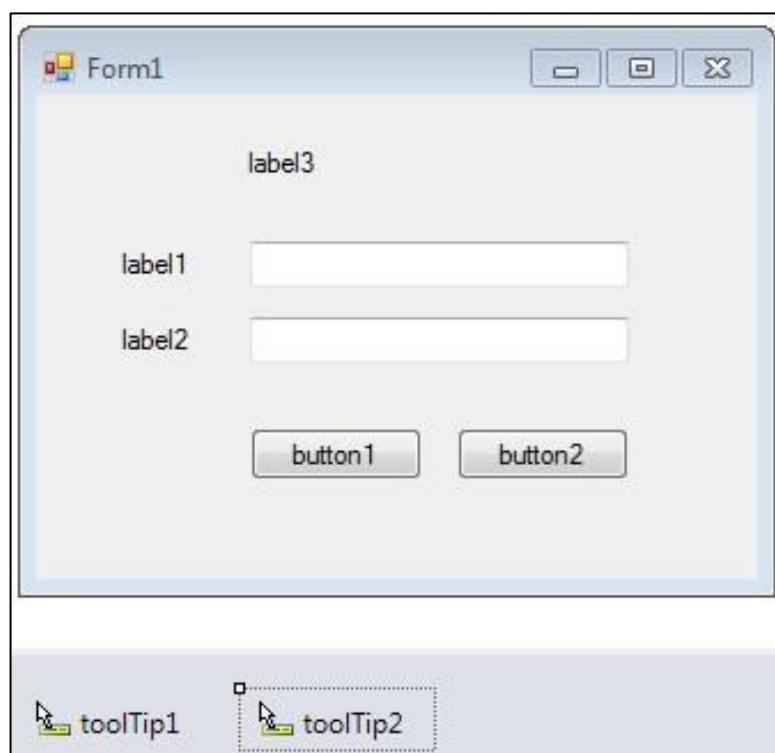
Hình 4.2: Giao diện form đăng nhập ví dụ 4.1

Yêu cầu:

- Khi rê chuột vào TextBox1: hiển thị dòng ghi chú “Nhập chuỗi ký không dấu, không khoảng trắng”
- Khi rê chuột vào TextBox2: hiển thị dòng ghi chú “Nhập ít nhất 6 ký tự, nhiều nhất 10 ký tự”
- Khi nhấn nút “Đăng nhập”: hiển thị MessageBox với nội dung “Bạn đăng nhập thành công”
- Khi nhấn nút “Thoát”: thoát khỏi chương trình

Hướng dẫn:

Bước 1: Thiết kế giao diện form ban đầu như hình 4.3. Sau đó kéo 2 *Tooltip* từ cửa sổ Toolbox thả vào Form1



Hình 4.3: Giao diện ban đầu form đăng nhập ví dụ 1

Bước 2: Thiết lập giá trị thuộc tính điều khiển trong cửa sổ Properties

- Form1:

Thuộc tính *Text*: “Đăng nhập”

- label3:

Thuộc tính *Text*: “Đăng nhập”

Thuộc tính *Size*: 16

- label1:

Thuộc tính *Text*: “Tên đăng nhập:”

- label2:

Thuộc tính *Text*: “Mật khẩu:”

- button1:  
Thuộc tính *Name*: btnDangNhap  
Thuộc tính *Text*: “Đăng nhập”
- button2:  
Thuộc tính *Name*: btnThoat  
Thuộc tính *Text*: “Thoát”
- toolTip1:  
Thuộc tính *Name*: ttpDangNhap
- toolTip2:  
Thuộc tính *Name*: ttpMatKhau

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnDangNhap:

```
private void btnDangNhap_Click(object sender, EventArgs e)
{
    MessageBox.Show("Bạn đăng nhập thành công");
}
```

- Sự kiện *Click* của nút btnThoat:

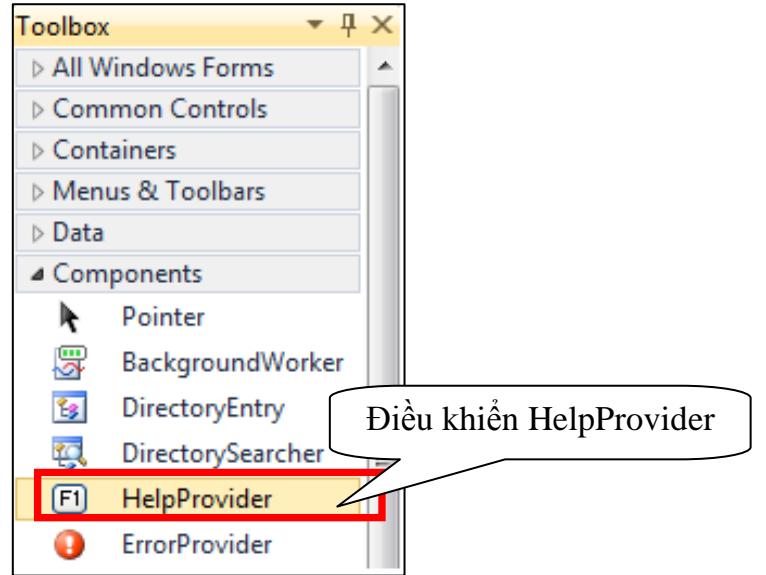
```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    ttpDangNhap.SetToolTip(txtDangNhap,
        "Nhập chuỗi ký không dấu, không khoảng trắng");
    ttpMatKhau.SetToolTip(txtMatKhau,
        "Nhập ít nhất 6 ký tự, nhiều nhất 10 ký tự");
}
```

#### 4.1.2. Điều khiển HelpProvider

Điều khiển *HelpProvider* cung cấp cửa sổ trợ giúp cho điều khiển. Với những ứng dụng có sử dụng *HelpProvider*, người dùng có thể gọi sự trợ giúp bằng cách ấn phím F1. Điều khiển *HelpProvider* được đặt trong nhóm Components của cửa sổ Toolbox như hình 4.4.



Hình 4.4: Điều khiển HelpProvider

- Một số thuộc tính thường dùng của *HelpProvider*:

Bảng 4.3: Bảng mô tả các thuộc tính của *HelpProvider*

Thuộc tính	Mô tả
<i>HelpNamespace</i>	Chỉ định tên tập trình trợ giúp định dạng chm hoặc html.

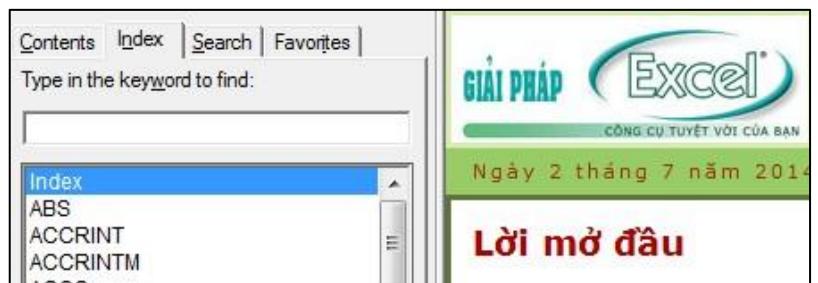
Điểm đặc biệt là khi thêm điều khiển *HelpProvider* vào form thì một số thuộc tính như: *HelpKeyword on helpProvider*, *HelpNavigator on helpProvider*, *HelpString on helpProvider* và *ShowHelp on helpProvider* sẽ xuất hiện trên tất cả các điều khiển có trên form.

Bảng 4.4: Bảng mô tả các thuộc tính của điều khiển khi thêm *HelpProvider*

Thuộc tính	Mô tả
<i>HelpKeyWord</i>	Từ khóa tìm kiếm, từ khóa này là chỉ mục hoặc chủ đề được truyền vào tập tin tìm kiếm. Thuộc tính <i>HelpNavigator</i> sẽ quy định từ khóa này tìm kiếm theo chủ đề hay theo chỉ mục.
<i>HelpNavigator</i>	Thiết lập cách thức hiển thị của tập tin trợ giúp. Gồm các thuộc tính thành viên như: <ul style="list-style-type: none"> <li>- <i>AssociateIndex</i>: Mở tập tin trợ giúp và hiển thị danh sách chỉ mục có ký tự trùng với ký tự đầu tiên trong thuộc tính <i>HelpKeyWord</i>.</li> </ul> Ví dụ 4.2: tập tin trợ giúp là tập tin đuôi .chm và thuộc tính <i>HelpKeyWord</i> chứa từ khóa “HLOOKUP”

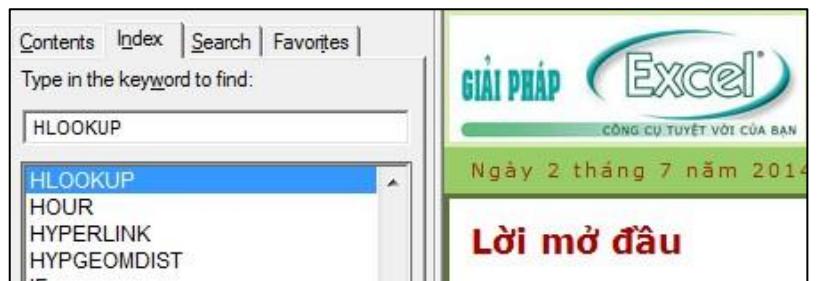


- Find: Giúp hiển thị nội dung trợ giúp có chuỗi ký tự trùng với từ khóa trong thuộc tính HelpKeyword.
- Index: Hiển thị danh mục các chỉ mục trong tập tin trợ giúp.



- KeywordIndex: Hiển thi danh mục là các chỉ mục như từ khóa trong thuộc tính HelpKeyword

Ví dụ 4.3: Từ khóa HelpKeyword là HLOOKUP



- TableOfContents: Hiển thị tất cả các nội dung trong tập tin trợ giúp



- Topic: Hiển thị danh mục các chủ đề trong tập tin trợ giúp, áp dụng với tập tin có hỗ trợ danh mục các chủ đề.
- TopicId: Hiển thị chủ đề có mã trùng với mã chủ đề chỉ định, áp dụng với tập tin có hỗ trợ danh mục các chủ đề và các chủ đề được đánh số.

<i>HelpString</i>	Hiển thị chuỗi trợ giúp cho điều khiển. Nếu thuộc tính <i>HelpProvider</i> không được thiết lập thì khi người dùng nhấn phím F1 sẽ hiển thị chuỗi trợ giúp này
<i>ShowHelp</i>	Nếu thiết lập giá trị True thì cho phép nội dung trợ giúp hiển thị trên một điều khiển nào đó. Nếu thiết lập giá trị False thì không hiển thị được

- Một số phương thức thường dùng của *HelpProvider*:

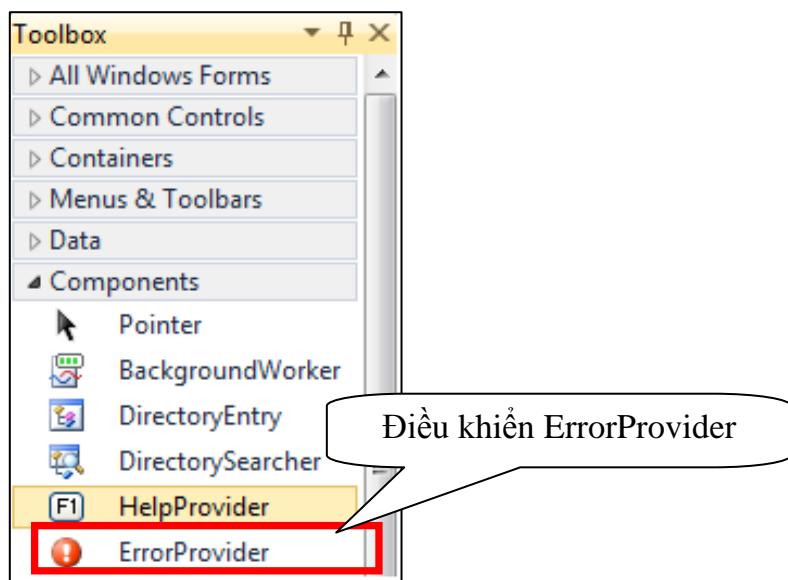
Các thuộc tính HelpKeyWord, HelpNavigator, HelpString, ShowHelp có thể được thiết lập giá trị trong cửa sổ Properties hoặc có thể sử dụng các phương thức như bảng 4.5 để thiết lập giá trị

Bảng 4.5: Bảng mô tả các phương thức của điều khiển khi thêm *HelpProvider*

Phương thức	Mô tả
<i>SetHelpKeyword</i>	Thiết lập giá trị cho thuộc tính HelpKeyword
<i>SetHelpNavigator</i>	Thiết lập giá trị cho thuộc tính HelpNavigator
<i>SetHelpString</i>	Thiết lập giá trị cho thuộc tính HelpString
<i>SetShowHelp</i>	Thiết lập giá trị cho thuộc tính ShowHelp

#### 4.1.3. Điều khiển *ErrorProvider*

*ErrorProvider* giúp báo cho người dùng biết thông tin lỗi của điều khiển trên form. Thông thường khi điều khiển trên form lỗi, *ErrorProvider* sẽ cung cấp một biểu tượng  để thông báo lỗi bên cạnh điều khiển đó. Điều khiển *ErrorProvider* được đặt trong nhóm Components của cửa sổ Toolbox như hình 4.5.



Hình 4.5: Điều khiển *ErrorProvider*

- Một số thuộc tính thường dùng của *ErrorProvider*:

*Bảng 4.6: Bảng mô tả các thuộc tính của HelpProvider*

Thuộc tính	Mô tả
<i>Icon</i>	Chọn biểu tượng thể hiện lỗi của điều khiển
<i>BlinkRate</i>	Tốc độ nhấp nháy của biểu tượng trong thuộc tính <i>Icon</i> . Tốc độ tính theo mili giây
<i>BlinkStyle</i>	Kiểu nhấp nháy của biểu tượng. Nếu thiết lập giá trị NeverBlink thì biểu tượng sẽ hiển thị mà không nhấp nháy.

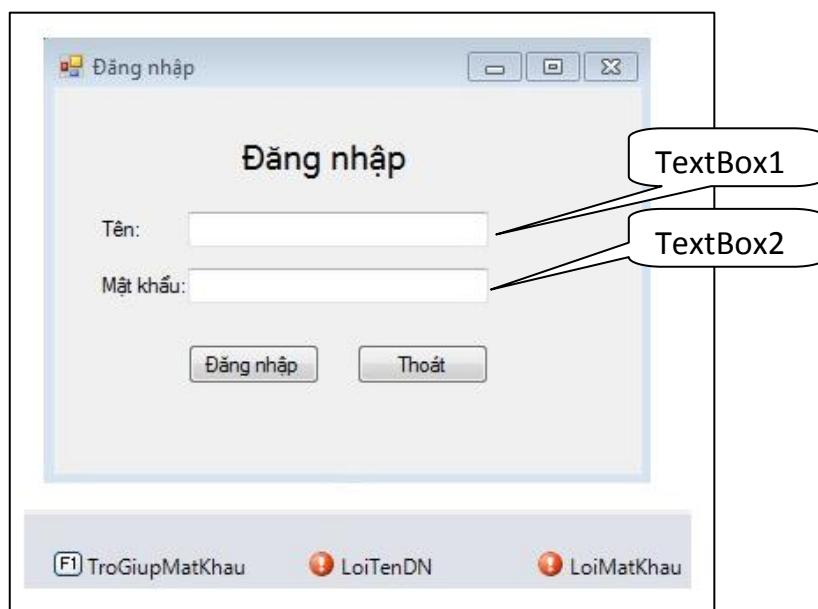
- Một số phương thức thường dùng của *ErrorProvider*:

*Bảng 4.7: Bảng mô tả các phương thức của HelpProvider*

Phương thức	Mô tả
<i>SetError(&lt;Điều khiển&gt;, &lt;Thông báo lỗi&gt;)</i>	Giúp hiển thị lỗi và thông báo lỗi của điều khiển. Thông báo lỗi hiển thị dưới dạng Tooltip
<i>Clear()</i>	Xóa biểu tượng <i>ErrorProvider</i> của điều khiển tương ứng trên form.
<i>GetError()</i>	Lấy chuỗi thông báo lỗi của điều khiển.

Ví dụ 4.4: Viết chương trình tạo form đăng nhập như hình 4.6. Yêu cầu ở Textbox nhập tên tài khoản không được có khoảng trắng; Textbox mật khẩu phải là ký tự số và không được để trống; Hiển thị trợ giúp cho điều khiển Textbox tên tài khoản, cụ thể khi nhấn F1 sẽ hiện trợ giúp tạo mật khẩu từ website:

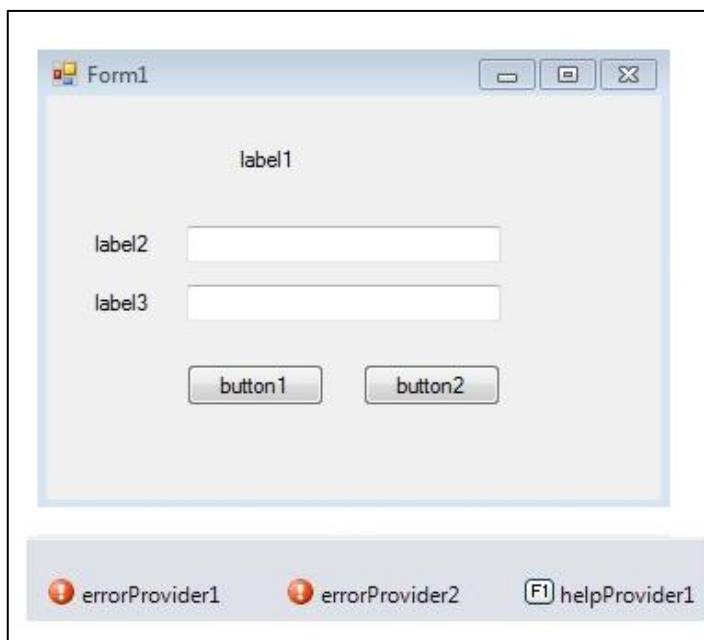
<http://phunutoday.vn/kham-pha-cong-nghe/cac-nguyen-tac-tao-mat-khau-an-toan-33828.html>



*Hình 4.6: Giao diện formm đăng nhập ví dụ 4.4*

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 4.7



Hình 4.7: Giao diện ban đầu form đăng nhập ví dụ 4.4

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties

- Form1:

Thuộc tính Text: “Đăng nhập”

- label1:

Thuộc tính Text: “Đăng nhập”

Thuộc tính Size: 14

- label2:

Thuộc tính Text: “Tên:”

- label3:

Thuộc tính Text: “Mật khẩu:”

- button1:

Thuộc tính Text: “Đăng nhập”

Thuộc tính Name: btnDangNhap

- button2:

Thuộc tính Text: “Thoát”

Thuộc tính Name: btnThoat

- TextBox1:

Thuộc tính Name: txtTen

- TextBox2:

Thuộc tính Name: txtMatKhau

Thuộc tính PasswordChar: \*

- errorProvider1:

Thuộc tính Name: LoiTenDN

- errorProvider2:

Thuộc tính Name: LoiMatKhau

- helpProvider1:

Thuộc tính Name: TroGiupMatKhau

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    //TabIndex giúp thiết lập thứ tự điều khiển khi ấn phím
    //tab
    txtTen.TabIndex = 0;
    txtMatKhau.TabIndex = 1;
    btnDangNhap.TabIndex = 2;
    btnThoat.TabIndex = 3;
    TroGiupMatKhau.SetShowHelp(txtTen, true);
    TroGiupMatKhau.HelpNamespace =
        "http://phunutoday.vn/kham-pha-cong-nghe/cac-nguyen-tac-
        tao-mat-khau-an-toan-33828.html";
}
```

- Sự kiện *TextChanged* của TextBox txtMatKhau:

```
private void txtMatKhau_TextChanged(object sender, EventArgs e)
{
    long so = 0;
    try
    {
        so = Convert.ToInt64(txtMatKhau.Text);
        LoiMatKhau.Clear();
    }
    catch (Exception ex)
    {
        LoiMatKhau.SetError(txtMatKhau, "Phải nhập ký tự số
            và không được để trống");
    }
}
```

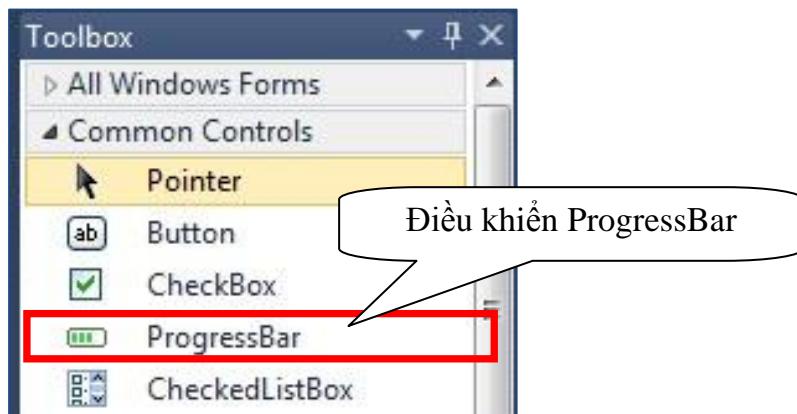
- Sự kiện *TextChanged* của TextBox txtTen:

```
private void txtTen_TextChanged(object sender, EventArgs e)
{
    if (txtTen.Text.IndexOf(' ') != -1)
        LoiTENDN.SetError(txtTen, "Nhập tên không được có
        khoảng trắng");
    else
        LoiTENDN.Clear();
}
```

## 4.2. Điều khiển ProgressBar và Timer

### 4.2.1. Điều khiển ProgressBar

ProgressBar sử dụng để hiển thị thời gian thực hiện của một công việc nào đó. ProgressBar được đặt trong nhóm Common Controls của cửa sổ Toolbox như hình 4.8



Hình 4.8: Điều khiển ProgressBar

- Một số thuộc tính thường dùng của ProgressBar:

Bảng 4.8: Bảng mô tả các thuộc tính của ProgressBar

Thuộc tính	Mô tả
<i>Maximum</i>	Giá trị tối đa của ProgressBar. Khi ProgressBar được lấp đầy nghĩa là ProgressBar đã đạt giá trị <i>Maximum</i> .
<i>Minimum</i>	Giá trị nhỏ nhất của ProgressBar. Khi ProgressBar trống rỗng nghĩa là ProgressBar đang có giá trị <i>Minimum</i> .
<i>Value</i>	Giữ giá trị hiện tại của ProgressBar, giá trị này nằm trong đoạn <i>Minimum</i> và <i>Maximum</i> .
<i>Style</i>	Kiểu hiển thị của ProgressBar.
<i>Step</i>	Lượng giá trị thêm vào Value khi phương thức <i>PerformStep()</i> được gọi.

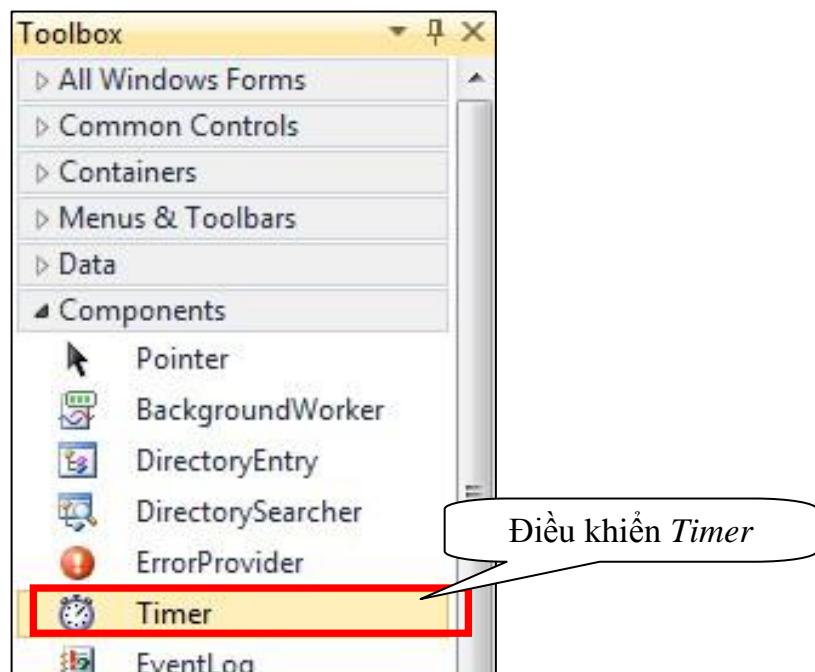
- Một số phương thức thường dùng của *ProgressBar*:

Bảng 4.9: Bảng mô tả các phương thức của *ProgressBar*

Phương thức	Mô tả
<i>PerformStep()</i>	Phương thức giúp tăng <i>ProgressBar</i> . Giá trị tăng là giá trị được thiết lập trong thuộc tính <i>Step</i> .
<i>Increment(&lt;giá trị&gt;)</i>	Phương thức giúp tăng <i>ProgressBar</i> . Giá trị tăng là tham số đầu vào <giá trị> của phương thức.

#### 4.2.2. Điều khiển Timer

Điều khiển *Timer* cho phép thực thi lại một hành động sau một khoảng thời gian xác định. *Timer* được đặt trong nhóm Components của cửa sổ Toolbox như hình 4.9



Hình 4.9: Điều khiển Timer

- Một số thuộc tính thường dùng của *Timer*:

Bảng 4.10: Bảng mô tả các thuộc tính của *Timer*

Thuộc tính	Mô tả
<i>Interval</i>	Thiết lập giá trị là một số nguyên. Giá trị nguyên này là thời lượng của một chu kỳ (tính bằng đơn vị mili giây).
<i>Enable</i>	Thiết lập giá trị <i>True</i> hoặc <i>False</i> . Nếu là giá trị <i>True</i> thì điều khiển <i>Timer</i> hoạt động, nếu là <i>False</i> thì điều khiển <i>Timer</i> không hoạt động.

- Một số phương thức thường dùng của Timer:

Bảng 4.11: Bảng mô tả các phương thức của Timer

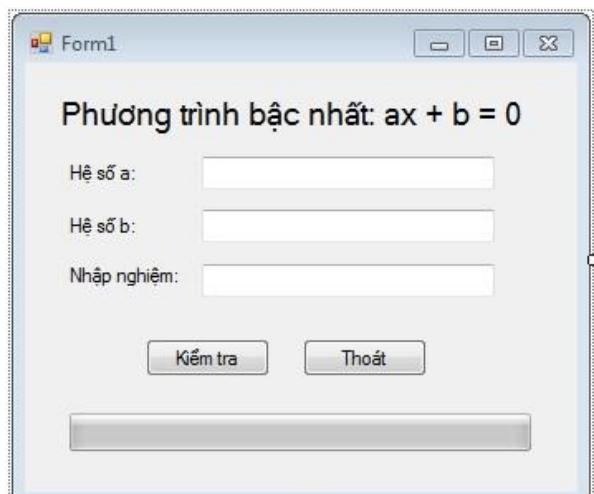
Phương thức	Mô tả
<i>Start()</i>	Kích hoạt điều khiển Timer hoạt động. Phương thức này tương ứng với việc thiết lập giá trị thuộc tính <i>Enable</i> là <i>True</i>
<i>Stop()</i>	Dừng hoạt động của điều khiển Timer. Phương thức này tương ứng với việc thiết lập giá trị thuộc tính <i>Enable</i> là <i>False</i> .

- Một số sự kiện thường dùng của Timer:

Bảng 4.12: Bảng mô tả các sự kiện của Timer

Sự kiện	Mô tả
<i>Tick</i>	Sự kiện được gọi trong mỗi chu kỳ Interval

Ví dụ 4.5: Viết chương trình hỗ trợ người dùng học giải phương trình bậc nhất:  $ax + b = 0$ . Thiết kế giao diện form như hình 4.10.

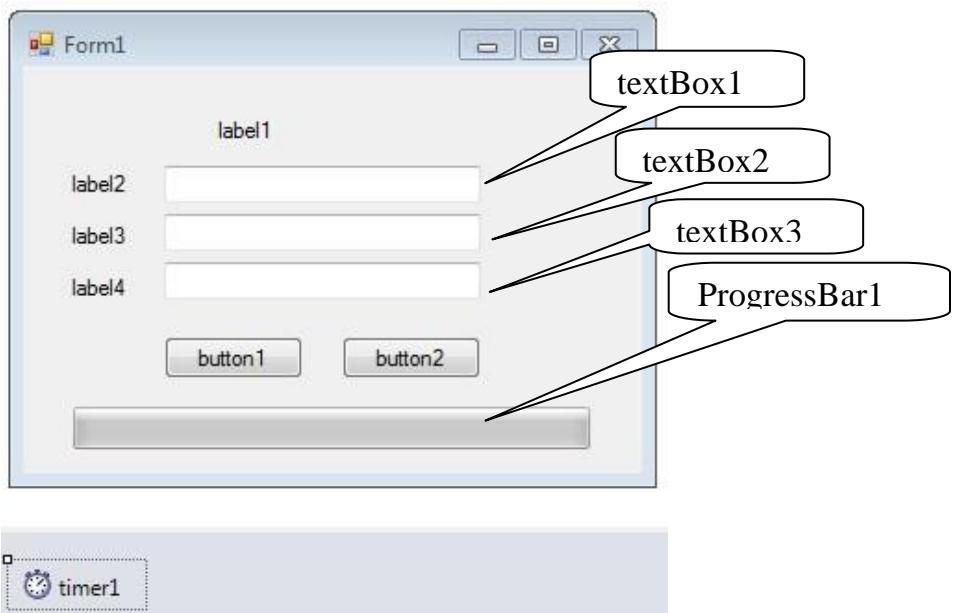


Hình 4.10: Giao diện form giải phương trình bậc nhất

Yêu cầu: Phát sinh ngẫu nhiên hệ số a và b của phương trình. Sau đó người dùng nhập kết quả và ấn nút trả lời. Nếu trả lời đúng thì hiện MessageBox với nội dung “Bạn đã làm đúng”, nếu trả lời sai thì hiện MessageBox với nội dung “Bạn đã trả lời sai”. Lưu ý: Thời gian để hoàn thành giải phương trình là 30 giây hiển thị tương ứng với ProgressBar, trong khoảng thời gian hết 30 giây người dùng không giải được sẽ hiển thị MessageBox với nội dung “Hết giờ làm bài”.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 4.11



Hình 4.11: Giao diện ban đầu form giải phương trình bậc nhất

Bước 2: Thiết lập giá trị thuộc tính trong cửa sổ Properties cho điều khiển

- label1:

Thuộc tính Text: "Phương trình bậc nhất:  $ax + b = 0$ "

Thuộc tính Size: 14

- label2:

Thuộc tính Text: "Hệ số a:"

- label3:

Thuộc tính Text: "Hệ số b:"

- label4:

Thuộc tính Text: "Nhập nghiệm:"

- textBox1:

Thuộc tính Name: txtA

Thuộc tính Enable: False

- textBox2:

Thuộc tính Name: txtB

Thuộc tính Enable: False

- textBox3:

Thuộc tính Name: txtX

- button1:

Thuộc tính Name: btnKiemTra

Thuộc tính Text: "Kiểm tra"

- button2:

- Thuộc tính Name: btnThoat  
 Thuộc tính Text: Thoát
- ProgressBar1:  
 Thuộc tính Name: ProGressTG  
 Thuộc tính Minimum: 0  
 Thuộc tính Maximum: 30000  
 Thuộc tính Step: 1000  
 Thuộc tính Style: Blocks
  - timer1:  
 Thuộc tính Name: ThoiGian  
 Thuộc tính Enable: True  
 Thuộc tính Interval: 1000

Bước 3: Viết mã lệnh cho các điều khiển

- Khai báo các biến

```
private int a=0;
private int b=0;
private float x = 0;
Random rd = new Random();
```

- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    a = rd.Next(-10, 10);
    txtA.Text = a.ToString();
    b = rd.Next(-10, 10);
    txtB.Text = b.ToString();
    x = -b / (float)a;
}
```

- Sự kiện *Tick* của Timer ThoiGian:

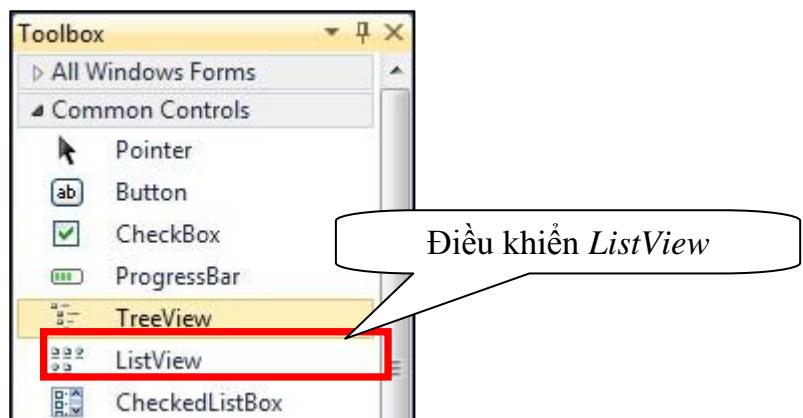
```
private void ThoiGian_Tick(object sender, EventArgs e)
{
    if (ProGressTG.Value == 30000)
    {
        ThoiGian.Enabled = false;
        MessageBox.Show("Hết giờ làm bài");
    }
    ProGressTG.PerformStep();
}
```

- Sự kiện *Click* của nút *btnKiemTra*:

```
private void btnKiemtra_Click(object sender, EventArgs e)
{
    float kq = float.Parse(txtX.Text);
    if (Math.Abs(kq - x) < 0.01)
    {
        MessageBox.Show("Bạn đã làm đúng");
        Close();
    }
    else
        MessageBox.Show("Bạn đã trả lời sai");
}
```

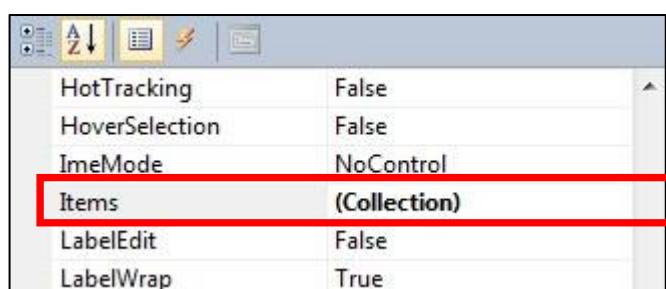
### 4.3. Điều khiển ListView

ListView là điều khiển cho phép hiển thị danh sách các đối tượng. Mỗi đối tượng hiển thị trong ListView được gọi là Item. Item là đối tượng được tạo từ lớp ListViewItem. Mỗi Item có thuộc tính Text là chuỗi ký tự hiển thị ở cột đầu tiên trong ListView, mỗi Item có các SubItem hiển thị ở các cột tiếp theo trong ListView. Điều khiển ListView đặt trong Common Controls của cửa sổ Toolbox như hình 4.12.



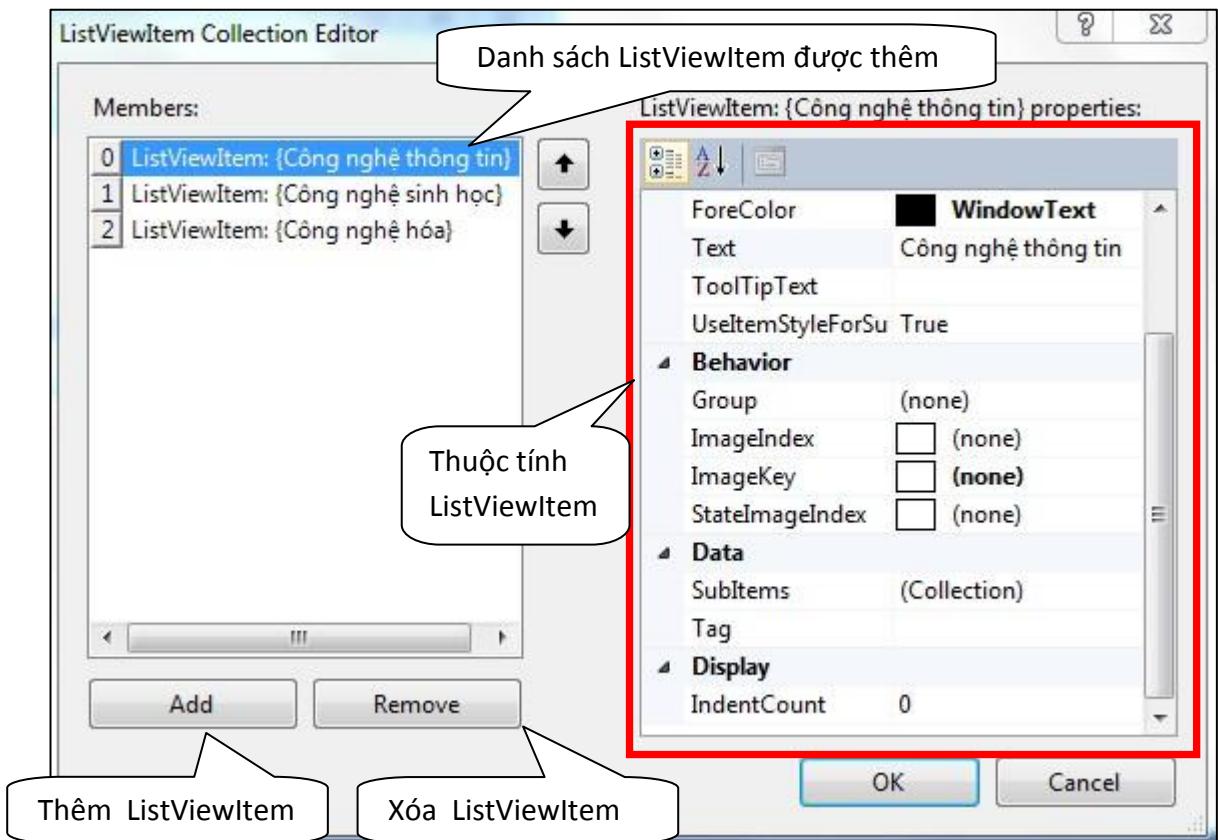
Hình 4.12: Điều khiển ListView

Lập trình viên có thể thêm ListViewItem vào ListView bằng cách chọn thuộc tính Items trong cửa sổ Properties của ListView như hình 4.13.



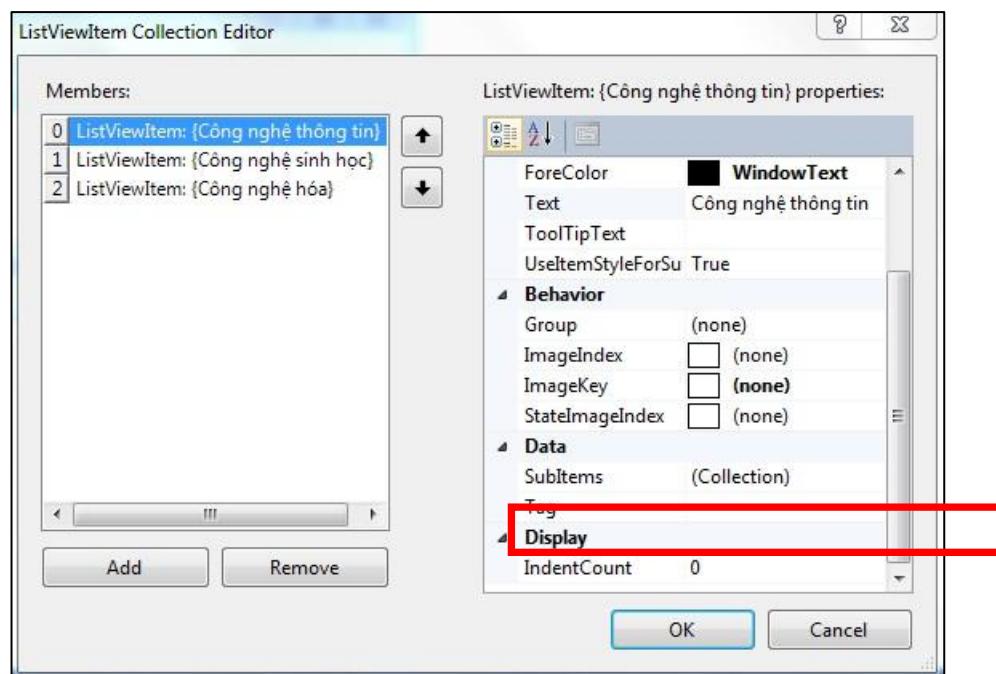
Hình 4.13: Thuộc tính Items trong cửa sổ Properties

Sau khi chọn thuộc tính Items trong cửa sổ Properties, cửa sổ ListViewItem Collection Editor sẽ hiển thị như hình 4.14. Lập trình viên có thể thêm hoặc xóa ListViewItem trong ListView bằng cách nhấn nút Add hoặc Remove.



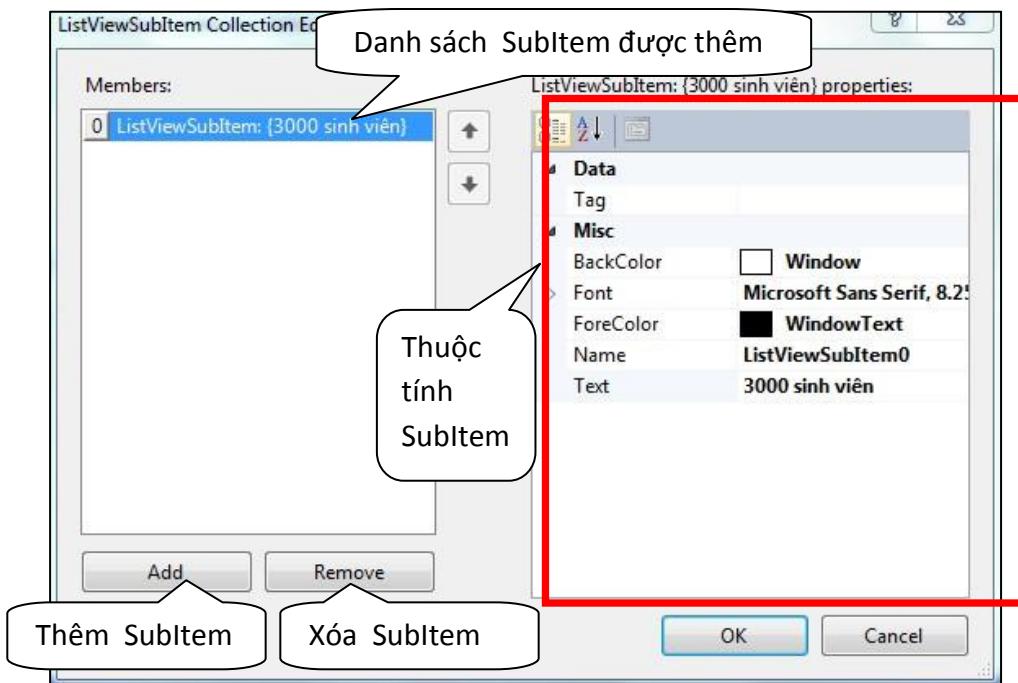
Hình 4.14: Cửa sổ ListViewItem Collection Editor

Thêm các SubItem của Item trong ListViewItem bằng cách chọn thuộc tính SubItems trong cửa sổ ListViewItem Collection Editor như hình 4.15.



Hình 4.15: Thuộc tính SubItems trong cửa sổ ListViewItem Collection Editor

Sau khi chọn thuộc tính SubItems trong cửa sổ *ListViewItem Collection Editor*, cửa sổ *ListViewSubItem Collection Editor* như hình 4.16 được hiển thị cho phép lập trình viên thêm hoặc xóa các SubItem.



Hình 4.16: Cửa sổ *ListViewSubItem Collection Editor*

- Một số thuộc tính thường dùng của *ListView*:

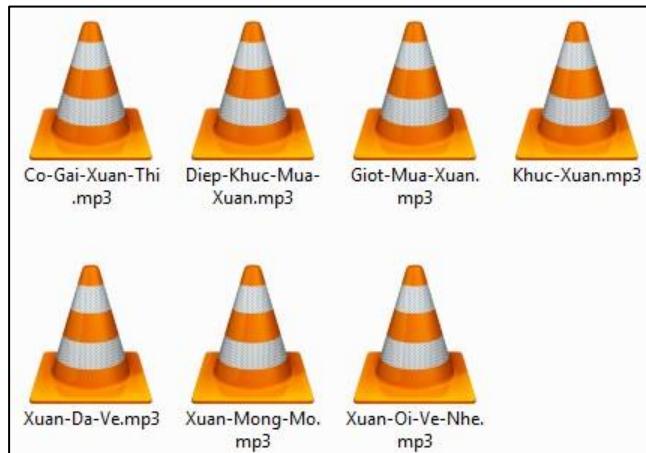
Bảng 4.13: Bảng mô tả các thuộc tính của *ListView*

Thuộc tính	Mô tả
<i>View</i>	<p>Thuộc tính <i>View</i> qui định cách hiển thị các <i>Item</i> trong <i>ListView</i>. Thuộc tính <i>View</i> có 5 giá trị:</p> <ul style="list-style-type: none"> <li>- <i>Detail</i>: Một Icon (Icon lấy từ <i>ImageList</i>) và <i>Text</i> được hiển thị ở cột đầu tiên. Tiếp theo là các <i>SubItem</i> được hiển thị ở các cột tiếp theo. Tuy nhiên để hiển thị Item dạng <i>Detail</i> thì tạo thêm <i>Column Header</i> cho <i>ListView</i>:</li> </ul> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>BackgroundImageTiled False BorderStyle Fixed3D CausesValidation True Checkboxes False <b>Columns (Collection)</b> ContextMenuStrip (none)</p> </div> <p><i>ListView</i> hiển thị Item dạng <i>Detail</i> như sau:</p>

Name	Title
Co-Gai-Xuan-Thi.mp3	Co Gai Xuan Thi
Diep-Khuc-Mua-Xuan.mp3	diep Khuc Mua Xuan
Giot-Mua-Xuan.mp3	Giot Mua Xuan
Khuc-Xuan.mp3	Khuc Xuan
Kia-Xuan-Da-Ve.mp3	Kia Xuan da Ve

- *LargeIcons*: Một biểu tượng lớn biểu diễn cho mỗi *Item* cùng với một nhãn ngay dưới icon. Các biểu tượng lớn này được lấy từ điều khiển *ImageList*, và được thiết lập trong thuộc tính *LargeImageList* của *ListView*.

*ListView* hiển thị Item dạng *LargeIcons* như sau:



- *List*: Mỗi *Item* sẽ được hiển thị như một biểu tượng nhỏ với một nhãn ở bên phải. Các biểu tượng trong *ListView* được sắp xếp theo các cột.

*ListView* hiển thị Item dạng *List* như sau:

Co-Gai-Xuan-Thi.mp3
Diep-Khuc-Mua-Xuan.mp3
Giot-Mua-Xuan.mp3
Khuc-Xuan.mp3
Kia-Xuan-Da-Ve.mp3
Lang-Nghe-Mua-Xuan-Ve.mp3

- *SmallIcons*: Mỗi *Item* nằm trong một cột gồm có biểu tượng nhỏ cùng với nhãn. Các biểu tượng lớn này được lấy từ điều khiển *ImageList*, và được thiết lập trong thuộc tính *SmallImageList* của *ListView*.

*ListView* hiển thị Item dạng *SmallIcons* như sau:

	 <ul style="list-style-type: none"> <li>- <b>Tiles:</b> Mỗi một Item sẽ hiển thị với biểu tượng có kích thước là tối đa cùng với một label và các subitem sẽ hiển thị các cột bên phải.</li> </ul> <p><i>ListView</i> hiển thị Item dạng <i>Tiles</i> như sau:</p> 
<i>Items</i>	Trả về các Item chứa trong <i>ListView</i> . Một số phương thức và thuộc tính thường dùng của <i>ListView.Items</i> :
<i>MultiSelect</i>	True/ False: Cho phép hoặc không cho phép chọn một lúc nhiều Item trong <i>ListView</i>
<i>FullRowSelect</i>	Khi chọn dòng dữ liệu highlighted cả dòng hay chỉ ô được chọn
<i>GridLines</i>	Nếu thiết lập True sẽ hiển thị các dòng và cột dạng lưới, thiết lập False không hiển thị dạng lưới
<i>SelectedItems</i>	Trả về tập các Items được chọn trong <i>ListView</i>
<i>LargeImageIcon</i>	Gán đối tượng <i>ImageList</i> cho <i>ListView</i>
<i>SmallImageIcon</i>	Gán đối tượng <i>ImageList</i> cho <i>ListView</i>

<i>FocusedItem.Index</i>	Trả về chỉ số dòng được chọn trong <i>ListView</i>
<i>SelectedIndices.Count</i>	Trả về số lượng Item được chọn trong <i>ListView</i>
<i>SelectedIndices</i>	Trả về danh sách chỉ mục các Item được chọn. Ví dụ 4.6: myListView.SeletedIndices[0]: trả về chỉ mục của Item đầu tiên được chọn trong danh sách các Item được chọn trong <i>ListView</i>

- Một số phương thức thường dùng của *ListView*:

Bảng 4.14: Bảng mô tả các phương thức của *ListView*

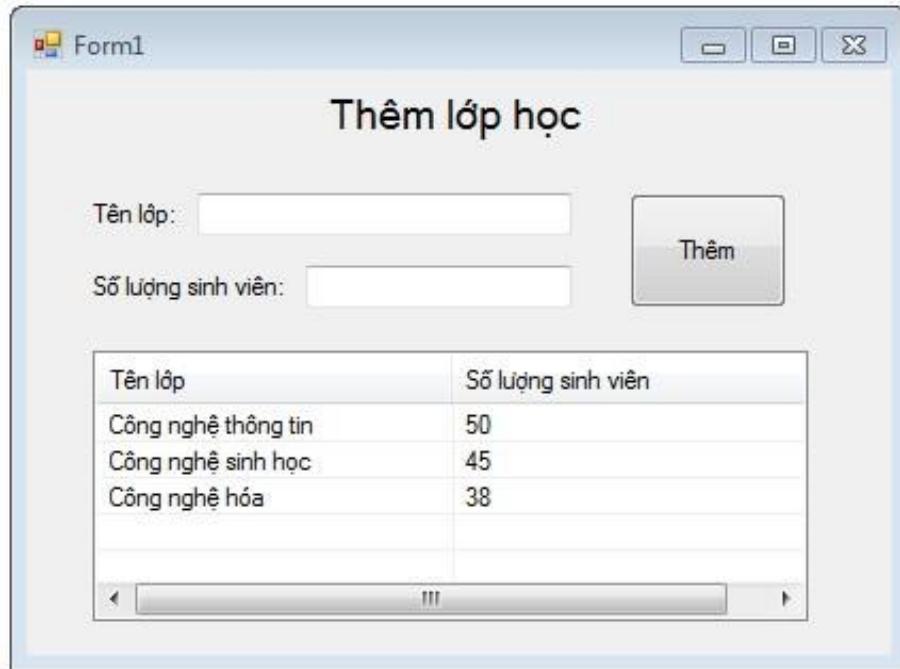
Phương thức	Mô tả
<i>Clear()</i>	Xóa tất cả các <i>Item</i> và <i>Column</i> trong <i>ListView</i>
<i>Sort()</i>	Sắp xếp các <i>Item</i> trong <i>ListView</i>
<i>GetItemAt(x,y)</i>	Lấy <i>Item</i> tại vị trí tọa độ x và y (x và y có thể lấy được thông qua sự kiện <i>Click</i> chuột)

- Một số sự kiện thường dùng của *ListView*:

Bảng 4.15: Bảng mô tả các sự kiện của *ListView*

Sự kiện	Mô tả
<i>SelectedIndexChanged</i>	Sự kiện phát sinh khi có sự thay đổi về chỉ mục được chọn của <i>Item</i> trên <i>ListView</i>
<i>ItemSelectionChanged</i>	Sự kiện phát sinh khi có sự thay đổi lựa chọn một <i>Item</i> trên <i>ListView</i>
<i>ItemCheck</i>	Xảy ra khi trạng thái chọn của <i>Item</i> thay đổi
<i>ColumnClick</i>	Sự kiện phát sinh khi một <i>column</i> trong <i>ListView</i> được click
<i>MouseClick</i>	Sự kiện phát sinh khi nhấp chuột chọn một <i>Item</i> trong <i>ListVlew</i>

Ví dụ 4.7: Thiết kế giao diện form như hình 4.17:

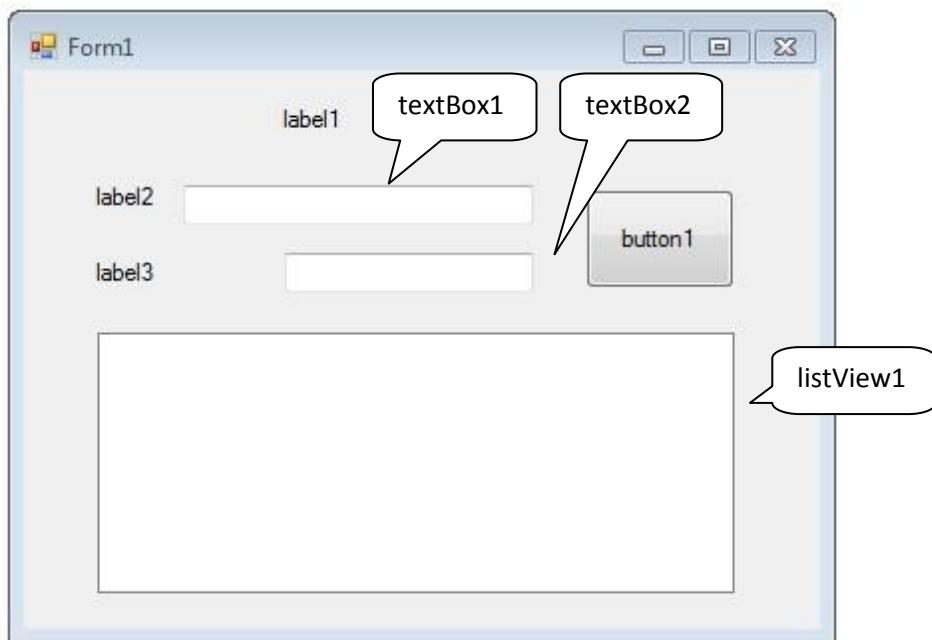


Hình 4.17: Giao diện form thêm lớp

Yêu cầu: Khi người dùng nhập xong tên lớp và số lượng sinh viên, sau đó nhấn nút “Thêm” thì trong ListView sẽ chèn một dòng vào cuối với tên lớp và số lượng vừa nhập.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 4.18



Hình 4.18: Giao diện ban đầu form thêm lớp

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển

- label1:

- Thuộc tính Text: “Thêm lớp học”
- Thuộc tính Size: 14
- label2:  
Thuộc tính Text: “Tên lớp:”
  - label3:  
Thuộc tính Text: “Số lượng sinh viên:”
  - button1:  
Thuộc tính Text: “Thêm”  
Thuộc tính Name: btnThem
  - listView1:  
Thuộc tính Name: myListview  
Thuộc tính View: Details  
Thuộc tính GridLines: True
  - textBox1:  
Thuộc tính Name: txtTenLop
  - textBox2:  
Thuộc tính Name: txtSoLuong

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnThem:

```
private void btnThem_Click(object sender, EventArgs e)
{
    if (txtSoLuong.Text != "" && txtTenLop.Text != "")
    {
        ListViewItem LVIItem = new
            ListViewItem(txtTenLop.Text);
        ListViewItem.ListViewSubItem LVSItem = new
            ListViewItem.ListViewSubItem(LVIItem, txtSoLuong.Text);
        LVIItem.SubItems.Add(LVSItem);
        myListview.Items.Add(LVIItem);
    }
}
```

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    myListview.Columns.Add("Tên lớp", 160);
    myListview.Columns.Add("Số lượng sinh viên", 180);
}
```

#### 4.4. Điều khiển TreeView

TreeView là điều khiển dùng để hiển thị danh sách các đối tượng dưới dạng phân cấp như hình 4.19. Đối tượng trong TreeView thường được gọi là node và cấu trúc phân cấp của TreeView được biểu diễn bởi lớp TreeNode. Mỗi một node trong TreeView có thể chứa các node khác. Node chứa một node khác gọi là node cha (RootNode) và node được chứa gọi là node con (ChildNode). Việc sử dụng điều khiển TreeView để hiển thị rất hữu ích, vì trình bày theo dạng phân cấp giúp việc hiển thị được rõ ràng và có hệ thống hơn. Điều khiển TreeView đặt trong nhóm Common Controls của cửa sổ Toolbox như hình 4.20.



Hình 4.19: Biểu diễn thư mục dạng phân cấp



Hình 4.20: Điều khiển TreeView

- Một số thuộc tính thường dùng của *TreeView*:

*Bảng 4.16: Bảng mô tả các thuộc tính của TreeView*

Thuộc tính	Mô tả
<i>Node</i>	Trả về một đối tượng thuộc lớp <i>TreeNode</i>
<i>SelectedNode</i>	Trả về node đang được chọn trong <i>TreeView</i>
<i>ShowPlusMinus</i>	Hiển thị dấu + và - trước mỗi <i>TreeNode</i>
<i>ShowRootLines</i>	Hiển thị đường thẳng nối giữa các <i>Root Node</i> trong một <i>TreeView</i>
<i>ImageList</i>	Hiển thị hình trước mỗi node trong <i>TreeView</i> . Lưu ý: Phải sử dụng thêm điều khiển <i>ImageList</i> , và gán tên đối tượng của điều khiển <i>ImageList</i> cho thuộc tính <i>ImageList</i> của <i>TreeView</i>
<i>ImageIndex</i>	Giá trị của thuộc tính <i>ImageIndex</i> là chỉ số của hình trong điều khiển <i>ImageList</i> . Khi gán chỉ số cho thuộc tính <i>ImageIndex</i> thì hình hiển thị trước mỗi node sẽ là hình có chỉ số tương ứng. Lưu ý: Phải sử dụng thuộc tính <i>ImageList</i> trước
<i>SelectedImageIndex</i>	Giá trị của thuộc tính <i>SelectedImageIndex</i> là chỉ số của hình trong điều khiển <i>ImageList</i> . Khi người dùng chọn node nào thì node đó sẽ có hình tương ứng như thuộc tính <i>SelectedImageIndex</i> chỉ định

- Một số phương thức thường dùng của *TreeView*:

*Bảng 4.17: Bảng mô tả các phương thức của TreeView*

Phương thức	Mô tả
<i>GetNodeCount()</i>	Đếm số node trong một <i>TreeView</i>
<i>ExpandAll()</i>	Hiển thị tất cả các node trên <i>TreeView</i>
<i>CollapseAll()</i>	Thu gọn tất cả các node trên <i>TreeView</i>
<i>GetNodeAt(x,y)</i>	Lấy một node tại một vị trí có tọa độ (x, y) trên màn hình. Lưu ý: Thường sử dụng sự kiện <i>MouseDown</i> hoặc <i>NodeMouseClick</i>

- Một số sự kiện thường dùng của *TreeView*:

*Bảng 4.18: Bảng mô tả các sự kiện của TreeView*

Sự kiện	Mô tả
<i>AfterCollapse</i>	Phát sinh khi thu gọn một <i>TreeNode</i>
<i>AfterExpand</i>	Phát sinh khi hiển thị các node trong <i>TreeNode</i>

<i>AfterSelect</i>	Phát sinh khi chọn một TreeNode
<i>NodeMouseClick</i>	Phát sinh khi chọn một node

TreeView là điều khiển để hiển thị các node, tuy nhiên việc hiển thị này thực chất là do TreeNode tạo ra. Do đó để làm việc với các node cần sử dụng các thuộc tính và phương thức của lớp TreeNode.

- Một số thuộc tính thường dùng của *TreeNode*:

*Bảng 4.19: Bảng mô tả các thuộc tính của TreeNode*

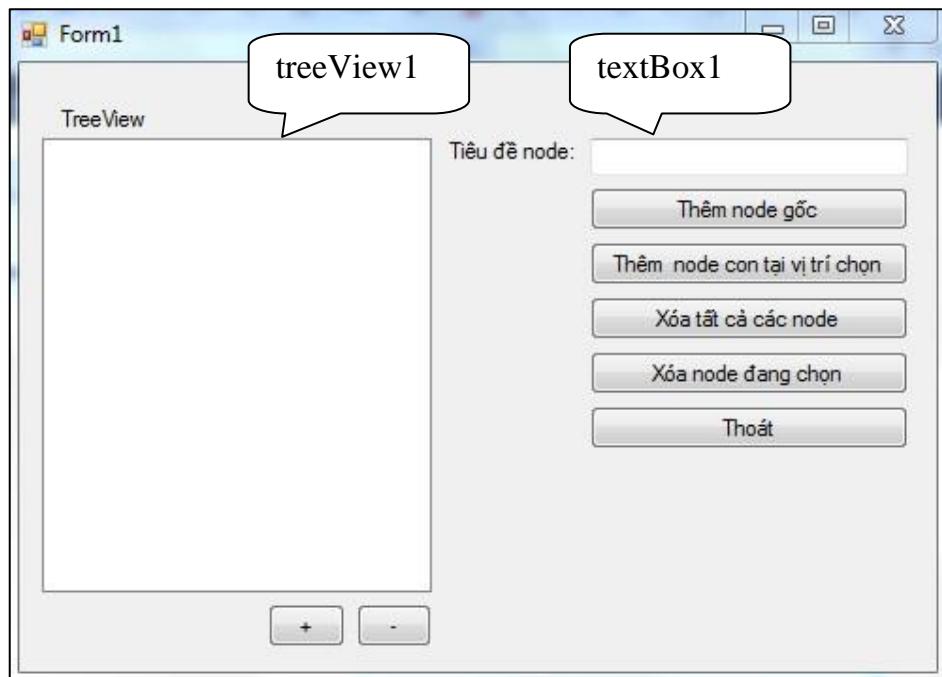
Thuộc tính	Mô tả
<i>Nodes</i>	Trả về tập các node
<i>Text</i>	Đọc/ gán chuỗi ký tự người dùng sẽ nhìn thấy ở mỗi node
<i>FirstNode</i>	Trả về node đầu tiên
<i>LastNode</i>	Trả về node cuối cùng
<i>NextNode</i>	Chuyển đến node tiếp theo
<i>PrevNode</i>	Lùi lại node trước đó
<i>Parent</i>	Trả về node cha của node hiện tại
<i>Index</i>	Trả về chỉ số của node

- Một số phương thức thường dùng của *TreeNode*:

*Bảng 4.20: Bảng mô tả các phương thức của TreeNode*

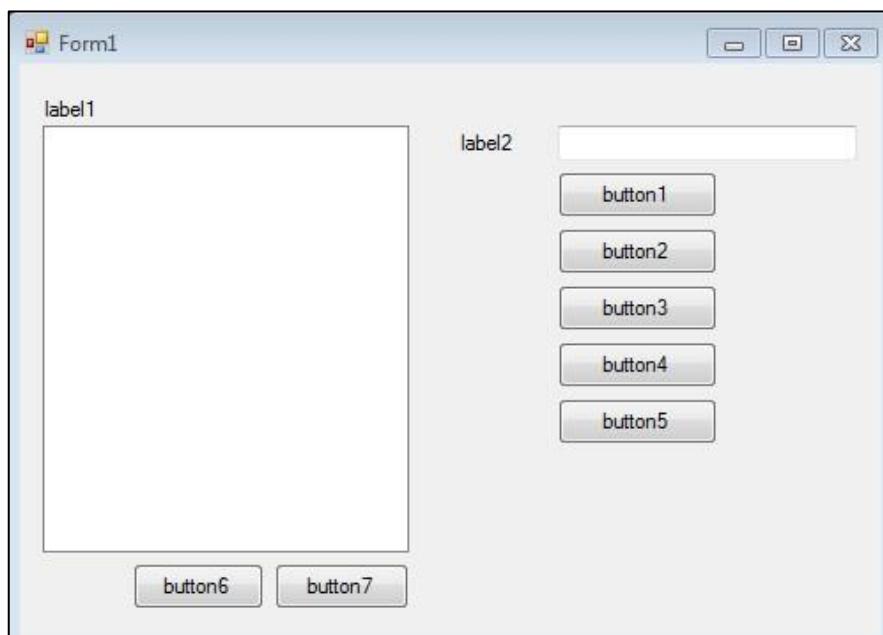
Phương thức	Mô tả
<i>Nodes.Add</i>	Thêm một node
<i>Nodes.Remove</i>	Xóa một node
<i>Nodes.Insert</i>	Chèn vào một node (chèn trước, chèn sau một node)
<i>Nodes.Clear</i>	Xóa tất cả các node con và node hiện tại

Ví dụ 4.8: Viết chương trình minh họa việc thêm sửa xóa các node trong một TreeView. Giao diện thiết kế như hình 4.21.



Hình 4.21: Giao diện form minh họa sử dụng TreeView ví dụ 4.8

Bước 1: Thiết kế giao diện ban đầu như hình 4.22



Hình 4.22: Giao diện ban đầu ví dụ 4.8

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties

- label1:  
Thuộc tính Text: "TreeView"
- label2:  
Thuộc tính Text: "Tiêu đề node:"
- treeView1:  
Thuộc tính Name: TV\_Test
- textBox1:

Thuộc tính Name: txtTieuDe

- button1:

Thuộc tính Text: “Thêm node gốc”

Thuộc tính Name: btnThemGoc

- button2:

Thuộc tính Text: “Thêm node con tại vị trí”

Thuộc tính Name: btnThemCon

- button3:

Thuộc tính Text: “Xóa tất cả các node”

Thuộc tính Name: btnXoaTatCa

- button4:

Thuộc tính Text: “Xóa node đang chọn”

Thuộc tính Name: btnXoaChon

- button5:

Thuộc tính Text: “Thoát”

Thuộc tính Name: btnThoat

- button6:

Thuộc tính Text: “+”

Thuộc tính Name: btnMoRong

- button7:

Thuộc tính Text: “-”

Thuộc tính Name: btnThuNho

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnThemGoc:

```
private void btnThemGoc_Click(object sender, EventArgs e)
{
    TV_Test.Nodes.Add(txtTieuDe.Text);
    txtTieuDe.Text = "";
}
```

- Sự kiện *Click* của nút btnThemCon:

```
private void btnThemCon_Click(object sender, EventArgs e)
{
    TV_Test.SelectedNode.Nodes.Add(txtTieuDe.Text);
    txtTieuDe.Text = "";
    TV_Test.ExpandAll();
}
```

- Sự kiện *Click* của nút btnXoaTaCa:

```
private void btnXoaTaCa_Click(object sender, EventArgs e)
{
    TV_Test.Nodes.Clear();
}
```

- Sự kiện *Click* của nút btnXoaChon:

```
private void btnXoaChon_Click(object sender, EventArgs e)
{
    TV_Test.SelectedNode.Remove();
}
```

- Sự kiện *Click* của nút btnMoRong:

```
private void btnMoRong_Click(object sender, EventArgs e)
{
    TV_Test.ExpandAll();
}
```

- Sự kiện *Click* của nút btnThuHep:

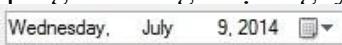
```
private void btnXoaChon_Click(object sender, EventArgs e)
{
    TV_Test.CollapseAll();
}
```

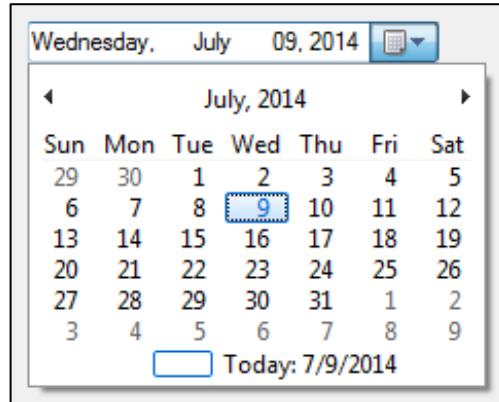
- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

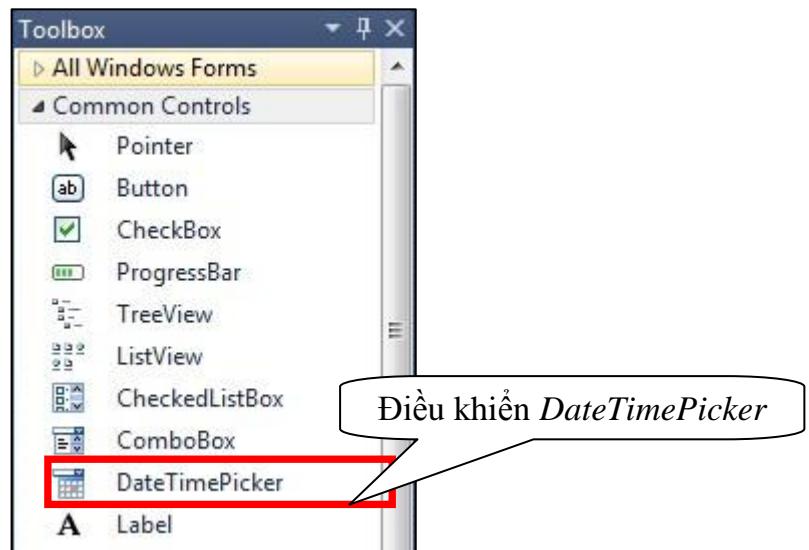
## 4.5. Điều khiển DateTimePicker, MonthlyCalendar

### 4.5.1. Điều khiển DateTimePicker

Điều khiển *DateTimePicker* cho phép người dùng chọn ngày tháng như một lịch biểu nhưng biểu diễn ở dạng ComboBox  khi người dùng nhấp chuột vào ComboBox sẽ sổ xuống lịch biểu như hình 4.23. Các đối tượng ngày tháng biểu diễn trong *DateTimePicker* thực chất là các đối tượng thuộc lớp *DateTime*. Điều khiển *DateTimePicker* được đặt trong nhóm Common Controls của cửa sổ Toolbox như hình 4.24.



Hình 4.23: Điều khiển DateTimePicker dạng lịch biểu

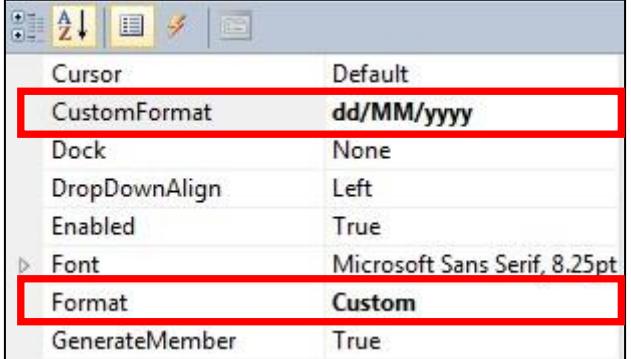


Hình 4.24: Điều khiển DateTimePicker trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của DateTimePicker:

Bảng 4.21: Bảng mô tả các thuộc tính của DateTimePicker

Thuộc tính	Mô tả
Format	Định dạng kiểu hiển thị của ngày tháng. Lưu ý: Thường sử dụng giá trị kiểu Short
Value	Trả về giá trị hiện thời của điều khiển DateTimePicker
Value.Date	Trả về ngày tháng năm
Value.Day	Trả về ngày của tháng
Value.Month	Trả về tháng
Value.Year	Trả về năm
Value.DateOfWeek	Trả về ngày của tuần (0 là chủ nhật, 1 là thứ 2, 2 là thứ 3, ... 6 là thứ 7)
Value.DateOfYear	Trả về ngày thứ bao nhiêu của năm
CustomFormat	Cho phép lập trình viên tạo ra một định dạng

	<p>khác về ngày tháng.</p> <p>Lưu ý: Định dạng ngày tháng năm như kiểu Việt Nam thì kiểu định dạng phải là dd/MM/yyyy. Khi đó thuộc tính format phải thiết lập là Cusom.</p> 
<i>MaxDate</i>	Thiết lập ngày lớn nhất cho phép người dùng chọn trên điều khiển <i>DateTimePicker</i>
<i>MinDate</i>	Thiết lập ngày nhỏ nhất cho phép người dùng chọn trên điều khiển <i>DateTimePicker</i>
<i>Text</i>	Trả về ngày hiển thị

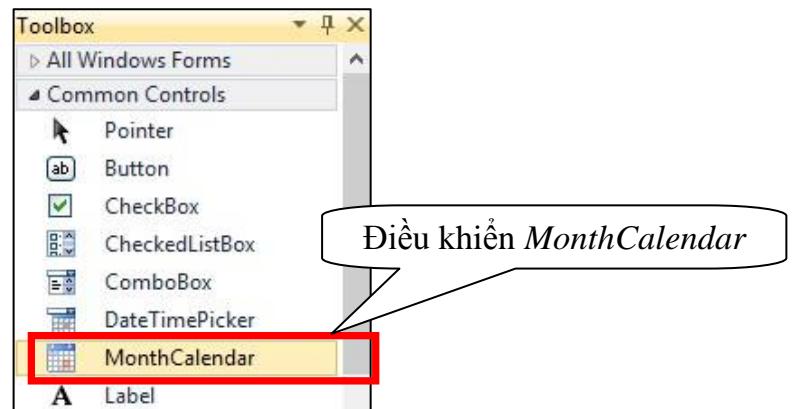
- Một số sự kiện thường dùng của *DateTimePicker*:

Bảng 4.22: Bảng mô tả các sự kiện của *DateTimePicker*

Sự kiện	Mô tả
<i>ValueChanged</i>	Phát sinh khi người dùng chọn giá trị khác với giá trị trước đó trên điều khiển <i>DateTimePicker</i>
<i>CloseUp</i>	Phát sinh người dùng kết thúc việc chọn ngày trên điều khiển <i>DateTimePicker</i>

#### 4.5.2. Điều khiển MonthCalendar

*MonthCalendar* là điều khiển hiển thị lịch dưới dạng một lịch biểu cho phép người dùng chọn ngày tháng. Nhưng khác biệt là *MonthCalendar* cho phép người dùng có thể chọn một tập các ngày hay nói cách khác là một tập các đối tượng thuộc lớp *DateTime*. Điều khiển *MonthCalendar* được đặt trong nhóm Common Controls của cửa sổ Toolbox như hình 4.24.



Hình 4.25: Điều khiển MonthCalendar trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của MonthCalendar:

Bảng 4.23: Bảng mô tả các thuộc tính của MonthCalendar

Thuộc tính	Mô tả
<i>MaxDate</i>	Thiết lập ngày lớn nhất cho phép người dùng chọn trên điều khiển MonthCalendar
<i>MinDate</i>	Thiết lập ngày nhỏ nhất cho phép người dùng chọn trên điều khiển MonthCalendar
<i>SelectionRange</i>	Trả về một dãy các ngày liên tục được chọn bởi người dùng
<i>SelectionStart</i>	Trả về ngày đầu tiên trong dãy tại thuộc tính <i>SelectionRange</i>
<i>SelectionEnd</i>	Trả về ngày cuối cùng trong dãy tại thuộc tính <i>SelectionRange</i>
<i>AnnuallyBoldedDates</i>	Chứa một mảng các ngày. Trong mỗi năm, các ngày trong mảng sẽ được bôi đen MonthCalendar
<i>BoldedDates</i>	Chứa mảng các ngày. Các ngày này sẽ được bôi đen trên điều khiển MonthCalendar tại những năm chỉ định.
<i>MaxSelectCount</i>	Thiết lập số lượng ngày tối đa mà người dùng có thể chọn
<i>MonthlyBoldedDates</i>	Chứa mảng các ngày. Trong mỗi tháng, các ngày trong mảng sẽ được bôi đen trên MonthCalendar

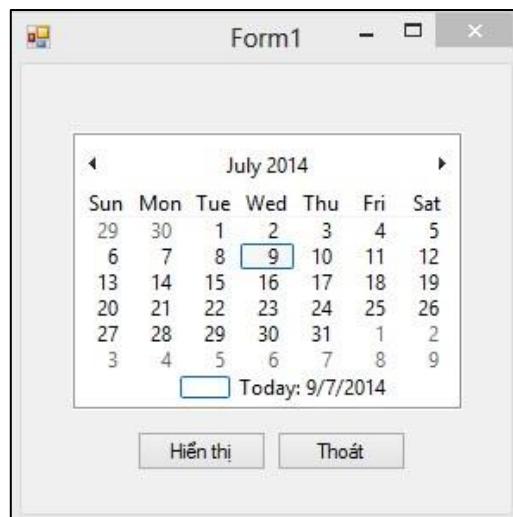
Người dùng có thể chọn một ngày nào đó trên *MonthCalendar* bằng cách nhấp chuột chọn ngày đó hoặc chọn một dãy nhiều ngày liên tiếp bằng cách nhấp chuột chọn ngày đầu tiên và giữ phím shift đồng thời chọn ngày cuối cùng của dãy. Số lượng các ngày được chọn trong dãy phải nhỏ hơn giá trị thiết lập trong thuộc tính *MaxSelectCount*

- Một số sự kiện thường dùng của *MonthCalendar*:

*Bảng 4.24: Bảng mô tả các sự kiện của MonthCalendar*

Sự kiện	Mô tả
<i>DateChanged</i>	Được phát sinh một ngày mới hoặc một dãy các ngày mới được chọn

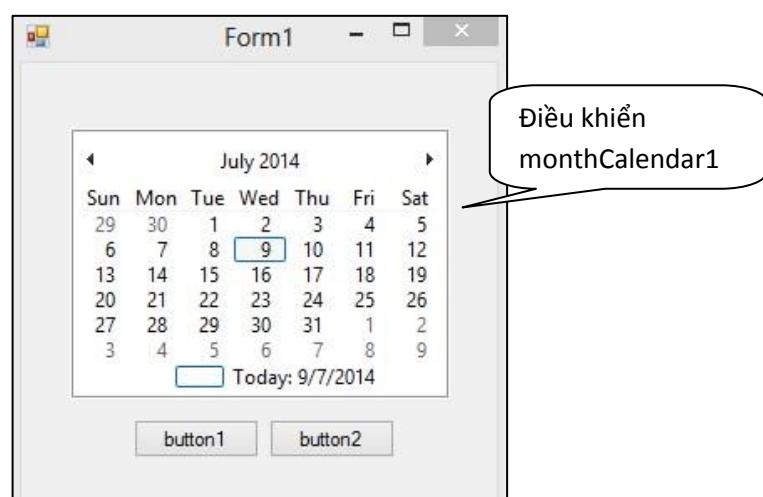
Ví dụ 4.9: Viết chương trình minh họa việc hiển thị lịch, thiết kế giao diện form như hình 4.26. Yêu cầu: khi nhấp nút hiển thị thì các ngày được chọn sẽ hiển thị trên MessageBox



*Hình 4.26: Giao diện chương trình hiển thị lịch ví dụ 4.9*

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu như hình 4.27



#### Hình 4.27: Giao diện ban đầu ví dụ 4.9

Bước 2: Thiết lập giá trị cho thuộc tính trong cửa sổ Properties

- button1:

Thuộc tính Text: “Hiển thị”

Thuộc tính Name: btnHienThi

- button2:

Thuộc tính Text: “Thoát”

Thuộc tính Name: btnThoat

- monthCalendar1:

Thuộc tính Name: MyMCalendar

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnHienThi:

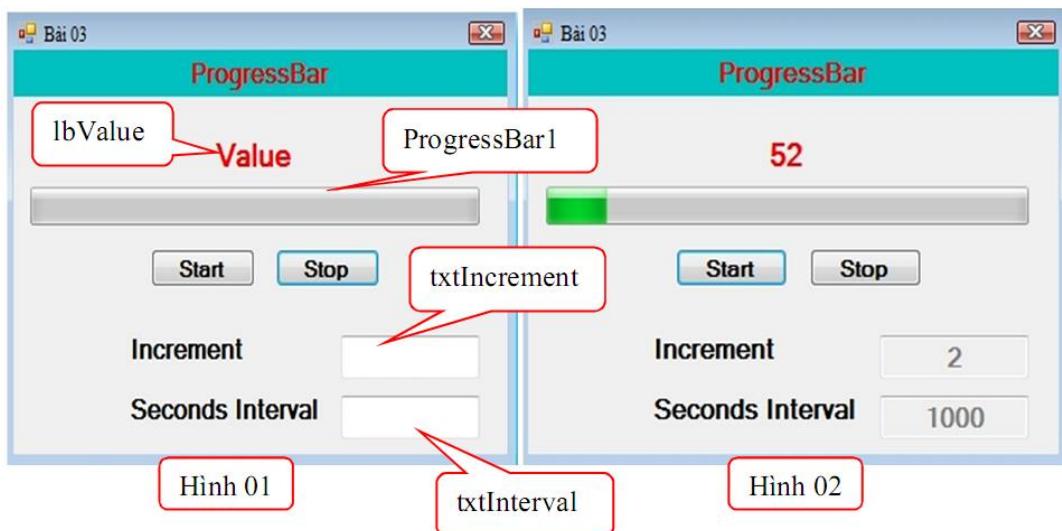
```
private void btnHienThi_Click(object sender, EventArgs e)
{
    string strngay="";
    DateTime i=new DateTime();
    for ( i = MyMCalendar.SelectionStart;
          i<= MyMCalendar.SelectionEnd; i=i.AddDays(1.0))
    {
        strngay += i.ToString() + "\n";
    }
    MessageBox.Show(strngay);
}
```

- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

### 4.6. Bài tập cuối chương

Câu 1: Thiết kế chương trình có giao diện như hình 4.28

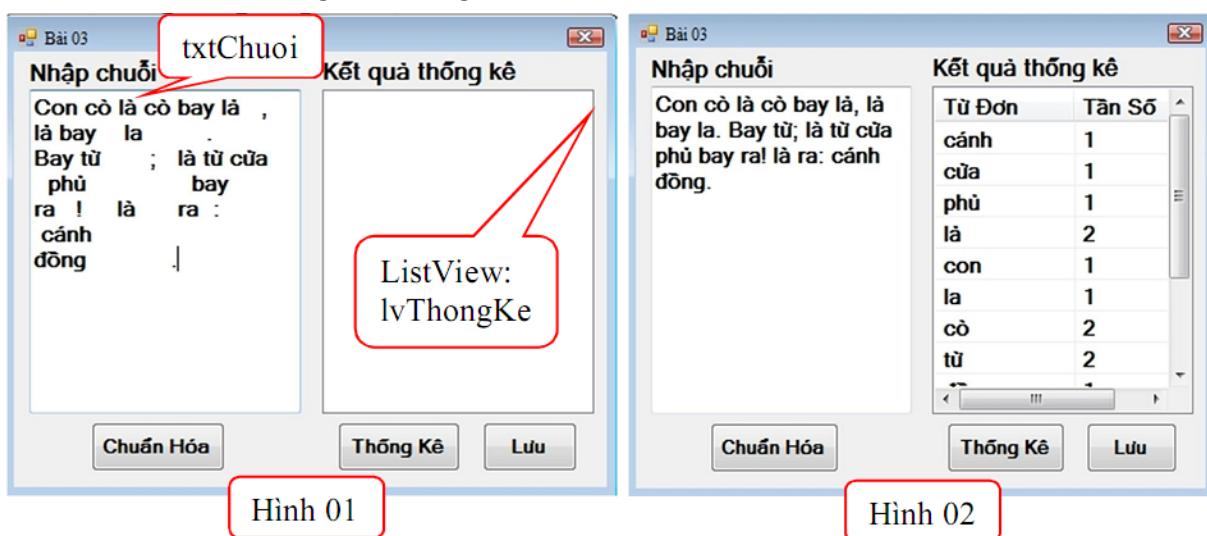


Hình 4.28: Giao diện chương trình ProgressBar

Yêu cầu:

- Người dùng nhập giá trị trong TextBox txtIncrement; nhập giá trị trong TextBox txtInterval.
- Khi người dùng nhấn nút Start ProgressBar1 sẽ tăng giá trị thuộc tính Value lên (trong khoảng từ Minimum đến Maximum) đồng thời thuộc tính Value được gán cho Label lbValue, mỗi lần tăng với giá trị bằng giá trị đã nhập trong txtIncrement, cứ sau mỗi thời gian được nhập trong txtInterval (/1000 giây) sẽ tăng giá trị thuộc tính Value lên. Khi thuộc tính Value có giá trị  $\geq$  Maximum thì thuộc tính Value sẽ được gán bằng Minimum trở lại. Ngoài ra khi nhấn nút Start nếu các TextBox đã có giá trị hợp lệ sẽ chuyển sang trạng thái chỉ đọc không cho người dùng chỉnh sửa (Hình 02).
- Khi người dùng nhấn nút Stop sẽ trả về trạng thái ban đầu.

Câu 2: Thiết kế chương trình có giao diện như hình 4.29.

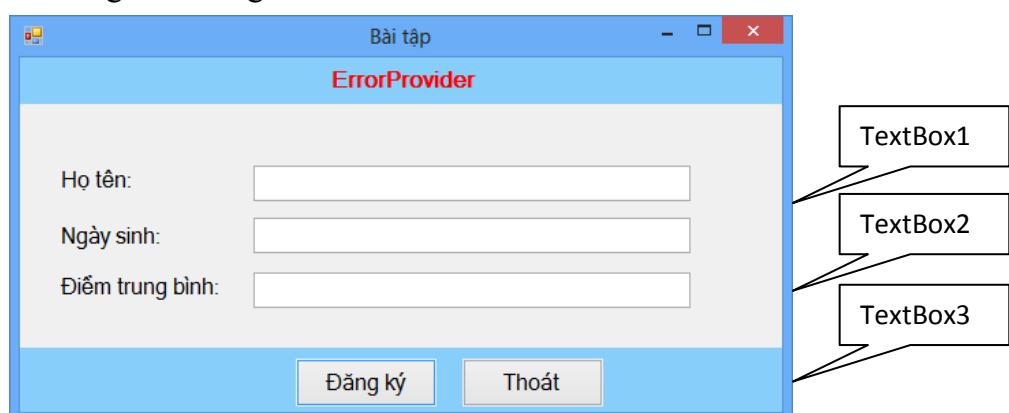


Hình 4.29: Giao diện chương trình chuẩn hóa chuỗi

Yêu cầu:

- Khi bắt đầu chạy chương trình hiển thị một MessageBox với nội dung là “Bạn có muốn tiếp tục Load Form?”, chuỗi hiển thị trên titlebar MessageBox là Họ tên - số máy của sinh viên dự thi.
  - o Nếu người dùng nhấn chọn No thì dừng việc Load Form, ngược lại,
  - o Nếu người dùng nhấn chọn Yes thì hiện Form với giao diện như Hình 01.
- Người dùng nhập một chuỗi bất kỳ vào TextBox “txtChuoi”. Khi người dùng nhấn Button “Chuẩn Hóa” sẽ định dạng lại chuỗi nhập theo yêu cầu sau:
  - o Loại bỏ các khoảng trắng thừa (loại bỏ các khoảng trắng ở đầu và cuối chuỗi, làm sao giữa các từ cách nhau một khoảng trắng.)
  - o Đầu câu nằm sát từ đứng kề trước và cách từ đứng kề sau một khoảng trắng. (Xử lý các dấu câu sau: dấu chấm (“.”), dấu phẩy (“,”), dấu hai chấm (“::”), dấu chấm than (“!”), dấu hỏi (“?”), dấu chấm phẩy (“;”))
- Chuỗi sau khi chuẩn hóa sẽ được hiển thị ở TextBox “txtChuoi”. (Xem Hình 02)
- Khi người dùng nhấn Button “Thông Kê” sẽ thống kê tần số xuất hiện của các từ đơn trong chuỗi ra ListView “lvThongKe” (không phân biệt chữ hoa, chữ thường khi thống kê). (Xem Hình 02)
- Khi người dùng nhấn Button “Lưu” thì sẽ hiện MessageBox thông báo “Đã lưu thành công” và thoát khỏi chương trình.

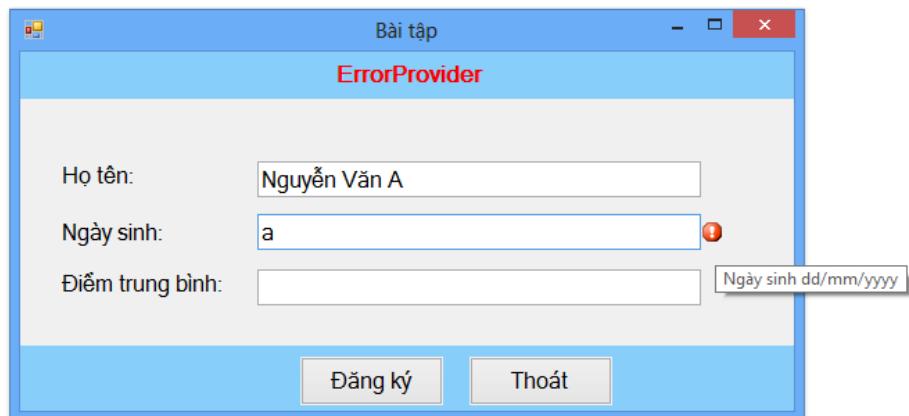
Câu 3: Thiết kế chương trình có giao diện như hình 4.30.



Hình 4.30: Giao diện chương trình nhập thông tin sinh viên

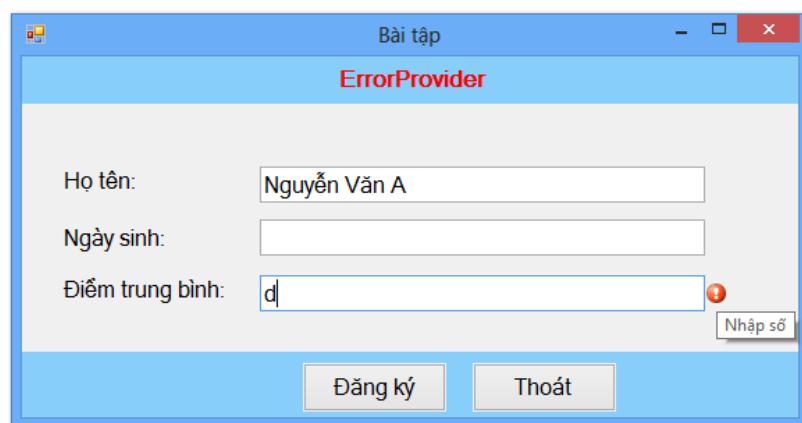
Yêu cầu:

- TextBox2 phải nhập ngày sinh theo định dạng dd/mm/yyyy. Nếu nhập sai thì sẽ hiện thông báo lỗi “Ngày sinh dd/mm/yyyy” như hình 4.31.



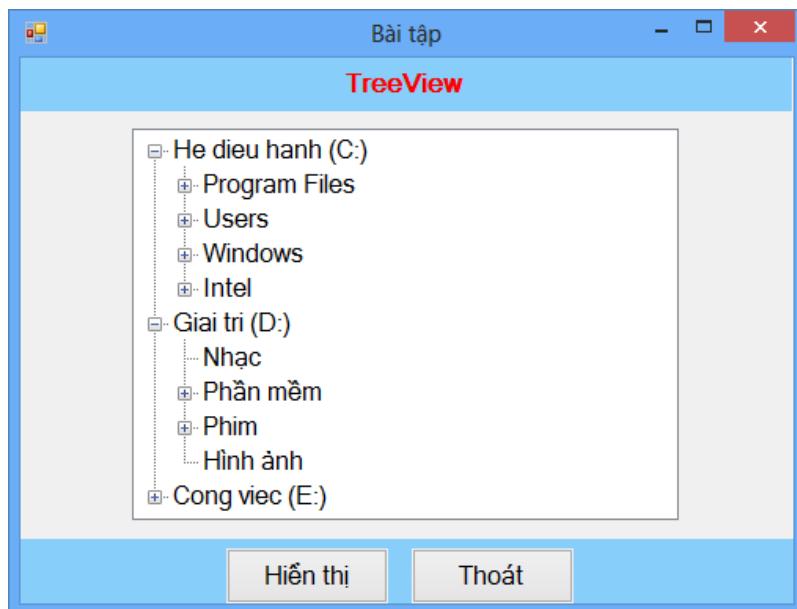
Hình 4.31: Nhập sai định dạng ngày tháng

- TextBox3 phải nhập điểm trung bình là số. Nếu nhập sai thì sẽ hiện thông báo lỗi “Nhập số” như hình 4.32.



Hình 4.42: Nhập sai định dạng số

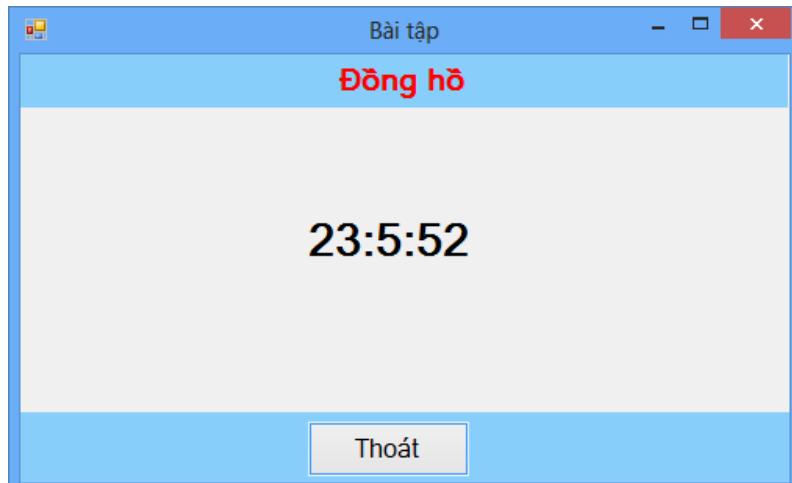
Câu 4: Viết chương trình hiển thị các thư mục trong máy tính. Giao diện chương trình như hình 4.43.



Hình 4.43: Giao diện chương trình hiển thị cây thư mục

- Khi nhấn Button “Hiển thị” thì sẽ hiển thị các thực mục có trong máy tính trên TreeView.
- Khi nhấn Button “Thoát”: đóng chương trình.

Câu 5: Viết chương trình tạo đồng hồ điện tử như hình 4.44 (sử dụng điều khiển Timer)

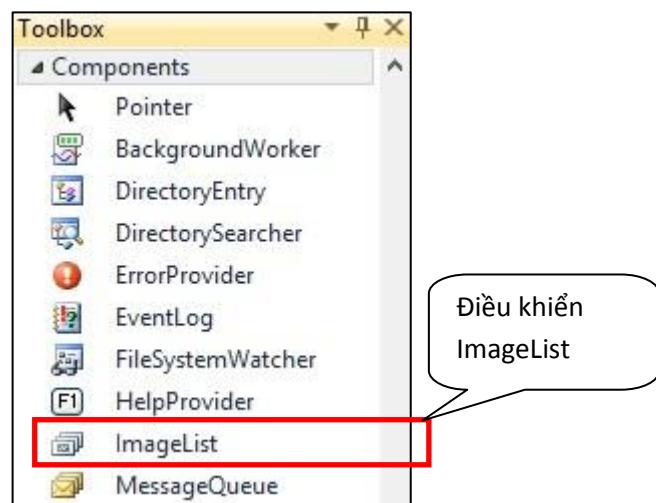


Hình 4.44: Giao diện chương trình đồng hồ điện tử

# CHƯƠNG 5: ĐIỀU KHIỂN DÙNG ĐỂ XÂY DỰNG MENU

## 5.1. Điều khiển ImageList

*ImageList* là một kiểu collection đặc biệt chứa các hình có kích thước và độ sâu màu được xác định trước. Các điều khiển khác nếu có hỗ trợ dùng *ImageList* thì dùng các hình trong *ImageList* thông qua chỉ mục. Một số điều khiển hỗ trợ sử dụng *ImageList* như: *ListView*, *TreeView*, *ToolBar*, *Button*, ... *ImageList* nằm trong nhóm Components của cửa sổ Toolbox như hình 5.1.

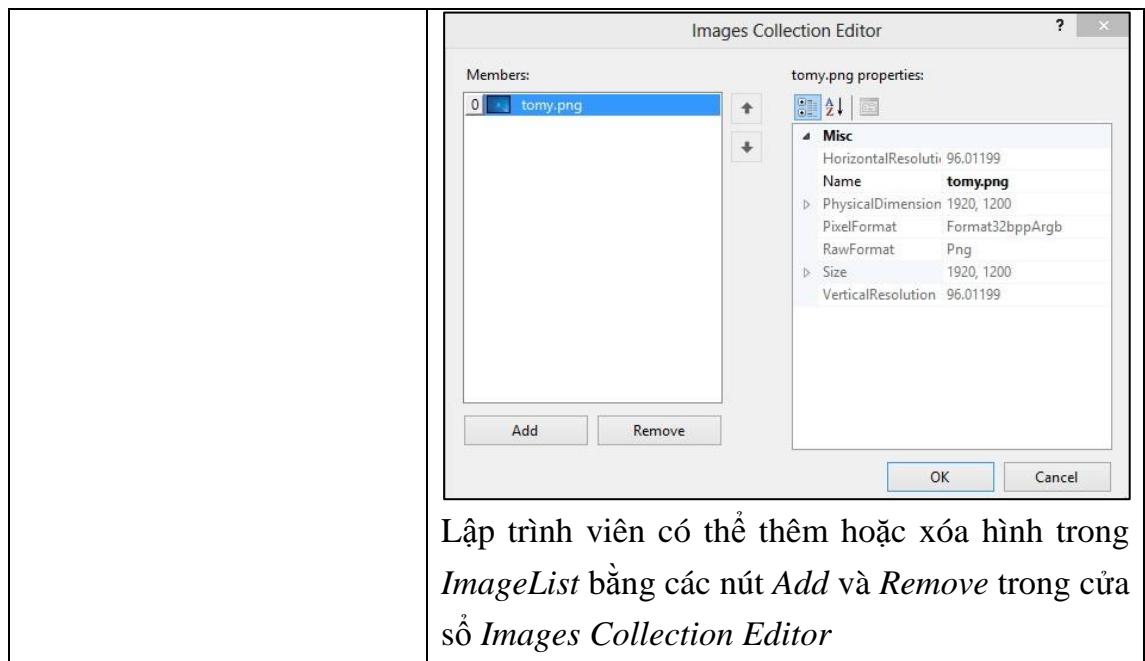


Hình 5.1: Điều khiển *ImageList* trong cửa sổ *Toolbox*

- Một số thuộc tính thường dùng của *ImageList*:

Bảng 5.1: Bảng mô tả các thuộc tính của *ImageList*

Thuộc tính	Mô tả
<i>ImageSize</i>	Kích thước của hình
<i>TransparentColor</i>	Định nghĩa độ trong suốt của màu
<i>ColorDepth</i>	Thiết lập độ sâu của hình được chứa trong <i>ImageList</i>
<i>Images</i>	Tập các hình chứa trong <i>ImageList</i> . Cửa sổ <i>Images Collection Editor</i> khi nhấp chuột chọn thuộc tính <i>Images</i> trong cửa sổ <i>Properties</i> :



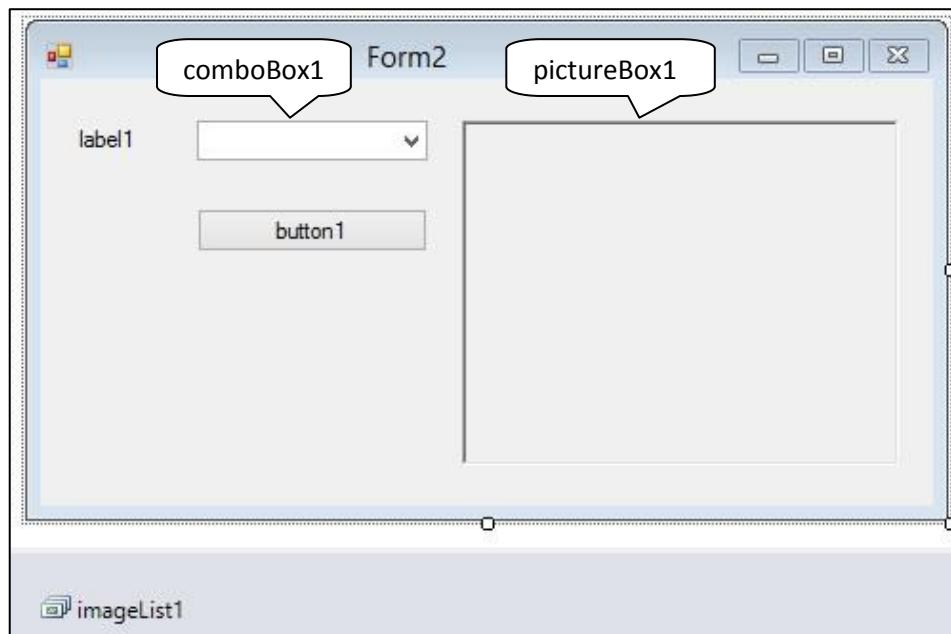
Ví dụ 5.1: Thiết kế giao diện chương trình như hình 5.2. Yêu cầu: Khi chọn hình muốn hiển thị trên ComboBox thì ở điều khiển PictureBox sẽ hiển thị hình tương ứng như đã chọn.



*Hình 5.2: Giao diện form hiển thị hình ví dụ 5.1*

Hướng dẫn:

Bước 1: Thiết kế giao diện chương trình ban đầu như hình 5.3



Hình 5.3: Giao diện ban đầu form hiển thị hình

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties

- label1:

Thuộc tính *Text*: “Chọn hình.”

- comboBox1:

Thuộc tính *Name*: cboChonHinh

- button1:

Thuộc tính *Text*: “Thoát”

Thuộc tính *Name*: btnThoat

- pictureBox1: //PictureBox là điều khiển dùng để hiển thị hình ảnh trên Form

Thuộc tính *Name*: picHinh

Thuộc tính *Size*: 229, 181

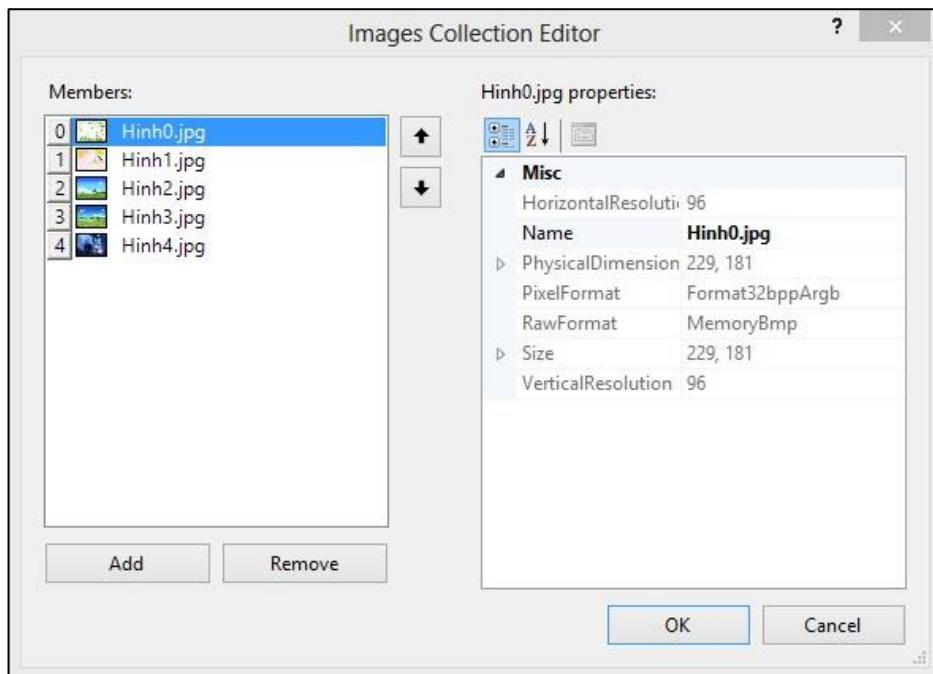
Thuộc tính *BorderStyle*: BorderStyle.FixedSingle

- imageList1:

Thuộc tính *Name*: MyImgList

Thuộc tính *ImageSize*: 229, 181

Thuộc tính *Images*: như hình 5.4



Hình 5.4: Thiết lập thuộc tính Images của ImageList1

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < MyImgList.Images.Count; i++)
    {
        cboChonHinh.Items.Add("Hinh" + i);
    }
    picHinh.BorderStyle=BorderStyle.FixedSingle
}
```

- Sự kiện *SelectedIndexChanged* của nút cboHienThi:

```
private void cboChonHinh_SelectedIndexChanged(object sender, EventArgs e)
{
    //Thuộc tính Image của PictureBox là thuộc tính chỉ định
    //hình sẽ được hiển thị
    picHinh.Image=
        MyImgList.Images[cboChonHinh.SelectedIndex];
}
```

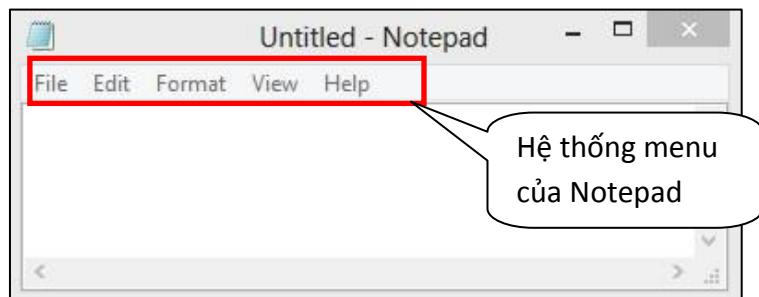
- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

## 5.2. Điều khiển ToolStrip

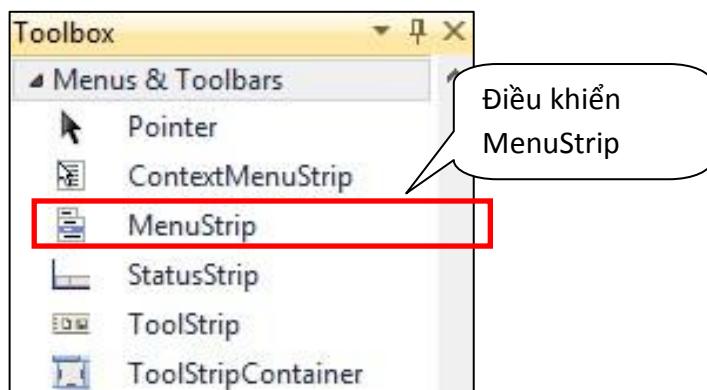
Điều khiển ToolStrip cho phép lập trình viên xây dựng hệ thống menu trên form. Menu có thể xây dựng ở dạng một cấp hoặc nhiều cấp. Ví dụ menu của Notepad như hình 5.5. ToolStrip cho phép xây dựng menu với các điều khiển:

- ToolStripSeparator
- ToolStripMenuItem (Menu con)
- ToolStripCombobox (Combobox)
- ToolStripTextbox (Textbox)



Hình 5.5: Menu của trình soạn thảo Notepad

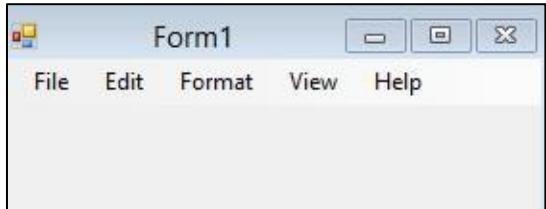
MenuStrip được đặt trong nhóm Menus & Toolbars của cửa sổ Toolbox như hình 5.6.

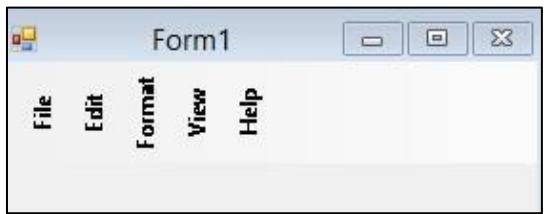
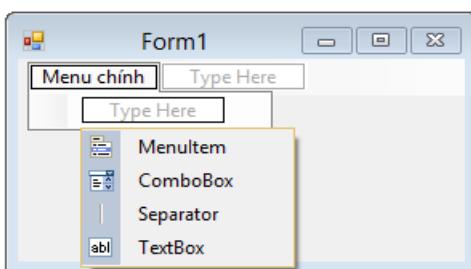
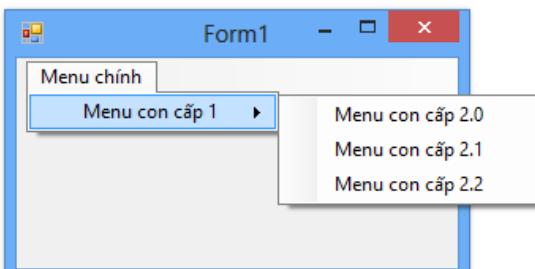
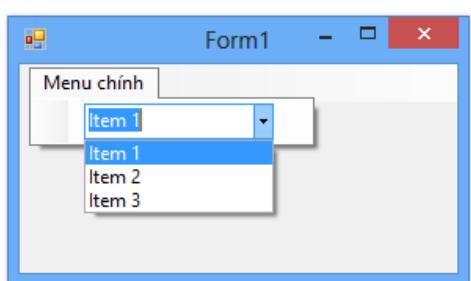


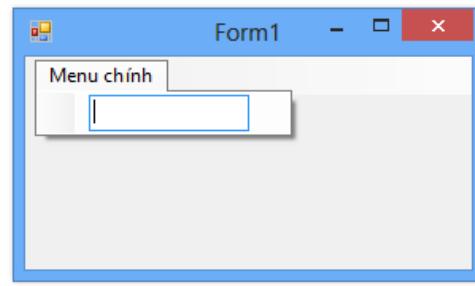
Hình 5.6: Điều khiển ToolStrip trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của ToolStrip:

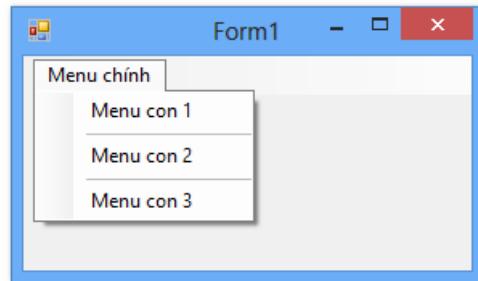
Bảng 5.2: Bảng mô tả các thuộc tính của ToolStrip

Thuộc tính	Mô tả
TextDirection	Chọn hình thức trình bày menu. - Hình thức Horizontal: 

	<ul style="list-style-type: none"> <li>- Hình thức Vertical 90:</li> </ul>  <ul style="list-style-type: none"> <li>- Hình thức Vertical 270:</li> </ul> 
<i>Items</i>	<p>Thêm các menu con. Kiểu menu có thể chọn một trong 4 dạng: MenuItem, ComboBox, Separator, TextBox.</p>  <ul style="list-style-type: none"> <li>- Dạng MenuItem (menu con):</li> </ul>  <ul style="list-style-type: none"> <li>- Dạng Combobox:</li> </ul>  <ul style="list-style-type: none"> <li>- Dạng TextBox:</li> </ul>



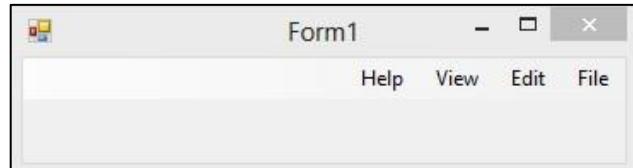
- Separator (Gạch phân cách):



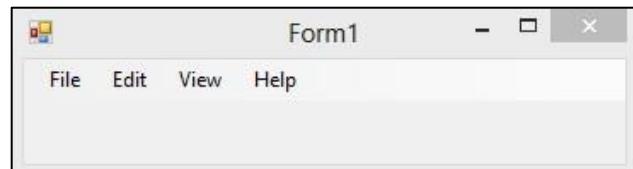
*RightToLeft*

Mang giá trị Yes hoặc No.

Nếu là Yes thì sẽ trình bày menu từ phải qua trái:



Nếu là No thì sẽ trình bày menu từ trái qua phải:



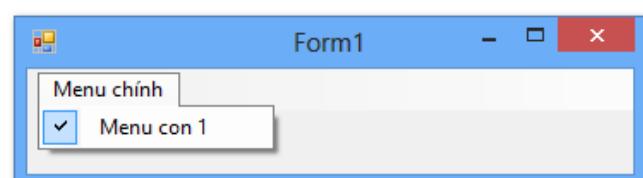
➤ Một số thuộc tính, phương thức và sự kiện của menu con trên *MenuStrip*:

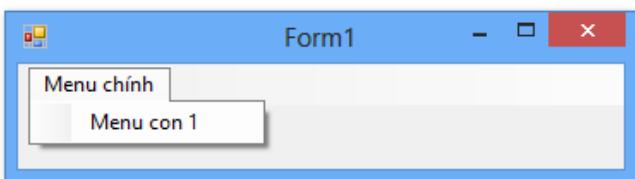
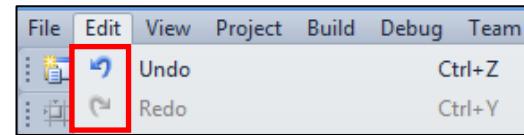
MenuStrip có 4 dạng menu con, mỗi menu con đều có thuộc tính, phương thức và sự kiện riêng tương ứng với mỗi dạng.

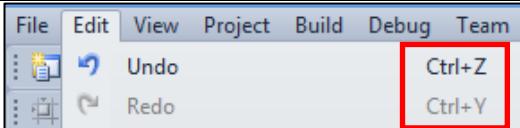
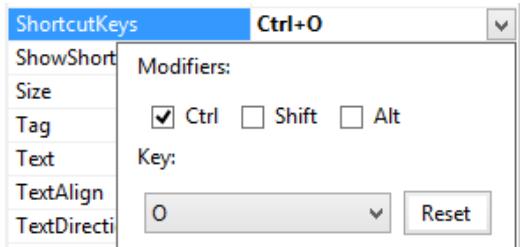
- Menu con dạng MenuItem:

Bảng 5.3: Bảng mô tả các thuộc tính thường dùng của MenuItem

Thuộc tính	Mô tả
<i>Checked</i>	Mang giá trị True hoặc False. - Nếu là True: Hiện biểu tượng CheckBox bên cạnh chuỗi Text



	<ul style="list-style-type: none"> <li>- Nếu là False: Không biểu tượng CheckBox</li> </ul> 
<i>CheckOnClick</i>	<p>Mang giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>- Nếu là True: Biểu tượng CheckBox sẽ xuất hiện bên cạnh chuỗi Text của menu con khi người dùng nhấp chuột chọn.</li> <li>- Nếu là False: Thao tác nhấp chuột của người dùng sẽ không ảnh hưởng gì đến việc hiển thị hay không hiển thị của biểu tượng CheckBox</li> </ul>
<i>CheckState</i>	<p>Cho biết trạng thái của CheckBox trên menu con. Có 3 trạng thái: UnChecked, Checked, Indeterminate.</p> <p>Lưu ý: Trạng thái Indeterminate chỉ có hiệu lực khi thuộc tính Checked là True.</p>
<i>DisplayStyle</i>	<p>Hình thức trình bày của menu con. Có 4 kiểu hiển thị:</p> <ul style="list-style-type: none"> <li>- None: Không hiển thị gì trên menu con</li> <li>- Text: Cho phép hiển thị chuỗi mô tả</li> <li>- Image: Cho phép hiển thị hình hoặc biểu tượng bên cạnh Text.</li> <li>- ImageAndText: Cho phép hiển thị hình (biểu tượng) và chuỗi mô tả.</li> </ul>
<i>Image</i>	<p>Hình ảnh xuất hiện bên cạnh chuỗi Text</p> 
<i>ImageScaling</i>	<p>Kiểu trình bày của hình trong thuộc tính <i>Image</i>. Có thể thiết lập một trong hai giá trị:</p> <ul style="list-style-type: none"> <li>- <i>None</i>: Hiển thị bình thường</li> <li>- <i>SizeToFit</i>: Hiển thị đúng kích cỡ của hình hoặc biểu tượng</li> </ul>
<i>ShortcutKeyDisplayString</i>	<p>Chuỗi trình bày ứng với phím tắt mô tả cho menu đó.</p>

	
<i>ShorcutKeys</i>	Tổ hợp phím tắt ứng với menu
	
<i>ShowShortcutKeys</i>	Mang giá trị <i>True</i> hoặc <i>False</i> : <ul style="list-style-type: none"> <li>- Nếu là <i>True</i>: Cho phép hiển thị giá trị trong thuộc tính <i>ShortcutKeyDisplayString</i>.</li> <li>- Nếu là <i>False</i>: giá trị trong thuộc tính <i>ShortcutKeyDisplayString</i> sẽ không hiển thị.</li> </ul>
<i>Text</i>	Chuỗi ký tự hiển thị trên menu
<i>TooltipText</i>	Chuỗi ký tự hiển thị khi rê chuột vào menu

Bảng 5.4: Bảng mô tả các sự kiện thường dùng của MenuItem

Thuộc tính	Mô tả
<i>CheckedChange</i>	Phát sinh khi trạng thái ( <i>CheckState</i> ) của <i>CheckBox</i> thay đổi
<i>Click</i>	Phát sinh khi người dùng nhấp chuột vào menu

- Menu con dạng *Separator*: Menu dạng này đơn thuần chỉ là một đường kẻ ngang giúp ngăn cách các menu với nhau, giúp hệ thống menu hiển thị một cách rõ ràng hơn.
- Menu con dạng *ComboBox*: Các thuộc tính, phương thức và sự kiện giống với điều khiển *ComboBox* tại mục 3.4 của chương 3.
- Menu con dạng *TextBox*: Các thuộc tính, phương thức và sự kiện giống với điều khiển *TextBox* tại mục 3.3 của chương 3.

Ví dụ 5.2: Thiết kế giao diện chương trình hiển thị thời gian như hình 5.7.



Hình 5.7: Giao diện chương trình hiển thị thời gian ví dụ 5.2

Yêu cầu:

Menu **Chức năng**: Chức mục **Thoát** dạng MenuItem. Khi người dùng nhấn chuột trái vào **Thoát** hoặc nhấn tổ hợp phím **Ctrl + T** sẽ thoát chương trình.

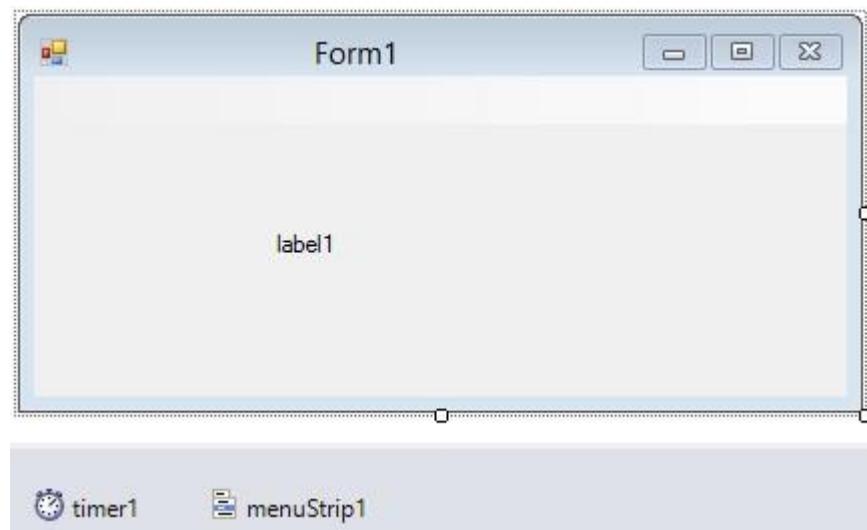
Menu **Nội dung hiển thị**: Dạng ComboBox. ComboBox chứa hai mục chọn:

**Hiển thị thời gian**: Giúp hiển thị giờ phút giây trên label **lblHienThi**

**Hiển thị ngày tháng**: Giúp hiển thị ngày tháng năm trên label **lblHienThi**

Hướng dẫn:

Bước 1: Thiết kế giao diện form ban đầu: Kéo các điều khiển từ cửa sổ Toolbox và form1 như hình 5.8.



Hình 5.8: Giao diện ban đầu form, hiển thị thời gian ví dụ 5.2

Bước 2: Thiết lập các giá trị cho điều khiển trong cửa sổ Properties

- form1:

Thuộc tính *Text*: "MenuStrip"

- label1:

Thuộc tính *Text*: "....."

Thuộc tính *Size*: 14

Thuộc tính *Name*: lblHienThi

- timer:

Thuộc tính *Name*: MyTime

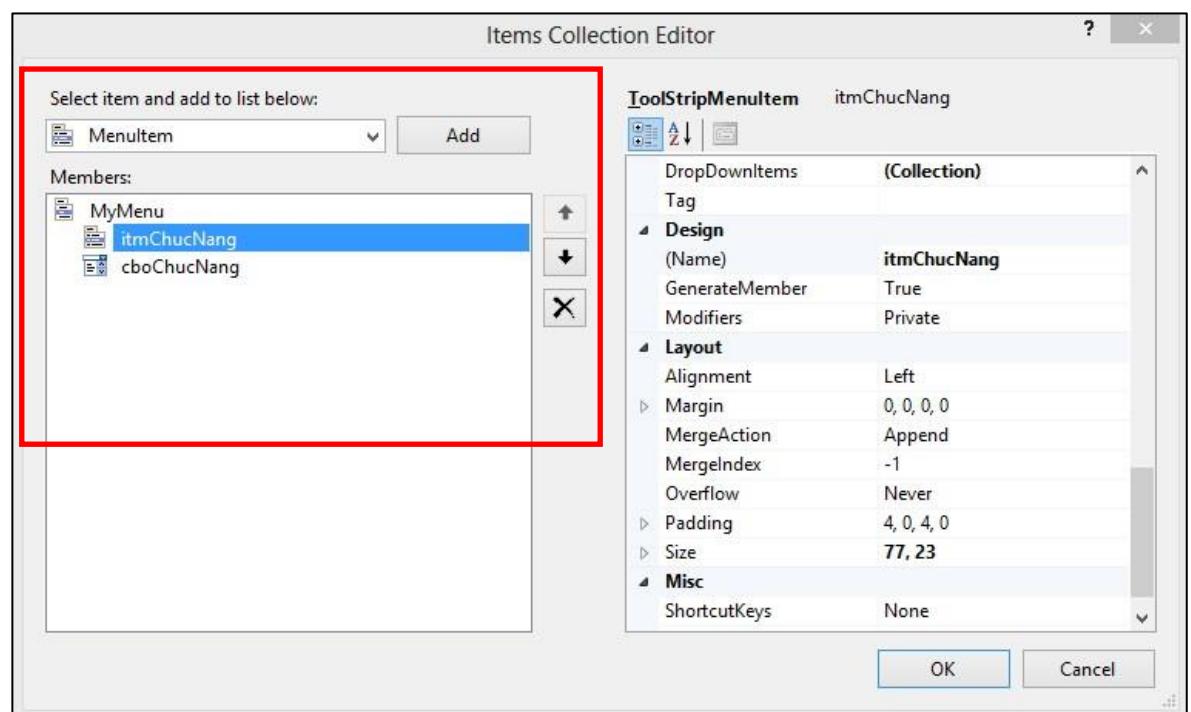
Thuộc tính *Enable*: True

Thuộc tính *Interval*: 1000

- menuStrip1:

Thuộc tính *Name*: MyMenu

Thuộc tính *DropDownItems* của **MyMenu**: Chọn thuộc tính *DropDownItems* trong cửa sổ Properties sẽ hiển thị cửa sổ Items Collection Editor của **MyMenu**. Tại cửa sổ này thêm hai menu có tên: **itmChucNang** (dạng MenuItem) và **cboChucNang** (dạng ComboBox) như hình 5.9



Hình 5.9: Cửa sổ Items Collection Editor

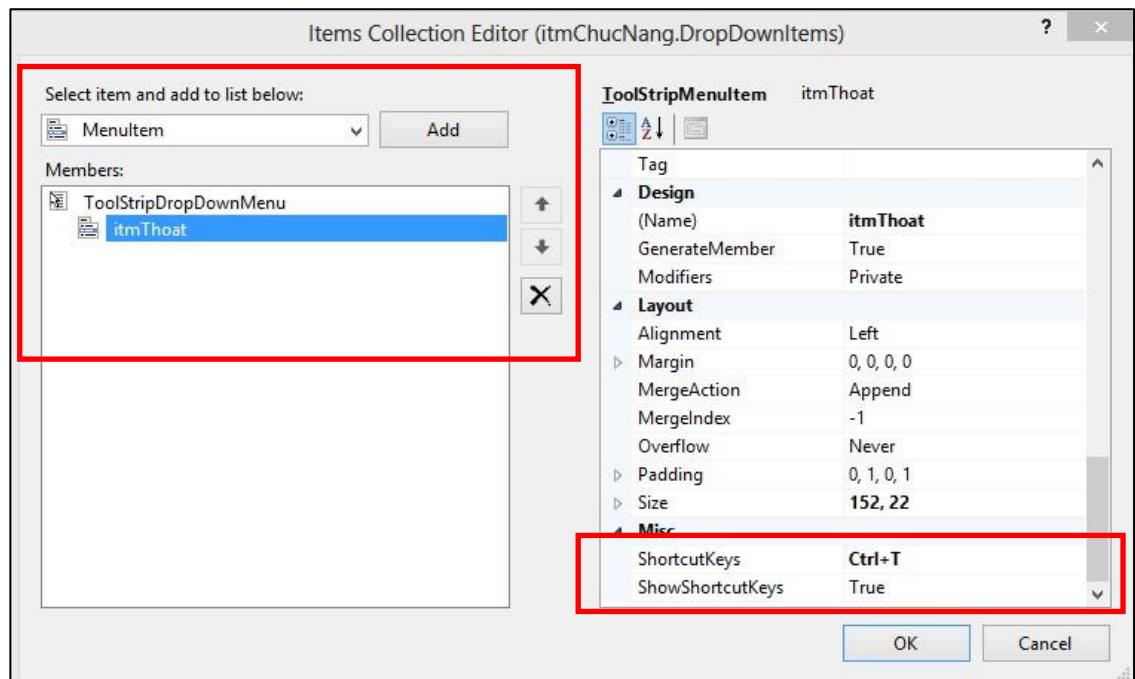
- Nhấn chuột chọn **itmChucNang** để thiết lập thuộc tính: Trên các thuộc tính của **itmChucNang**, chọn thuộc tính *DropDownItems* sẽ hiển thị cửa sổ Items Collection Editor của **itmChucNang**. Trên cửa sổ này thêm một menu dạng MenuItem có tên **itmThoat** như hình 5.10.

Thiết lập thuộc tính *ShortCutKeys* của **itmThoat**: Ctrl + T

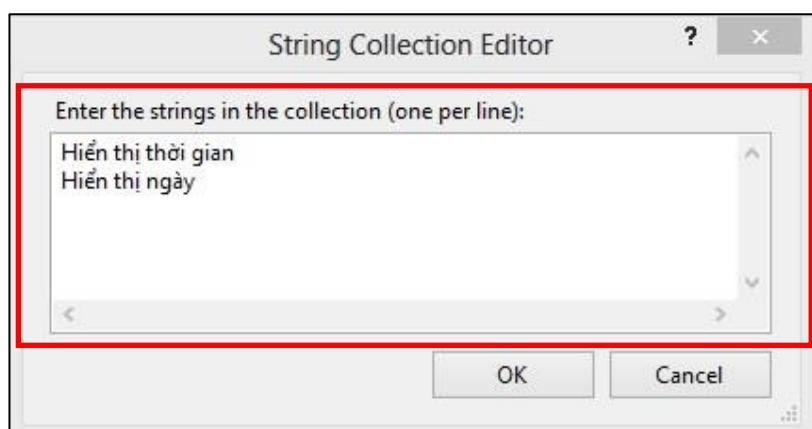
Thiết lập thuộc tính *ShortCutKeyDisplayString* của **itmThoat**: Ctrl + T

Thiết lập thuộc tính *ShowShortCutKeys* của **itmThoat**: True

- Nhấn chuột **cboChucNang** để thiết lập thuộc tính: Trên các thuộc tính của **cboChucNang**, chọn thuộc tính *Items* và thêm hai mục chọn: “Hiển thị thời gian”, “Hiển thị ngày tháng” như hình 5.11.



Hình 5.10: Cửa sổ Items Collection Editor của itmChucNang



Hình 5.11: Cửa sổ String Collection Editor của cboChucNang

### Bước 3: Viết mã lệnh cho các điều khiển

- Khai báo biến:

```
int chon=3;
DateTime dt = new DateTime();
```

- Sự kiện Tick của MyTimer:

```
private void MyTimer_Tick(object sender, EventArgs e)
{
    dt = DateTime.Now;
    if (chon == 0)
    {
        lblHienThi.Text = dt.Hour + ":" + dt.Minute + ":" +
                          dt.Second;
    }
}
```

- Sự kiện *Click* của itmThoat:

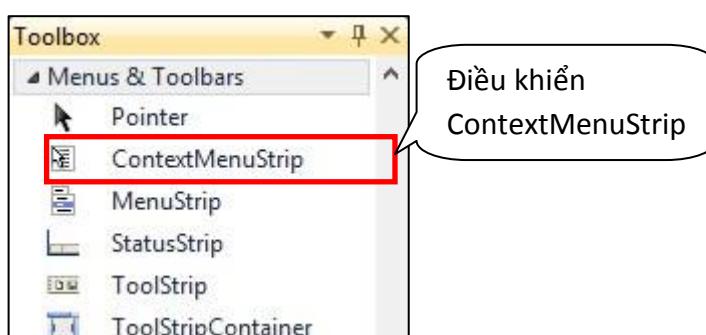
```
private void itmThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *SelectedIndexChanged* của cboChucNang:

```
private void cboChucNang_SelectedIndexChanged(object sender,
EventArgs e)
{
    chon = cboChucNang.SelectedIndex;
    dt = DateTime.Now;
    if (chon == 1)
    {
        lblHienThi.Text = dt.Date.ToString("dd/MM/yyyy");
    }
}
```

### 5.3. Điều khiển ContextMenuStrip

Điều khiển *ContextMenuStrip* dùng để thiết kế menu Popup (menu ngũ cành). Menu Popup là menu dạng như loại menu khi người dùng nhấn chuột phải vào màn hình desktop thì hiện lên một menu. Trong lập trình ứng dụng Windows Form, menu Popup sẽ xuất hiện khi người nhấn chuột phải vào các điều khiển như: *form*, *label*, *button*, *textbox*, ... Điều khiển *ContextMenuStrip* nằm trong nhóm Menus & Toolbars của cửa sổ Toolbox như hình 5.12



Hình 5.12: Điều khiển *ContextMenuStrip* trong cửa sổ Toolbox

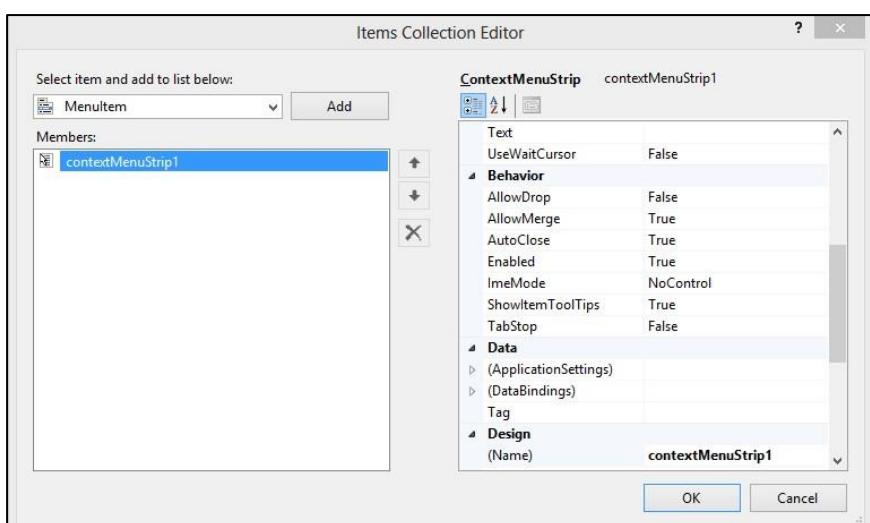
Phần lớn các điều khiển trong cửa sổ Toolbox đều hỗ trợ thuộc tính *ContextMenuStrip*. Do đó muốn menu Popup xuất hiện trên điều khiển nào thì chỉ cần khai báo thuộc tính *ContextMenuStrip* của điều khiển đó là một điều khiển *ContextMenuStrip*.

Giống như điều khiển *MenuStrip*, điều khiển *ContextMenuStrip* cũng cho phép lập trình viên xây dựng menu theo 4 dạng: *MenuItem*, *ComboBox*, *TextBox*, *Separator*. Các thuộc tính, phương thức và sự kiện của 4 dạng menu trên điều khiển *ContextMenuStrip* cũng tương tự như trên điều khiển *MenuStrip* (xem mục 5.2 của chương).

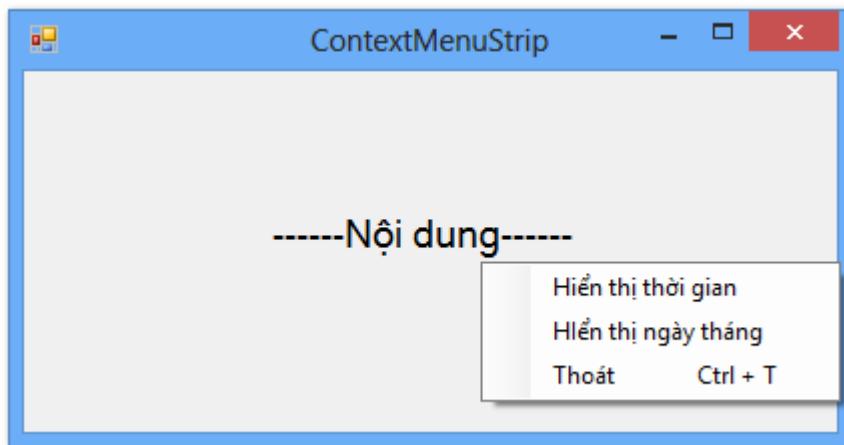
- Một số thuộc tính thường dùng của *ContextMenuStrip*:

Bảng 5.5: Bảng mô tả các thuộc tính thường dùng của *ContextMenuStrip*

Thuộc tính	Mô tả
<i>Items</i>	Thêm các menu con, kiểu menu con thuộc một trong bốn dạng: <i>Separator</i> , <i>MenuItem</i> , <i>ComboBox</i> , <i>TextBox</i> . Lập trình viên có thể thêm các menu trong cửa sổ <i>Items Collection Edition</i> . Cửa sổ <i>Items Collection Edition</i> có thể mở lên bằng cách nhấp chuột trái vào biểu tượng  của thuộc tính <i>Items</i> trong cửa sổ <i>Properties</i> .
<i>RightToLeft</i>	Mang giá trị True: Trình bày menu từ phải qua trái Mang giá trị False: Trình bày menu từ trái qua phải



Ví dụ 5.3: Viết chương trình hiển thị thời gian và tạo *ContextMenuStrip* như hình 5.13.



Hình 5.13: Giao diện chương trình hiển thị thời gian ví dụ 5.3

Yêu cầu:

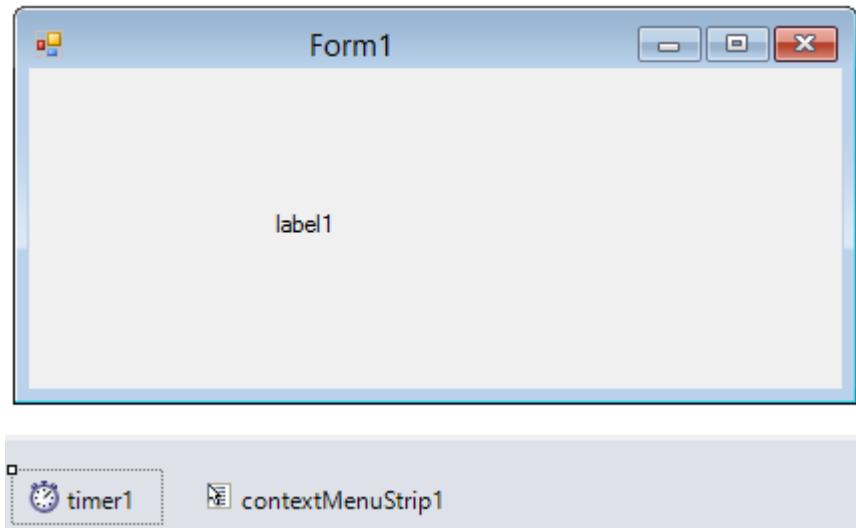
Khi chọn mục “Hiển thị thời gian”: Sẽ hiển thị giờ phút giây trên label lblNoiDung.

Khi chọn mục “Hiển thị ngày tháng”: Sẽ hiển thị ngày tháng năm trên label lblNoiDung.

Khi chọn mục Thoát hoặc nhấn tổ hợp phím Ctrl + T sẽ thoát chương trình.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu của chương trình như hình 5.14



Hình 5.14: Giao diện ban đầu chương trình hiển thị thời gian ví dụ 5.3

Bước 2: Thiết lập giá trị cho các điều khiển từ cửa sổ Properties

- form1:

Thuộc tính Text: “ContextMenuStrip”

- label1:

Thuộc tính Text: “-----Nội dung-----”

Thuộc tính Name: lblNoiDung

Thuộc tính Size: 14

- timer1:

Thuộc tính *Name*: MyTimer

- *contextMenuStrip1*:

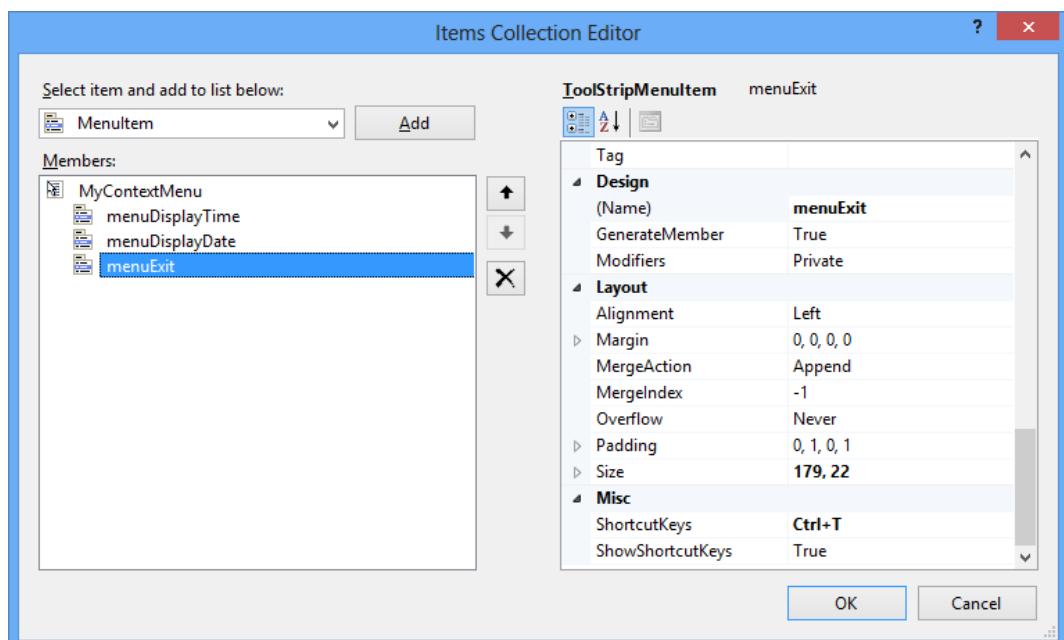
Thuộc tính *Name*: MyContextMenu

Thuộc tính *Items*: Thêm ba menu dạng *MenuItem*: *menuDisplayTime*, *menuDisplayDate* và *menuExit* trong cửa sổ *Items Collection Editor* như hình 5.15.

Thiết lập thuộc tính *ShortCutKeys* của *menuExit*: Ctrl + T

Thiết lập thuộc tính *ShortCutKeyDisplayString* của *menuExit*: Ctrl + T

Thiết lập thuộc tính *ShowShortCutKeys* của *menuExit*: True



Hình 5.15: Cửa sổ *Items Collection Editor*

Bước 3: Viết mã lệnh cho các điều khiển

- Khai báo biến:

```
int chon = 3;  
DateTime dt = new DateTime();
```

- Sự kiện *Click* của *menuDisplayTime*:

```
private void menuDisplayTime_Click(object sender, EventArgs e)  
{  
    chon = 0;  
}
```

- Sự kiện *Click* của *menuDisplayDate*:

```
private void menuDisplayDate_Click(object sender, EventArgs e)  
{  
    chon = 1;  
    lblHienThi.Text = dt.Date.ToString("dd/MM/yyyy");  
}
```

- Sự kiện *Click* của menuExit:

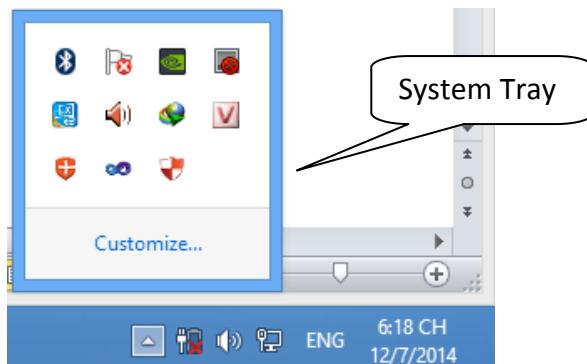
```
private void menuDisplayTime_Click(object sender, EventArgs e)
{
    chon = 0;
}
```

- Sự kiện *Tick* của MyTimer:

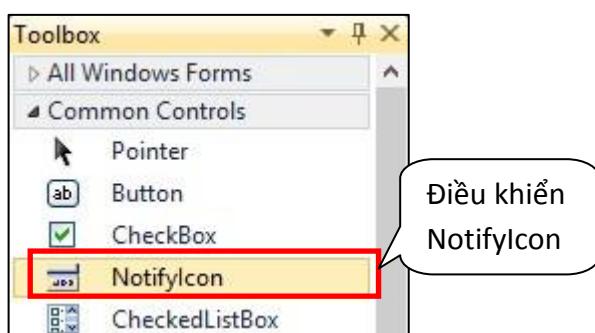
```
private void MyTimer_Tick(object sender, EventArgs e)
{
    dt = DateTime.Now;
    if (chon == 0)
    {
        lblHienThi.Text = dt.Hour + ":" + dt.Minute + ":" + dt.Second;
    }
}
```

## 5.4. Điều khiển NotifyIcon

Điều khiển *NotifyIcon* cho phép lập trình viên trình bày biểu tượng trên System Tray của màn hình Desktop như hình 5.16. Với việc sử dụng điều khiển *NotifyIcon*, khi người dùng di chuyển chuột đến biểu tượng thì chuỗi khai báo trong thuộc tính *Text* của điều khiển *NotifyIcon* sẽ hiển lên như mô *Tooltip*. Điều khiển *NotifyIcon* nằm trong nhóm Common Controls của cửa sổ Toolbox như hình 5.17



Hình 5.16: System Tray trên màn hình Desktop



Hình 5.17: Điều khiển NotifyIcon

- Một số thuộc tính thường dùng của *NotifyIcon*:

*Bảng 5.6: Bảng mô tả các thuộc tính thường dùng của NotifyIcon*

Thuộc tính	Mô tả
<i>Icon</i>	Biểu tượng sẽ xuất hiện dưới khay hệ thống khi form hiển thị
<i>Text</i>	Chuỗi xuất hiện khi di chuột vào biểu tượng dưới khay hệ thống
<i>ContextMenuStrip</i>	Menu Popup sẽ xuất hiện khi người dùng nhấp chuột phải vào biểu tượng dưới khay hệ thống
<i>Visible</i>	Mang giá trị True: Hiển thị biểu tượng trên System Tray Mang giá trị False: Không hiển thị biểu tượng trên System Tray

Để hiển thị một *NotifyIcon* trên System Tray, lập trình viên bắt buộc phải thiết lập một biểu tượng nào đó cho thuộc tính *Icon* và thuộc tính *Visible* phải mang giá trị True.

- Một số sự kiện thường dùng của *NotifyIcon*:

*Bảng 5.7: Bảng mô tả các thuộc tính thường dùng của NotifyIcon*

Sự kiện	Mô tả
<i>MouseClick</i>	Phát sinh khi người dùng nhấp chuột chọn vào biểu tượng được thiết lập trong thuộc tính <i>Icon</i>

Thông thường các biểu tượng trên System Tray được sử dụng bằng sự kiện *MouseClick* của chuột. Với sự kiện *MouseClick* khi nhấp chuột phải sẽ hiển thị một *ContextMenuStrip*, khi nhấp chuột trái sẽ mở chương trình được liên kết với *NotifyIcon*.

Ví dụ 5.4: Viết chương trình tạo form như hình 5.18.



*Hình 5.18: Giao diện chương trình ví dụ 5.4*

Yêu cầu:

Khi chạy chương trình sẽ hiển thị một biểu tượng 🏠 trên System Tray.

Khi nhấp chuột phải vào biểu tượng sẽ hiển thị một *ContextMenuStrip* như hình 5.19:

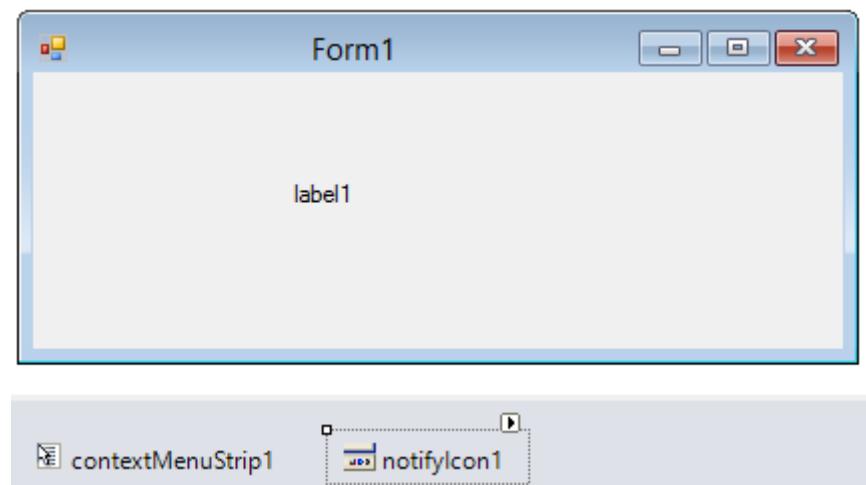


Hình 5.19: ContextMenuStrip trên biểu tượng của NotifyIcon

Khi nhấp chuột trái vào biểu tượng sẽ hiển thị MessageBox với chuỗi hiển thị: “Khoa Công nghệ thông tin, Trường Đại học Lạc Hồng. Xin chào các bạn!”

Hướng dẫn:

Bước 1: Thiết kế giao diện form ban đầu như hình 5.20



Hình 5.20: Giao diện ban đầu ví dụ 5.4

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển ở cửa sổ Properties

- form1:

Thuộc tính *Text*: “NotifyIcon”

Thuộc tính *StartPosition*: FormStartPosition.CenterScreen;

- label1:

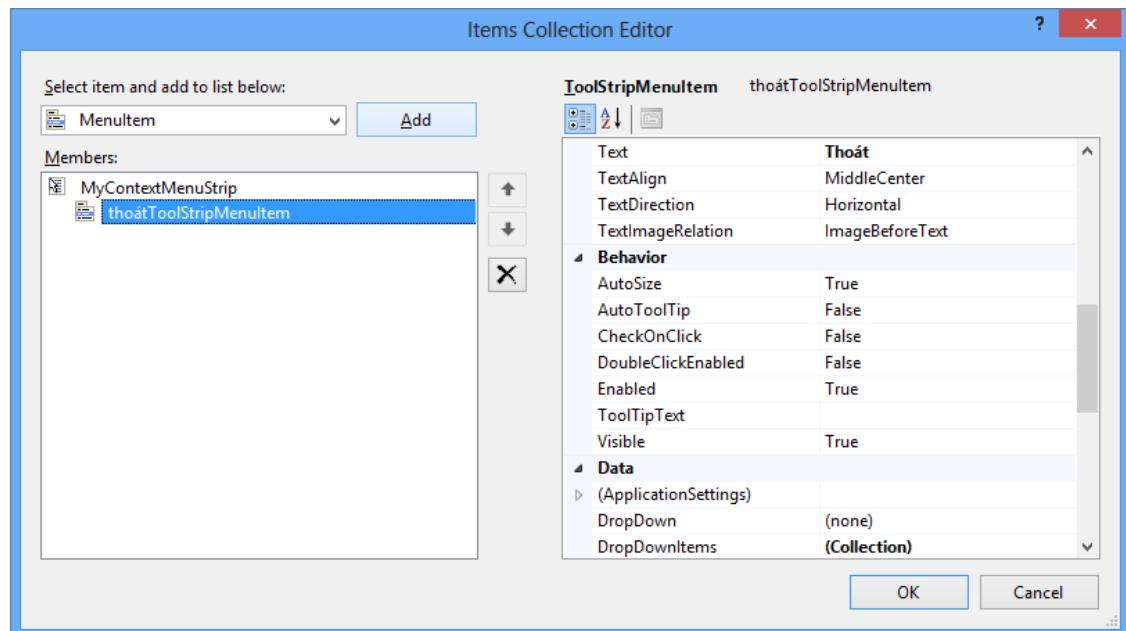
Thuộc tính *Text*: “Khoa Công nghệ thông tin”

Thuộc tính *Size*: 14

- contextMenuStrip1:

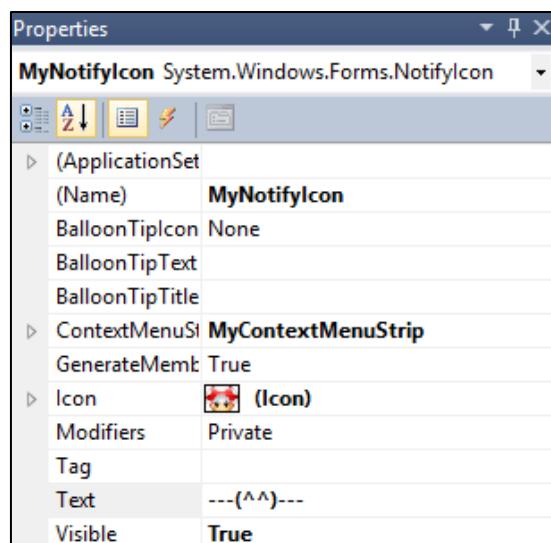
Thuộc tính *Name*: MyContextMenuStrip

Thuộc tính *Items*: Thêm một menu dạng MenuItem trong cửa sổ Items Collection Editor như hình 5.21.



Hình 5.21: Cửa sổ Item Collection Editor ví dụ 5.4

- notifyIcon1: Thiết lập giá trị thuộc tính của NotifyIcon như hình 5.22



Hình 5.22: Thuộc tính của NotifyIcon

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện Click của thoátToolStripMenuItem:

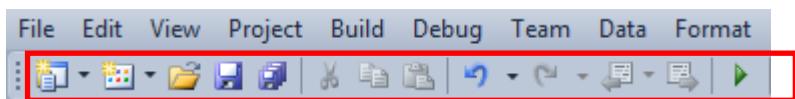
```
private void thoátToolStripMenuItem_Click(object sender,
EventArgs e)
{
    Close();
}
```

- Sự kiện *MouseClick* của *MyNotifyIcon*:

```
private void MyNotifyIcon_MouseClick(object sender,
MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
        MessageBox.Show("Khoa Công nghệ thông tin, Trường
                        Đại học Lạc Hồng\n Xin chào các bạn");
}
```

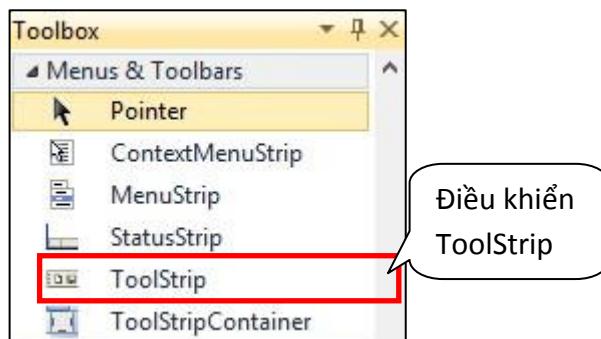
## 5.5. Điều khiển ToolStrip

*ToolStrip* là điều khiển cho phép tạo thanh công cụ trên form. Thông thường trong các ứng dụng Windows Forms, *ToolStrip* thường được bố trí phía dưới điều khiển *MenuStrip* như hình 5.23.



Hình 5.23: Vị trí của điều khiển *ToolStrip* trong Visual Studio 2010

Điều khiển *ToolStrip* nằm trong nhóm Menus & Toolbars của cửa sổ Toolbox như hình 5.24.

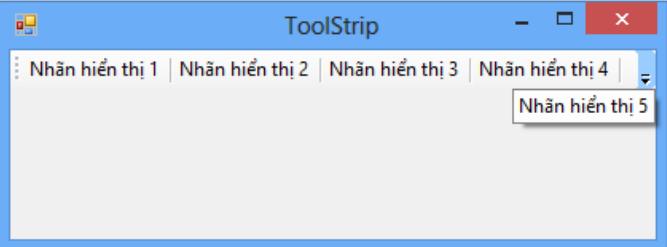
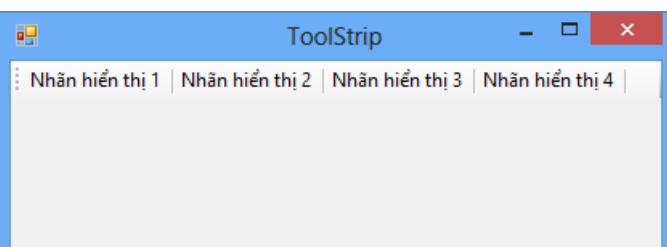


Hình 5.24: *ToolStrip* trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của *ToolStrip*:

Bảng 5.8: Bảng mô tả các thuộc tính thường dùng của *ToolStrip*

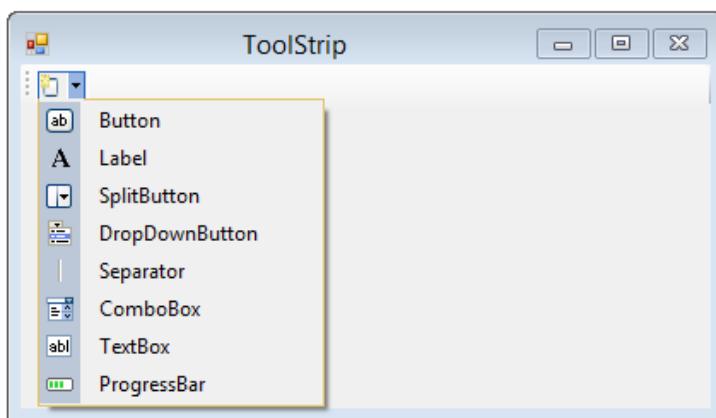
Thuộc tính	Mô tả
<i>Item</i>	Quản lý việc thêm xóa các điều khiển trên <i>ToolStrip</i>
<i>AllowItemReorder</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>Nếu là True: cho phép người dùng sắp xếp lại vị trí của các điều khiển trên <i>ToolStrip</i>, thay đổi vị trí bằng cách giữ phím Alt và nhấn chuột trái vào điều khiển và kéo đến vị trí mới trên <i>ToolStrip</i>.</li> <li>Nếu là False: Các vị trí của điều khiển trên</li> </ul>

	ToolStrip cố định không thể thay đổi bởi người dùng.
<i>AllowMerge</i>	<p>Cho phép người dùng giữ phím Alt và giữ chuột trái vào điều khiển trên ToolStrip này và kéo thả vào một ToolStrip khác.</p> <p>Lưu ý: Thuộc tính này chỉ có hiệu lực khi thuộc tính <i>AllowItemReorder</i> là True</p>
<i>CanOverflow</i>	<p>Mang hai giá trị True và False.</p> <ul style="list-style-type: none"> <li>- Nếu là giá trị True: Khi số lượng điều khiển trong ToolStrip vượt ra khỏi phạm vi kích thước thì những điều khiển này sẽ được thu nhỏ trong biểu tượng  ở góc phải của ToolStrip.</li> </ul>  <ul style="list-style-type: none"> <li>- Nếu là giá trị False: Những điều khiển nằm ngoài phạm vi kích thước sẽ không được thu nhỏ trong biểu tượng  ở góc phải của ToolStrip. Biểu tượng  sẽ không xuất hiện trên ToolStrip.</li> </ul> 
<i>Dock</i>	Quy định vị trí hiển thị của ToolStrip trên form.
<i>LayoutStyle</i>	Kiểu trình bày của ToolStrip

ShowItemTooltips	<p>Mang hai giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép hiển thị chuỗi khai báo trong thuộc tính ToolTipText của mỗi điều khiển chứa trong ToolStrip.</li> <li>- Nếu là False: Chuỗi khai báo trong ToolTipText của các điều khiển chứa trong ToolStrip không được hiển thị.</li> </ul>
------------------	---

### 5.5.1. Các điều khiển chứa trong ToolStrip

Điểm đặc biệt của điều khiển *ToolStrip* là được cấu tạo dưới dạng container cho phép chứa các điều khiển khác như hình 5.25.



Hình 5.25: Các điều khiển ToolStrip chứa

Các điều khiển có thể tạo trên *ToolStrip* bao gồm:

- *ToolStripDropDownButton*: Điều khiển *ToolStripDropDownButton* cho phép lập trình viên tạo một menu dạng sổ xuống và sẽ hiển thị khi người dùng nhấp chuột vào. Các menu được tạo trong điều khiển *ToolStripDropDownButton* là dạng *MenuItem*.
- *ToolStripButton*: Là điều khiển tương tự như điều khiển *Button*. Điều khiển này xuất hiện trên *ToolStrip* ở dạng một biểu tượng (*Icon*). Sự kiện thường dùng của *ToolStripButton* là sự kiện *Click*
- *ToolStripLabel*: Là điều khiển có chức năng như chức năng của điều khiển *Label* và điều khiển *LinkLabel*. Khi thuộc tính *IsLink* của *ToolStripLabel* là true thì điều khiển *ToolStripLabel* trở thành *LinkLabel*, khi thuộc tính *IsLink* là false *ToolStripLabel* như một điều khiển *Label* thông thường.
- *ToolStripProgressBar*: Tương tự như điều khiển *ProgressBar* và nằm trên *ToolStrip*.

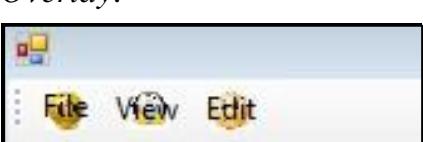
- *ToolStripSeparator*: Là điều khiển dùng để hiển thị dấu gạch phân cách, giúp phân cách các điều khiển trong *ToolStrip* với nhau để hiển thị một cách dễ nhìn hơn.
- *ToolStripComboBox*: Điều khiển *ToolStripComboBox* có các thuộc tính, phương thức và sự kiện tương tự như điều khiển *ComboBox*, nhưng được đặt trên *MenuStrip*. Dạng hiển thị của *ToolStripComboBox* cũng có các dạng như: *Simple*, *DropDown* hoặc *DropDownList*.
- *ToolStripTextBox*: Tương tự như điều khiển *TextBox*, được đặt trên *MenuStrip*. Người dùng có thể gõ chuỗi ký tự vào ô văn bản, khi đó chuỗi ký tự đó sẽ được truyền vào cho thuộc tính *ToolStripTextBox.Text*. Điểm khác biệt chính của *ToolStripTextBox* và *TextBox* là điều khiển *ToolStripTextBox* không có thuộc tính *MultiLine*, do đó ô văn bản của *ToolStripTextBox* chỉ có một dòng.
- *ToolStripSplitButton*: Là điều khiển kết hợp chức năng của *ToolStripButton* và *ToolStripDropDownButton*. Điều khiển này hiển thị một nút mà người dùng có thể nhấn nút để thực thi mã lệnh, ngoài ra cũng cho phép hiển thị một menu theo dạng sổ xuống như của *ToolStripDropDownButton*. Lập trình viên có thể sử dụng sự kiện *ToolStripSplitButton.Click* để viết mã lệnh khi nhấn nút bấm hoặc có thể viết mã lệnh cho mỗi sự kiện *ToolStripMenuItem.Click*

➤ Hiển thị hình trên các điều khiển của *ToolStrip*

Các điều khiển *ToolStripButton*, *ToolStripSplitButton* và *ToolStripDropDownButton* có thể hiển thị hình, chuỗi mô tả hoặc cả hai. Bảng 5.9 mô tả các thuộc tính thường sử dụng để biểu diễn hình và chuỗi mô tả trên các điều khiển *ToolStripButton*, *ToolStripSplitButton*, *ToolStripDropDownButton* của *ToolStrip*

Bảng 5.9: Bảng mô tả thuộc tính của điều khiển trên *ToolStrip*

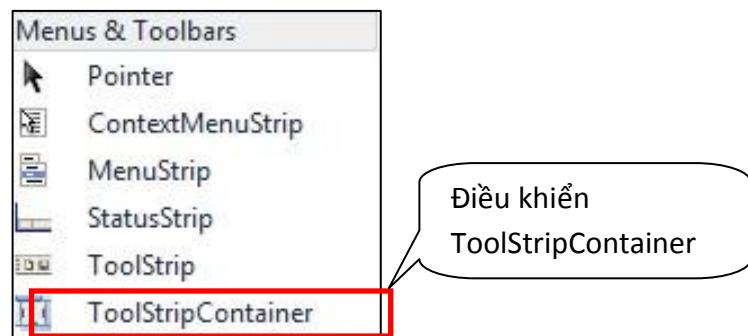
Thuộc tính	Mô tả
<i>DisplayStyle</i>	Gồm các giá trị: <i>None</i> , <i>Text</i> , <i>Image</i> , <i>ImageAndText</i> <ul style="list-style-type: none"> <li>- <i>None</i>: Không hiển thị Image và Text trên điều khiển.</li> <li>- <i>Text</i>: Chỉ hiển thị chuỗi mô tả được thiết lập trong thuộc tính <i>Text</i> trên điều khiển (không hiển thị hình).</li> <li>- <i>Image</i>: Chỉ hiển thị hình được thiết lập trong thuộc tính <i>Image</i> trên điều khiển (không hiển thị chuỗi mô tả)</li> <li>- <i>ImageAndText</i>: Hiển thị cả hình và chuỗi mô tả</li> </ul>

	trên điều khiển
<i>Image</i>	Cài định hình sẽ hiển thị trên điều khiển
<i>TextImageRelation</i>	Kiểu hiển thị của <i>Text</i> và <i>Image</i> trên điều khiển: <i>Overlay</i> , <i>ImageAboveText</i> , <i>TextAboveImage</i> , <i>TextBeforeImage</i> , <i>ImageBeforeText</i> - <i>ImageBeforeText</i> :  - <i>TextBeforeImage</i> :  - <i>ImageAboveText</i> :  - <i>TextAboveImage</i> :  - <i>Overlay</i> : 
<i>ImageScaling</i>	Kích thước của hình sẽ vừa với điều khiển hay không. - <i>None</i> : Hình sẽ hiển thị đúng với kích thước thật - <i>SizeToFit</i> : Hình sẽ hiển thị với kích thước bằng với kích thước của điều khiển
<i>ImageTransparentColor</i>	Làm trong suốt màu của hình

### 5.5.2. ToolStripContainer

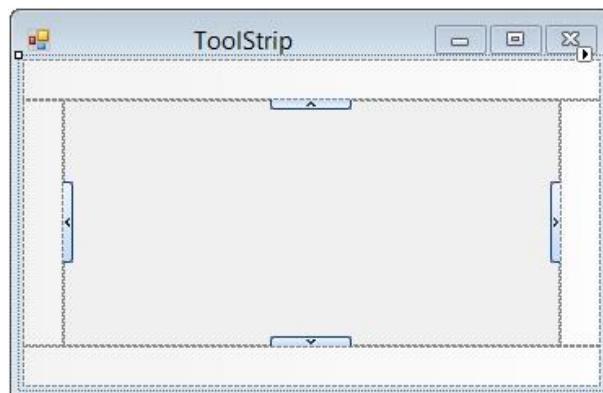
*ToolStripContainer* là dạng điều khiển thiết kế, chứa các điều khiển *ToolStrip* bên trong. Điểm đặc biệt là khi sử dụng *ToolStripContainer*, người dùng có thể kéo và di chuyển các *ToolStrip* trên các cạnh của form (nếu trên cạnh form có sử dụng

*ToolStripContainer*). ToolStripContainer nằm trong nhóm Menus & Toolbars của cửa sổ Toolbox như hình 5.26.



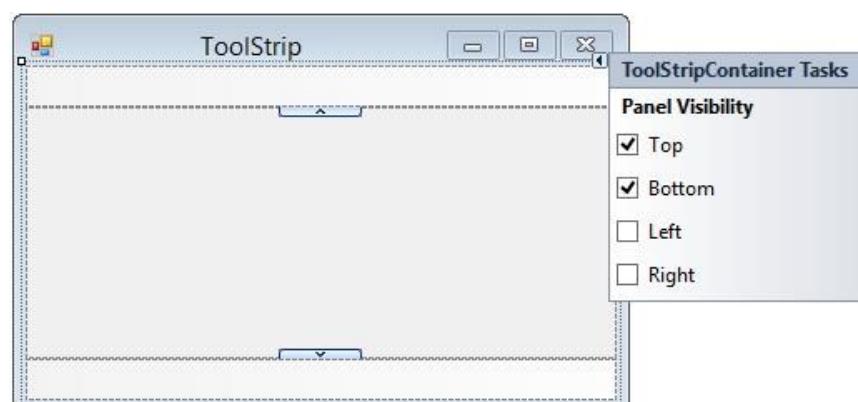
Hình 5.26: Điều khiển ToolStripContainer

*ToolStripContainer* được cấu tạo bởi 5 panel: 4 panel dạng *ToolStripPanel* được bố trên nằm trên 4 cạnh của form và 1 panel dạng *ContentPanel* nằm giữa form. Thông thường khi sử dụng *ToolStripContainer* trên form thì thuộc tính *Dock* của điều khiển này được thiết lập là *Fill* như hình 5.27.



Hình 5.27: ToolStripContainer có thuộc tính Dock thiết lập giá trị Fill

Lập trình viên có thể loại bỏ các panel dạng *ToolStripPanel* nếu không muốn panel bố trí trên một cạnh nào đó của form. Ví dụ nếu chỉ muốn sử dụng *ToolStripContainer* không có hai panel bên cạnh phải và cạnh trái của form có thể thiết lập như hình 5.28.



Hình 5.28: ToolStripContainer không có hai panel bên cạnh trái và cạnh phải của form

Có thể loại bỏ panel như hình 5.28 hoặc thiết lập giá trị các thuộc tính: *LeftToolStripPanelVisible*, *RightToolStripPanelVisible*, *TopToolStripPanelVisible*, *BottomToolStripPanelVisible*

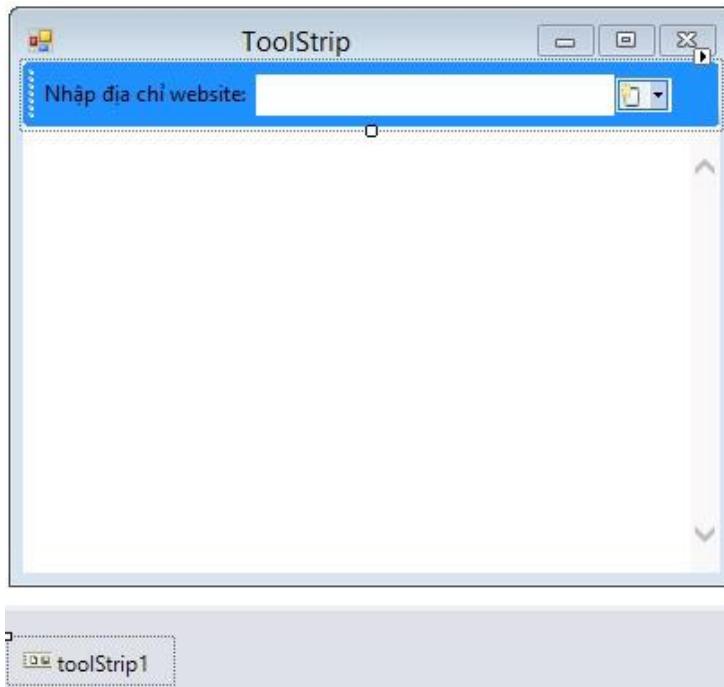
- Một số thuộc tính thường dùng của *ToolStripContainer*:

*Bảng 5.10: Bảng mô tả thuộc tính của ToolStripContainer*

Thuộc tính	Mô tả
<i>Dock</i>	Kiểu hiển thị của <i>ToolStripContainer</i> . <ul style="list-style-type: none"> <li>- <i>Fill</i>: Hiển thị bằng với kích thước form</li> <li>- <i>Top</i>: Hiển thị nằm sát cạnh trên của form</li> <li>- <i>Bottom</i>: Hiển thị nằm sát cạnh dưới của form</li> <li>- <i>Left</i>: Hiển thị sát cạnh trái của form</li> <li>- <i>Right</i>: Hiển thị sát cạnh phải của form</li> </ul> Lưu ý: Thường thì thuộc tính <i>Dock</i> được thiết lập giá trị <i>Fill</i>
<i>LeftToolStripPanelVisible</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Hiển thị panel bên cạnh trái của form</li> <li>- Nếu là False: Không hiển thị panel bên cạnh trái của form</li> </ul>
<i>RightToolStripPanelVisible</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Hiển thị panel bên cạnh phải của form</li> <li>- Nếu là False: Không hiển thị panel bên cạnh phải của form</li> </ul>
<i>TopToolStripPanelVisible</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Hiển thị panel cạnh trên của form</li> <li>- Nếu là False: Không hiển thị panel cạnh trên của form</li> </ul>
<i>BottomToolStripPanelVisible</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Hiển thị panel cạnh dưới của form</li> <li>- Nếu là False: Không hiển thị panel cạnh dưới của form</li> </ul>

Ví dụ 5.5: Thiết kế chương trình như hình 5.29.

Yêu cầu sau khi nhập địa chỉ website trong TextBox của ToolStrip và nhấn Enter thì sẽ hiển thị website có địa chỉ vừa nhập.

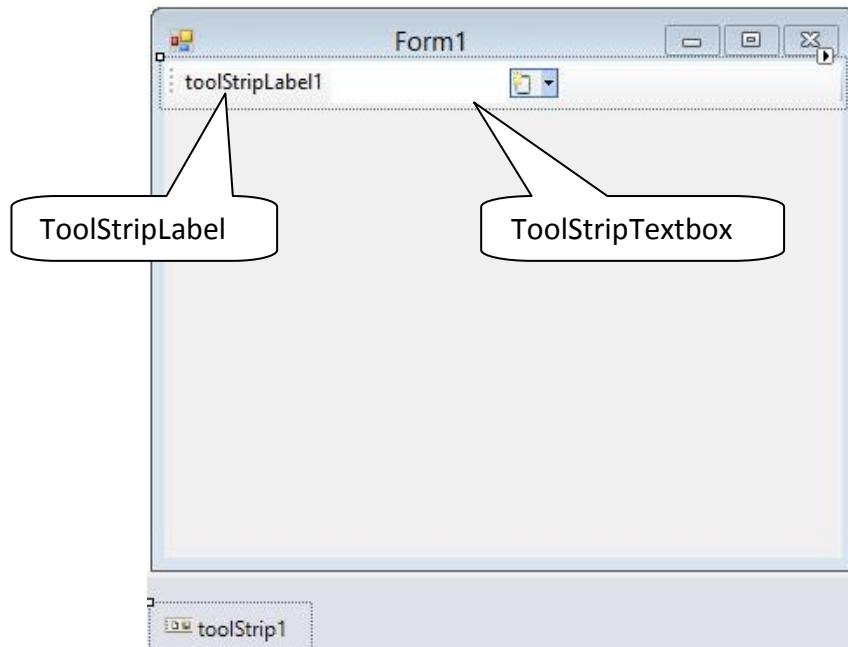


Hình 5.29: Giao diện form hiển thị Website ví dụ 5.5

Hướng dẫn:

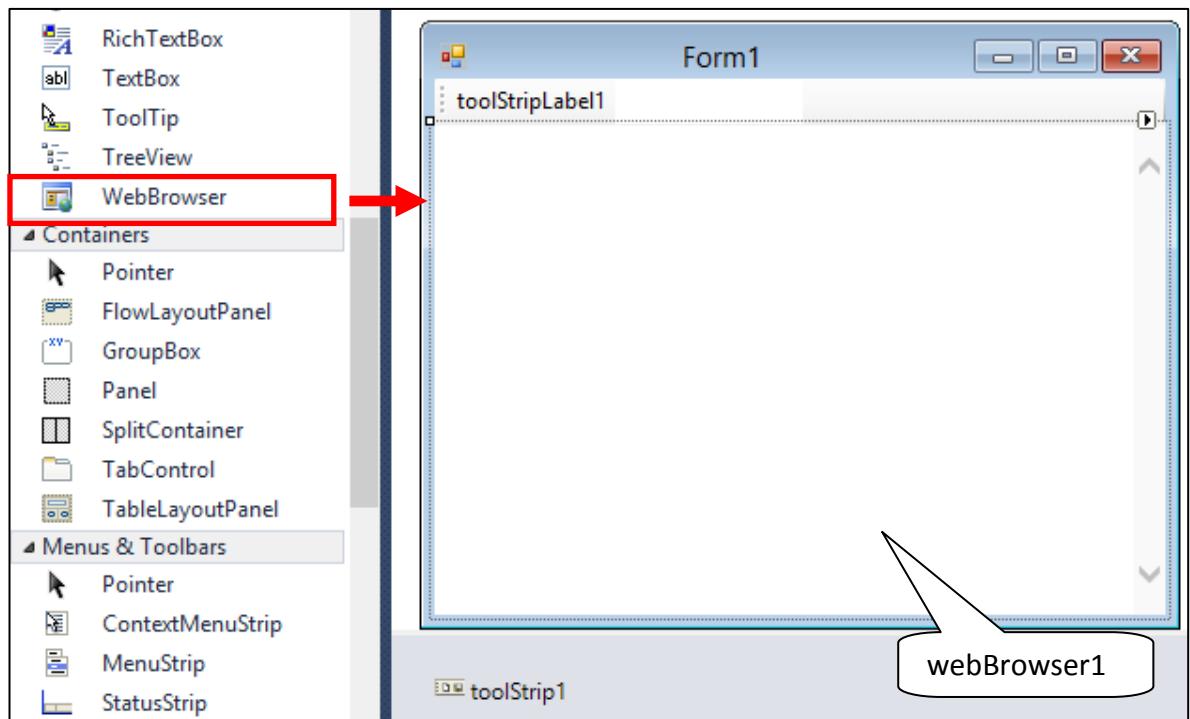
Bước 1: Thiết kế giao diện chương trình.

- Kéo ToolStrip từ cửa sổ Toolbox vào form. Sau đó thêm hai điều khiển ToolStripLabel và ToolStripTextBox trên ToolStrip như hình 5.30:



Hình 5.30: Giao diện form sau khi thêm ToolStrip

- Kéo điều khiển WebBrowser từ cửa sổ Toolbox và form như hình 5.31. WebBrowser là điều khiển cho phép hiển thị website trên form.



Hình 5.31: Giao diện form sau khi thêm điều khiển WebBrowser

Bước 2: Thiết lập các giá trị cho thuộc tính của điều khiển trong cửa sổ Properties

- form1:

Thuộc tính Text: "ToolStrip"

Thuộc tính Size: 405, 324

- toolStrip1:

Thuộc tính AutoSize: False

Thuộc tính Size: 389, 37

Thuộc tính BackColor: Color.DodgerBlue

- toolStripLabel1:

Thuộc tính Text: "Nhập địa chỉ website:"

- toolStripTextbox:

Thuộc tính Name: txtAddress

Thuộc tính AutoSize: False

Thuộc tính Size: 200, 25

- webBrowser1:

Thuộc tính Name: MyWebSite

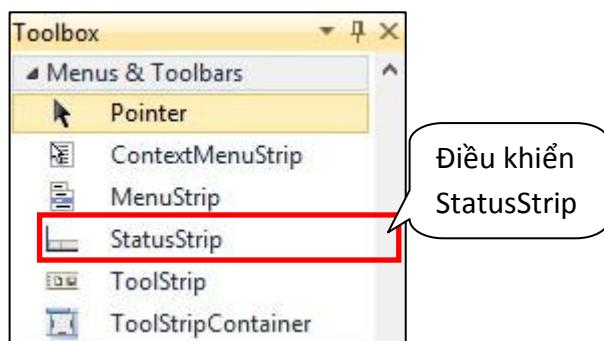
Bước 3: Viết mã lệnh cho các điều khiển trên form

- Sự kiện *KeyDown* của *txtAddress*:

```
private void txtAddress_KeyDown(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
        MyWebsite.Navigate(txtAddress.Text);
}
```

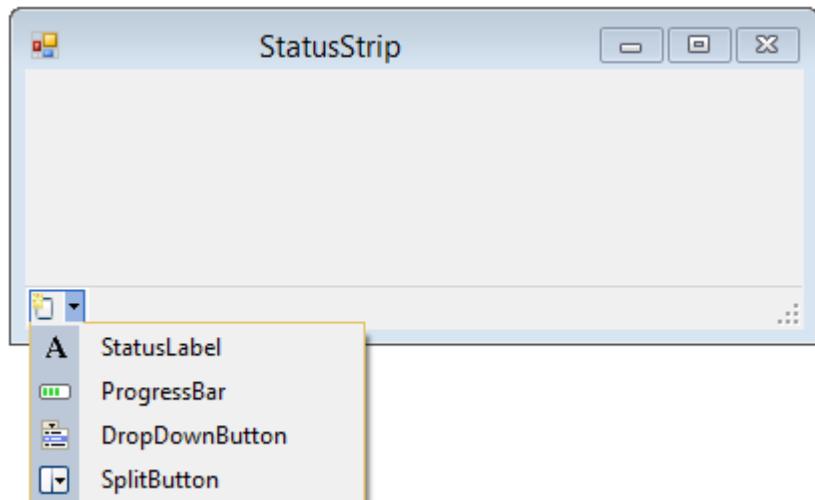
## 5.6. Điều khiển StatusStrip

*StatusTrip* sử dụng để hiển thị thông tin trạng thái của ứng dụng, *StatusStrip* nằm ở vị trí dưới cùng của form. *StatusStrip* nằm trong nhóm Menus & Toolbars của cửa sổ Toolbox như hình 5.32.



Hình 5.32: *StatusStrip* trong cửa sổ Toolbox

Cũng giống như *ToolStrip*, *StatusStrip* cũng có thể chứa các điều khiển khác như hình 5.33.



Hình 5.33: Các điều khiển có thể chứa trên *StatusStrip*

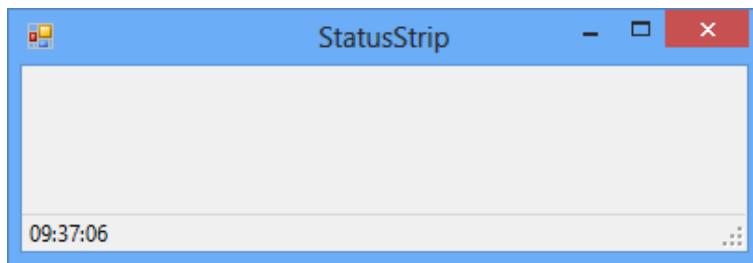
Các điều khiển: *StatusLabel*, *SplitButton*, *DropDownButton* hay *ProgressBar* trên *StatusStrip* có chức năng tương tự như trên *ToolStrip*. Trong các điều khiển trên, điều khiển thường sử dụng trên *StatusStrip* nhất là *StatusLabel*

- Một số thuộc tính thường dùng của *StatusStrip*:

Bảng 5.11: Bảng mô tả thuộc tính của StatusStrip

Thuộc tính	Mô tả
Item	Quản lý việc thêm, xóa các điều khiển trên StatusStrip như: <i>StatusLabel</i> , <i>SplitButton</i> , <i>DropDownButton</i> , <i>ProgressBar</i> .
TextDirection	Chiều hiển thị của chuỗi mô tả trong thuộc tính Text.
LayoutStyle	Vị trí hiển thị của StatusStrip.

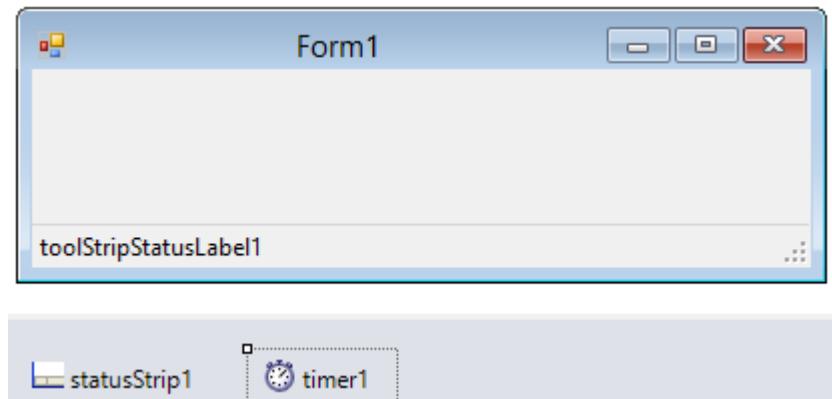
Ví dụ 5.6: Viết chương trình hiển thị thời gian trên StatusStrip của form như hình 5.34



Hình 5.34: Giao diện chương trình hiển thị thời gian trên StatusStrip ví dụ 5.6

Hướng dẫn:

Bước 1: Thiết kế giao diện chương trình: Kéo điều khiển *StatusStrip* và *Timer* từ cửa sổ Toolbox và form. Sau đó thêm điều khiển *StatusLabel* trên *StatusStrip* như hình 5.35.



Hình 5.35: Giao diện form sau khi thêm StatusStrip

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties

- form1:

Thuộc tính Text: "StatusStrip"

Thuộc tính Size: 405, 138

- timer1:

Thuộc tính Name: MyTimer

Thuộc tính Interval: 1000

Thuộc tính Enable: True

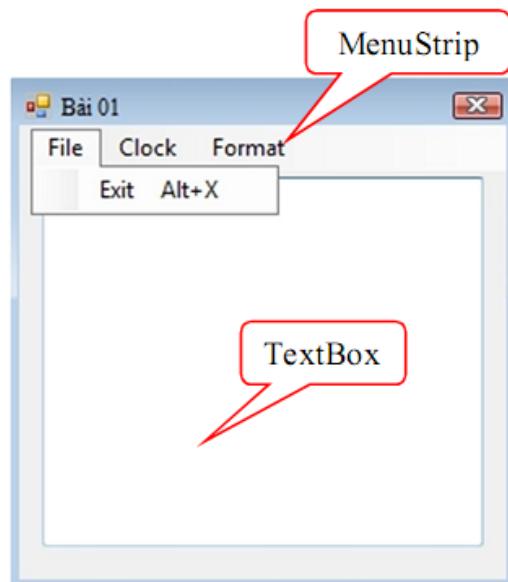
- toolStripStatusLabel1:  
Thuộc tính Name: MyStatusLabel
- Bước 3: Viết mã lệnh cho điều khiển
- Sự kiện *Tick* của MyTimer:

```
private void MyTimer_Tick(object sender, EventArgs e)
{
    MyStatusLabel.Text = DateTime.Now.ToString("hh:mm:ss");
}
```

## 5.7. Bài tập cuối chương

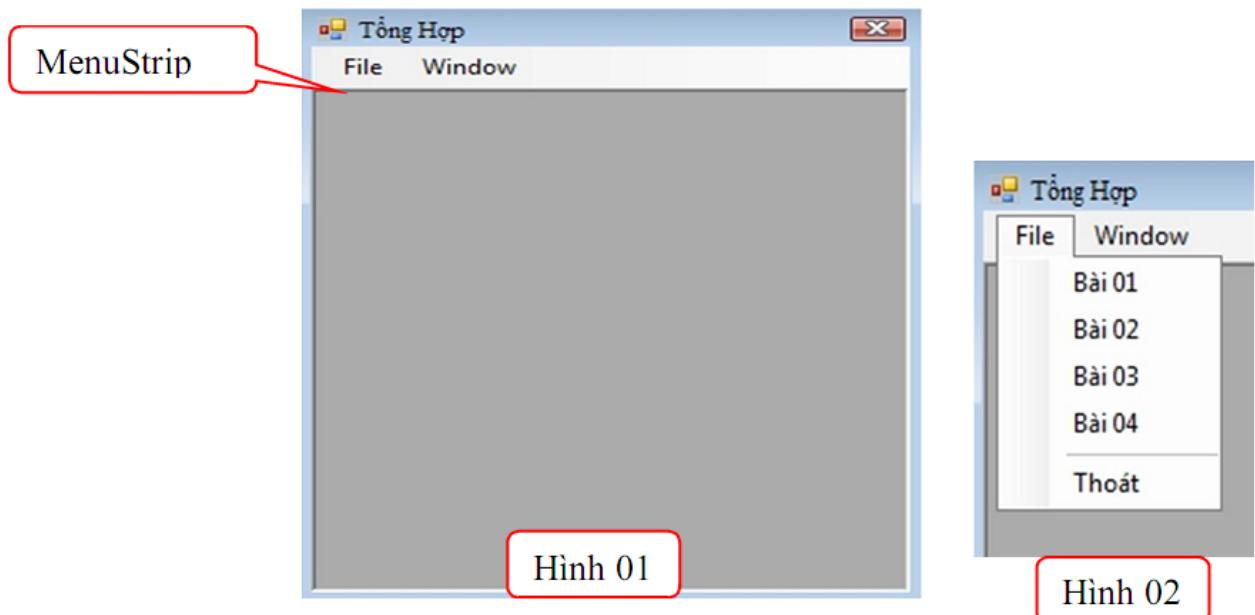
Câu 1: Viết chương trình soạn thảo văn bản có giao diện như hình 5.36. với các yêu cầu sau:

- Trên Form có một TextBox.
- Menu File có chứa mục Exit để thoát chương trình.
- Menu Clock có chứa mục:
  - o Date: cho xuất hiện ngày hiện tại trong TextBox.
  - o Time: cho hiện giờ hiện tại trong TextBox.
- Menu Format có chứa mục:
  - o Font: chọn font cho TextBox.
  - o Color: chọn màu cho văn bản trong TextBox.
  - o Align có chứa các mục:
    - o Left : canh lề trái cho TextBox.
    - o Right: canh lề phải cho TextBox.
    - o Center: canh giữa cho TextBox.



Hình 5.36: Giao diện chương trình soạn thảo văn bản

Câu 2: Viết chương trình gồm các bài tập có giao diện như hình 5.37.

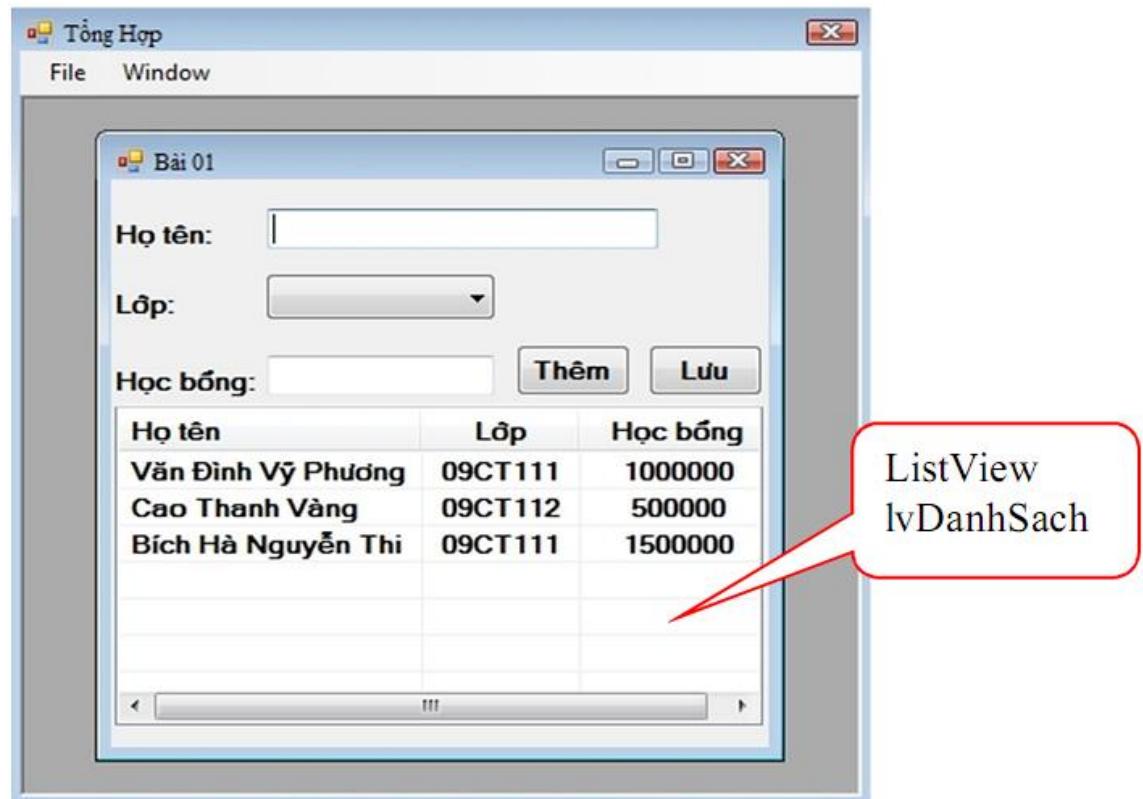


Hình 5.37: Giao diện form các bài tập

Yêu cầu:

Khi nhấn F5 sẽ được form hiển thị như hình 01. Nhấp chuột chọn Menu sẽ được các MenuItem hiển thị như hình 02.

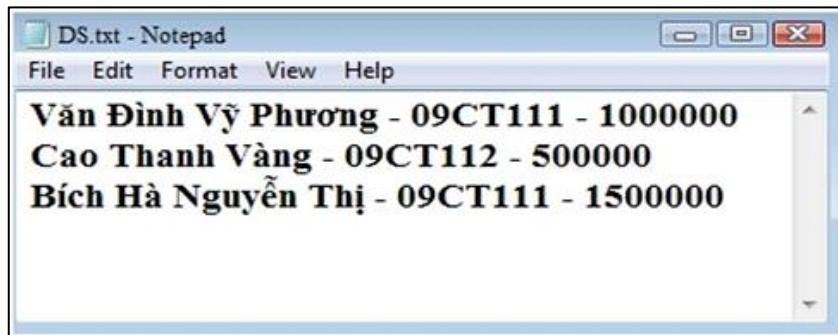
- Khi chọn MenuItem “Bài 01” sẽ hiển thị giao diện như hình 5.38.



Hình 5.38: Giao diện bài tập 01

- Cho người dùng nhập Họ tên, Chọn lớp, nhập học bổng.

- Khi nhấn nút Thêm thì kiểm tra xem các thông tin được nhập (Họ tên, Học bổng) và chọn (Lớp) chưa? Nếu chưa, thông báo cho người dùng nhập, chọn lại. Ngược lại thì thêm các thông tin vào ListView lvDanhSach
- Khi nhấn nút Lưu thì lưu danh sách các sinh viên cấp học bổng vào một tập tin văn bản (\*.txt) với dạng như hình 5.39.



Hình 5.39: Giao diện tập tin văn bản khi nhấn nút lưu

- Khi chọn MenuItem “Bài 02” sẽ hiển thị giao diện như hình 5.40.  
Yêu cầu:
  - Người dùng chọn sở thích và màu thích của mình.
  - Khi nhấn nút Sở Thích Của Bạn thì xuất MessageBox các sở thích được chọn như hình 5.41.
  - Khi nhấn nút Màu Bạn Thích thì xuất MessageBox màu được chọn như hình 5.42.



Hình 5.40: Giao diện bài tập 02



Hình 5.41: MessageBox sở thích

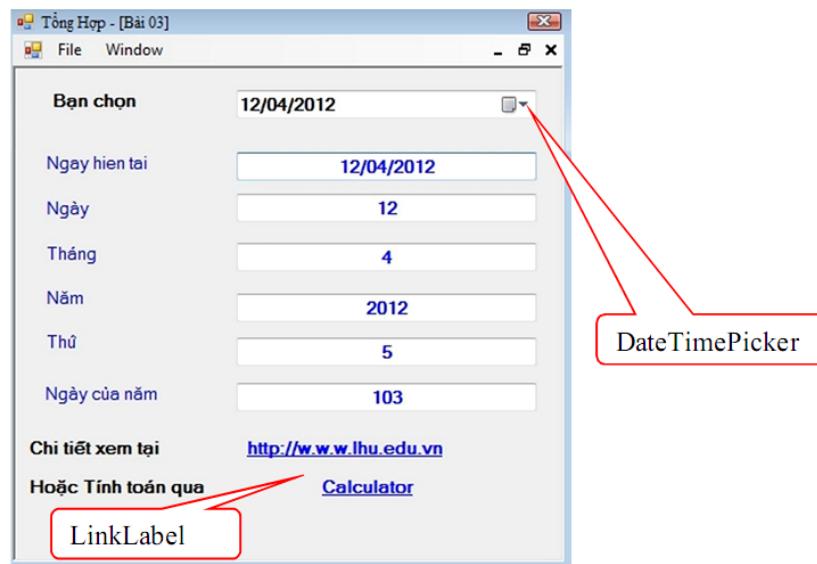


Hình 5.42: MessageBox màu đã chọn

- Khi chọn MenuItem “Bài 03” sẽ hiển thị giao diện như hình 5.43.

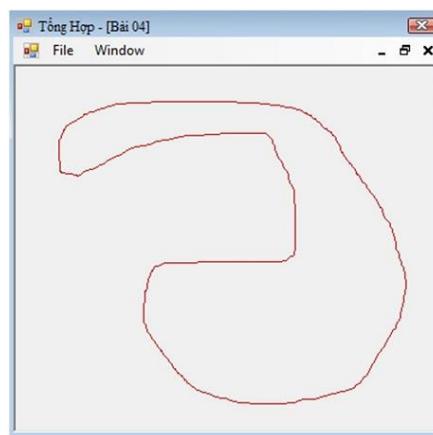
Yêu cầu:

- Khi người dùng chọn giá trị trong DateTimePicker thì hiển thị các giá trị tương ứng vào các TextBox bên dưới.
- Khi người dùng nhấn LinkLabel sẽ mở ứng dụng tương ứng.



Hình 5.43: Giao diện bài tập 03

- Khi chọn MenuItem “Bài 04” sẽ hiển thị giao diện như hình 5.44



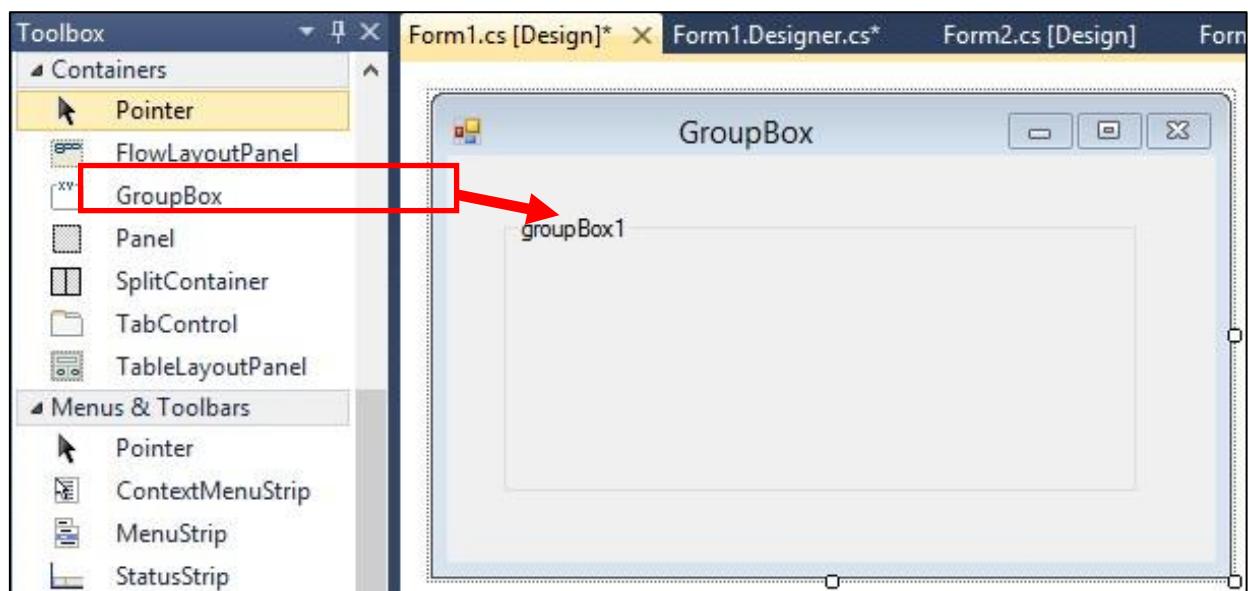
Hình 5.44: Giao diện bài tập 04

- Cho phép người dùng sử dụng chuột trái để vẽ trên form

# CHƯƠNG 6: ĐIỀU KHIỂN CHÚA CÁC ĐIỀU KHIỂN KHÁC

## 6.1. Điều khiển GroupBox

GroupBox là dạng điều khiển chúa, có thể chứa các điều khiển khác hiển thị trên form, giúp cho việc thiết kế giao diện của form dễ nhìn và khoa học hơn. *GroupBox* không hỗ trợ thanh trượt (ScrollBar). *GroupBox* có thể có tiêu đề hiển thị, tiêu đề này được thiết lập trong thuộc tính *Text*. Nếu không muốn hiển thị tiêu đề, lập trình viên có thể thiết lập chuỗi rỗng trong thuộc tính *Text*. Điều khiển *GroupBox* được đặt trong nhóm *Containers* của cửa sổ *Toolbox* như hình 6.1.



Hình 6.1: Điều khiển *GroupBox* trong cửa sổ *Toolbox*

Thông thường *GroupBox* sử dụng để nhóm điều khiển *RadioButton* (xem mục 3.5.2 của chương 3).

- Một số thuộc tính thường dùng của *GroupBox*:

Bảng 6.1: Bảng mô tả các thuộc tính của *GroupBox*

Thuộc tính	Mô tả
<i>Name</i>	Đặt tên cho <i>GroupBox</i>
<i>Text</i>	Chuỗi hiển thị
<i>Font</i>	Thiết lập kiểu chữ, kích thước chữ, ..
<i>ForeColor</i>	Thiết lập màu chữ hiển thị
<i>BackColor</i>	Thiết lập màu nền của <i>GroupBox</i>
<i>Visible</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"><li>- Nếu là True: Hiển thị <i>GroupBox</i></li><li>- Nếu là False: Không hiển thị <i>GroupBox</i></li></ul>
<i>AutoSize</i>	Mang giá trị True hoặc False.

	<ul style="list-style-type: none"> <li>- Nếu là True: <i>GroupBox</i> tự động thay đổi kích thước để có thể hiển thị hết các điều khiển chứa bên trong</li> <li>- Nếu là False: <i>GroupBox</i> có kích thước như lập trình viên thiết lập.</li> </ul>
<i>AutoSizeMode</i>	<p>Quy định cách thức điều khiển thay đổi kích thước.</p> <ul style="list-style-type: none"> <li>- <i>GrowAndShrink</i>: <i>GroupBox</i> có thể co và giãn</li> <li>- <i>GrowOnly</i>: Mặc định, chỉ giãn lớn</li> </ul>

Khi thiết lập giá trị thuộc tính cho *GroupBox* trong bảng 6.1, thì những điều khiển như: *RadioButton*, *TextBox*, *Label*, ... nếu nằm trong *GroupBox* cũng sẽ có những giá trị thuộc tính tương tự như của *GroupBox*.

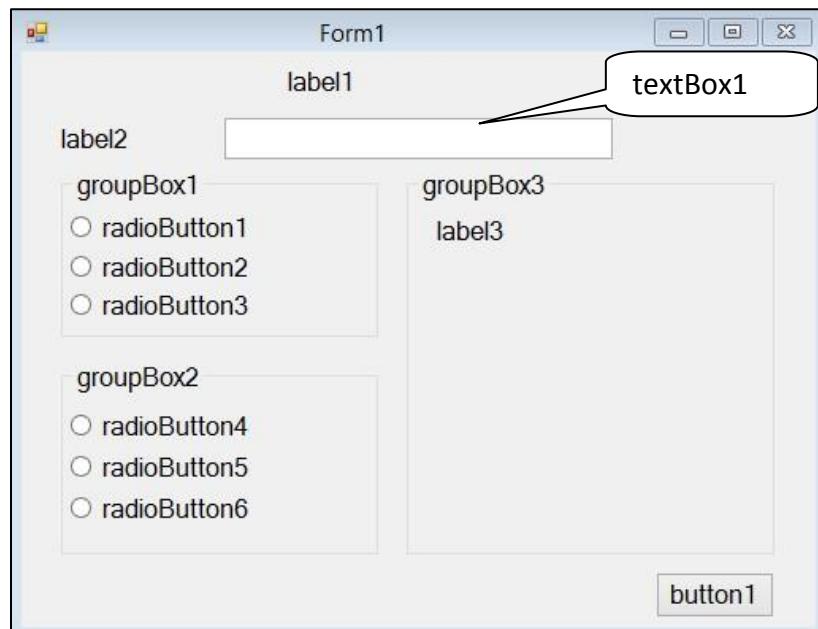
Ví dụ 6.1: Viết chương trình định dạng chuỗi văn bản, thiết kế giao diện chương trình như hình 6.2.



Hình 6.2: Giao diện form định dạng chuỗi ví dụ 6.1

Yêu cầu: Khi người dùng nhập chuỗi văn bản trong *TextBox* thì *Label* “Chuỗi định dạng” sẽ hiển thị chuỗi văn bản vừa nhập. Khi nhấp chuột chọn các *RadioButton* trong *GroupBox* “Màu” và *GroupBox* “Kiểu hiển thị” thì chuỗi định dạng sẽ thay đổi định dạng tương ứng với lựa chọn của người dùng.

Bước 1: Thiết kế giao diện chương trình. Kéo các điều khiển: *Label*, *GroupBox*, *RadioButton*, *TextBox* từ cửa sổ *Toolbox* vào form như hình 6.3.



Hình 6.3: Giao diện form sau khi thêm các điều khiển vào form

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties

- label1:

Thuộc tính Text: “Định dạng chuỗi”

Thuộc tính Size: 14

Thuộc tính FontStyle: Bold

- label2:

Thuộc tính Text: “Nhập chuỗi muốn định dạng:”

- label3:

Thuộc tính Text: “Chuỗi định dạng”

Thuộc tính Name: lblChuoiDinhDang

- textBox1:

Thuộc tính Name: txtNhapChuoi

- groupBox1:

Thuộc tính Text: “Màu”

- groupBox2:

Thuộc tính Text: “Kiểu hiển thị”

- groupBox3:

Thuộc tính Text: “Chuỗi sau khi định dạng”

- radioButton1:

Thuộc tính Name: radXanh

- radioButton2:

Thuộc tính Name: radDo

- radioButton3:

Thuộc tính Name: radDen

- radioButton4:

Thuộc tính Name: radInDam

- radioButton5:

Thuộc tính Name: radInNghieng

- radioButton6:

Thuộc tính Name: radGachChan

- button1:

Thuộc tính Name: btnThoat

Thuộc tính Text: “Thoát”

Bước 3: Viết mã lệnh cho điều khiển

- Sự kiện *Click* của nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *TextChanged* của txtNhapChuoi:

```
private void txtNhapChuoi_TextChanged(object sender, EventArgs e)
{
    lblChuoiDinhDang.Text = txtNhapChuoi.Text;
}
```

- Sự kiện *CheckedChanged* của radXanh:

```
private void radXanh_CheckedChanged(object sender, EventArgs e)
{
    if (radXanh.Checked == true)
    {
        lblChuoiDinhDang.ForeColor = Color.Blue;
    }
}
```

- Sự kiện *CheckedChanged* của radDo:

```
private void radDo_CheckedChanged(object sender, EventArgs e)
{
    if (radDo.Checked == true)
    {
        lblChuoiDinhDang.ForeColor = Color.Red;
    }
}
```

- Sự kiện *CheckedChanged* của radDen:

```
private void radDen_CheckedChanged(object sender, EventArgs e)
{
    if (radDen.Checked == true)
    {
        lblChuoiDinhDang.ForeColor = Color.Black;
    }
}
```

- Sự kiện *CheckedChanged* của radInDam:

```
private void radInDam_CheckedChanged(object sender, EventArgs e)
{
    if (radInDam.Checked == true)
    {
        lblChuoiDinhDang.Font = new
        System.Drawing.Font("Microsoft Sans Serif", 12F,
        System.Drawing.FontStyle.Bold);

    }
}
```

- Sự kiện *CheckedChanged* của radInNghieng:

```
private void radInNghieng_CheckedChanged(object sender,
EventArgs e)
{
    if (radInNghieng.Checked == true)
    {
        lblChuoiDinhDang.Font = new
        System.Drawing.Font("Microsoft Sans Serif", 12F,
        System.Drawing.FontStyle.Italic);

    }
}
```

- Sự kiện *CheckedChanged* của radGachChan:

```
private void radGachChan_CheckedChanged(object sender,
EventArgs e)
{
    if (radGachChan.Checked == true)
    {
        lblChuoiDinhDang.Font = new
        System.Drawing.Font("Microsoft Sans Serif", 12F,
        System.Drawing.FontStyle.Underline);

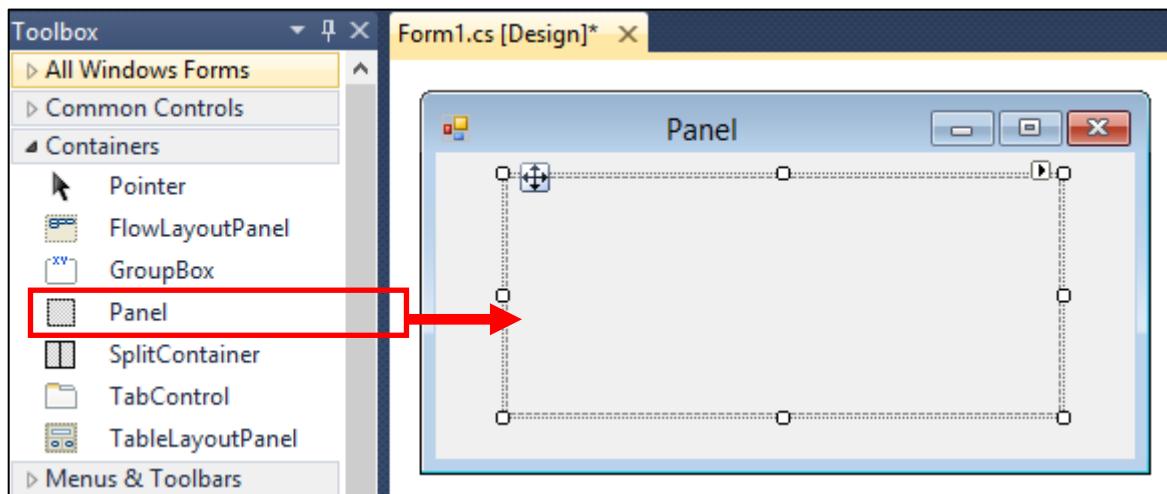
    }
}
```

## 6.2. Điều khiển Panel

Cũng như GroupBox, Panel là một điều khiển dùng để chứa các điều khiển khác. Panel có các thuộc tính AutoSize, AutoSizeMode như GroupBox và thuộc tính đường viền BorderStyle như Label.

Điểm khác biệt của Panel với GroupBox là điều khiển Panel không có tiêu đề mô tả (không có thuộc tính Text) và có thanh trượt ScrollBar ngang và ScrollBar dọc (có thuộc tính AutoScroll).

Điều khiển Panel đặt trong nhóm Containers của cửa sổ Toolbox như hình 6.4



Hình 6.4: Điều khiển Panel trong cửa sổ Toolbox

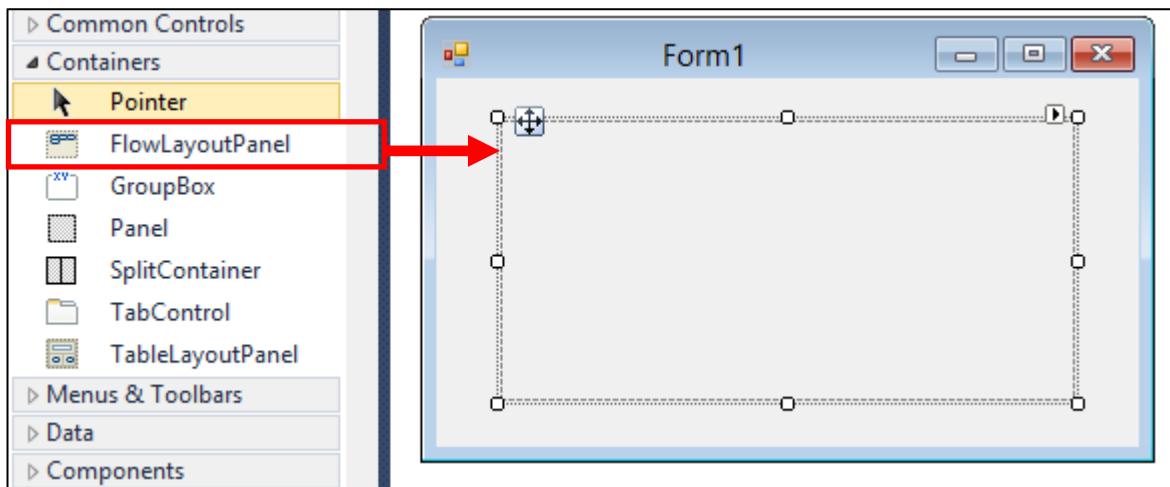
- Một số thuộc tính thường dùng của Panel:

Bảng 6.2: Bảng mô tả các thuộc tính của Panel

Thuộc tính	Mô tả
<i>AutoSize</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"><li>- Nếu là True: Panel tự động xuất hiện thanh trượt khi kích thước Panel không thể hiển thị hết các điều khiển chứa bên trong</li><li>- Nếu là False: Panel sẽ không hiển thị thanh trượt</li></ul>
<i>BorderStyle</i>	Kiểu đường viền của Panel khi hiển thị. Có 3 giá trị: None, FixedSingle và Fixed3D <ul style="list-style-type: none"><li>- None: Không hiển thị đường viền</li><li>- FixedSingle: Quanh Panel sẽ hiển thị một đường viền đơn.</li><li>- Fixed3D: Hiển thị đường viền của Panel dạng 3 chiều.</li></ul>

### 6.3. Điều khiển FlowLayoutPanel

*FlowLayoutPanel* là lớp con của điều khiển *Panel*, do đó có thể chứa các điều khiển khác như *Panel*. Mục đích chính của *FlowLayoutPanel* là giúp bố trí các điều khiển trên form một cách có tổ chức và khoa học. Khi thêm một điều khiển nào đó vào *FlowLayoutPanel* thì *FlowLayoutPanel* sẽ tự động sắp xếp các điều khiển đặt bên trong theo quy tắc định trước và đồng thời cũng thay đổi kích thước của các điều khiển bên trong cho phù hợp với kích thước của *FlowLayoutPanel*. Vì vậy có thể nói điều khiển *FlowLayoutPanel* là điều khiển hỗ trợ tuyệt vời trong việc thiết kế giao diện người dùng. Điều khiển *FlowLayoutPanel* nằm trong nhóm *Containers* của cửa sổ *Toolbox* như hình 6.5.

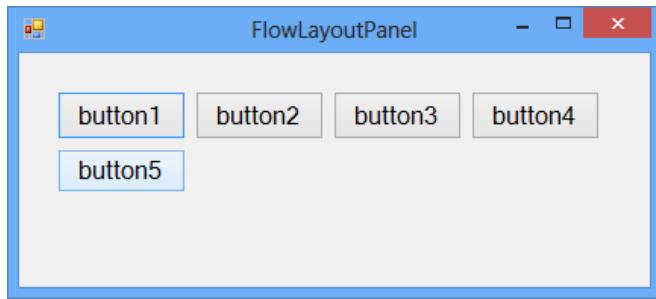


Hình 6.5: Điều khiển *FlowLayoutPanel* trên cửa sổ *Toolbox*

Điều khiển *FlowLayoutPanel* cũng hỗ trợ dạng thanh trượt (*ScrollBar*) như *Panel*, khi thuộc tính *AutoScroll* được thiết lập là *True* thì khi kích thước các điều khiển được chia vượt ngoài kích thước *FlowLayoutPanel*, thì *FlowLayoutPanel* sẽ hiển thị thanh trượt.

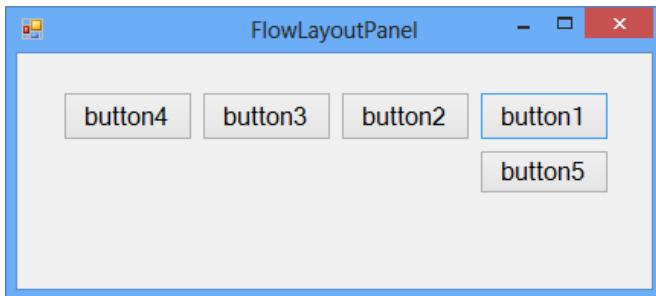
Việc bố trí các điều khiển khi thêm vào *FlowLayoutPanel* như thế nào là do thuộc tính *FlowDirection* quy định. Thuộc tính này mang 4 giá trị cho phép các điều khiển lăn lướt thêm vào theo 4 hướng: từ trái qua phải (*LeftToRight*), từ phải qua trái (*RightToLeft*), từ trên xuống (*TopDown*) và từ dưới lên (*BottomUp*). Các điều khiển được thêm vào đến khi vượt ngoại phạm vi của *FlowLayoutPanel*, nếu muốn các điều khiển tự động bố trí xuống dòng mới hoặc sang một cột mới như các hình 6.6, hình 6.7, hình 6.8 và hình 6.9, thì cần phải thiết lập thuộc tính *WrapContents* là *True*. Còn nếu thuộc tính *WrapContents* là *False* thì *FlowLayoutPanel* sẽ hiển thị thanh trượt (thuộc tính *AutoScroll* là *True*) để hiển thị các điều khiển nằm ngoài phạm vi.

Thuộc tính *FlowDirection* là *LeftToRight*:



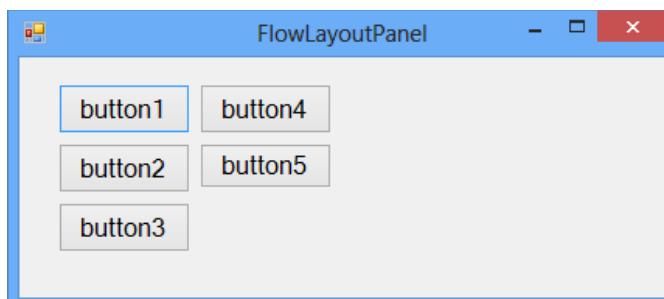
Hình 6.6: *Bố trí điều khiển từ trái sang phải với WrapContents là True*

Thuộc tính *FlowDirection* là *RightToLeft*:



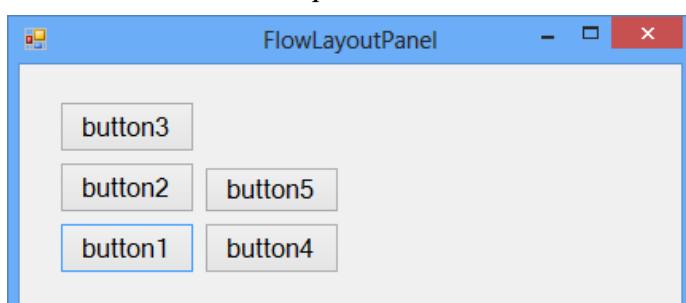
Hình 6.7: *Bố trí điều khiển từ phải sang trái với WrapContents là True*

Thuộc tính *FlowDirection* là *TopDown*:



Hình 6.8: *Bố trí điều khiển từ trên xuống dưới với WrapContents là True*

Thuộc tính *FlowDirection* là *BottomUp*:



Hình 6.9: *Bố trí điều khiển từ dưới lên trên với WrapContents là True*

- Một số thuộc tính thường dùng của *FlowLayoutPanel*:

Bảng 6.3: Bảng mô tả các thuộc tính của *FFlowLayoutPanel*

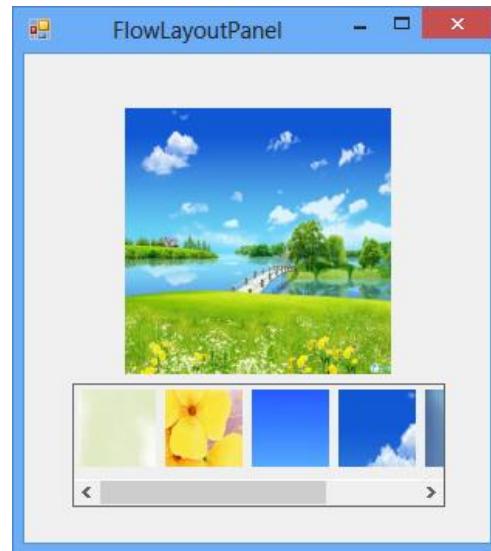
Thuộc tính	Mô tả
<i>AutoScroll</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: <i>FFlowLayoutPanel</i> tự động xuất hiện thanh trượt khi kích thước Panel không thể hiển thị hết các điều khiển chứa bên trong</li> <li>- Nếu là False: <i>FFlowLayoutPanel</i> sẽ không hiển thị thanh trượt</li> </ul>
<i>BorderStyle</i>	Kiểu đường viền của <i>FFlowLayoutPanel</i> khi hiển thị. Có 3 giá trị: None, FixedSingle và Fixed3D <ul style="list-style-type: none"> <li>- None: Không hiển thị đường viền</li> <li>- FixedSingle: Quanh <i>FFlowLayoutPanel</i> sẽ hiển thị một đường viền đơn.</li> <li>- Fixed3D: Hiển thị đường viền của <i>FFlowLayoutPanel</i> dạng 3 chiều.</li> </ul>
<i>FlowDirection</i>	Cách thức bố trí các điều khiển khi các điều khiển nằm ngoài phạm vi của <i>FFlowLayoutPanel</i> . Bao gồm 4 giá trị: <i>LeftToRight</i> , <i>RightToLeft</i> , <i>TopDown</i> , <i>BottomUp</i>
<i>WrapContents</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Các điều khiển vượt ngoài kích thước <i>FFlowLayoutPanel</i> sẽ tự động bố trí trên một dòng mới hoặc một cột mới</li> <li>- Nếu là False: <i>FFlowLayoutPanel</i> sẽ xuất hiện thanh trượt để hiển thị các điều khiển ngoài kích thước của <i>FFlowLayoutPanel</i> (với thuộc tính <i>AutoScroll</i> là True). Nếu thuộc tính <i>AutoScroll</i> là False thì các điều khiển nằm ngoài kích thước sẽ bị ẩn đi.</li> </ul>

➤ Phương thức thường dùng của *FFlowLayoutPanel*:

Bảng 6.4: Bảng mô tả các phương thức của *FFlowLayoutPanel*

Phương thức	Mô tả
<i>Controls.Add</i>	Phương thức có chức năng thêm điều khiển vào <i>FlowLayoutPanel</i>

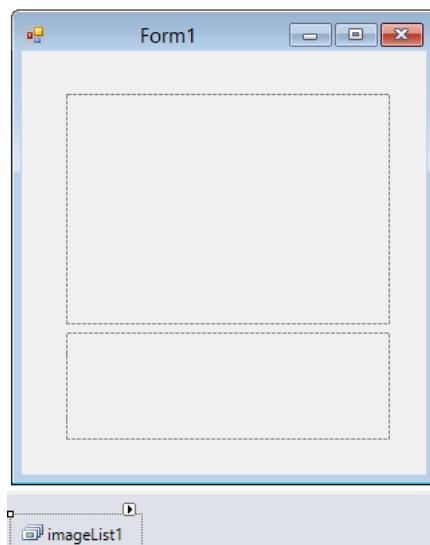
Ví dụ 6.2: Thiết kế giao diện chương trình gồm có 1 *PictureBox* và 1 *FlowLayoutPanel* như hình 6.10. Với *FlowLayoutPanel* chứa danh sách các hình. Khi người dùng nhập chuột chọn hình nào trong *FlowLayoutPanel* thì hình đó sẽ hiển thị trên *PictureBox*.



Hình 6.10: Giao diện form hiển thị hình ví dụ 6.2

Hướng dẫn:

Bước 1: Thiết kế giao diện chương trình. Kéo các điều khiển FlowLayoutPanel, PictureBox và ImageList và form như hình 6.11.



Hình 6.11: Giao diện form sau khi thêm PictureBox, FlowLayoutPanel và ImageList

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties

- Form1:
  - Thuộc tính Text: "FlowLayoutPanel"
- pictureBox1:
  - Thuộc tính Name: myPictureBox
- flowLayoutPanel1:
  - Thuộc tính Name: myFlowLayoutPanel
  - Thuộc tính BorderStyle: FixSingle
  - Thuộc tính AutoScroll: True

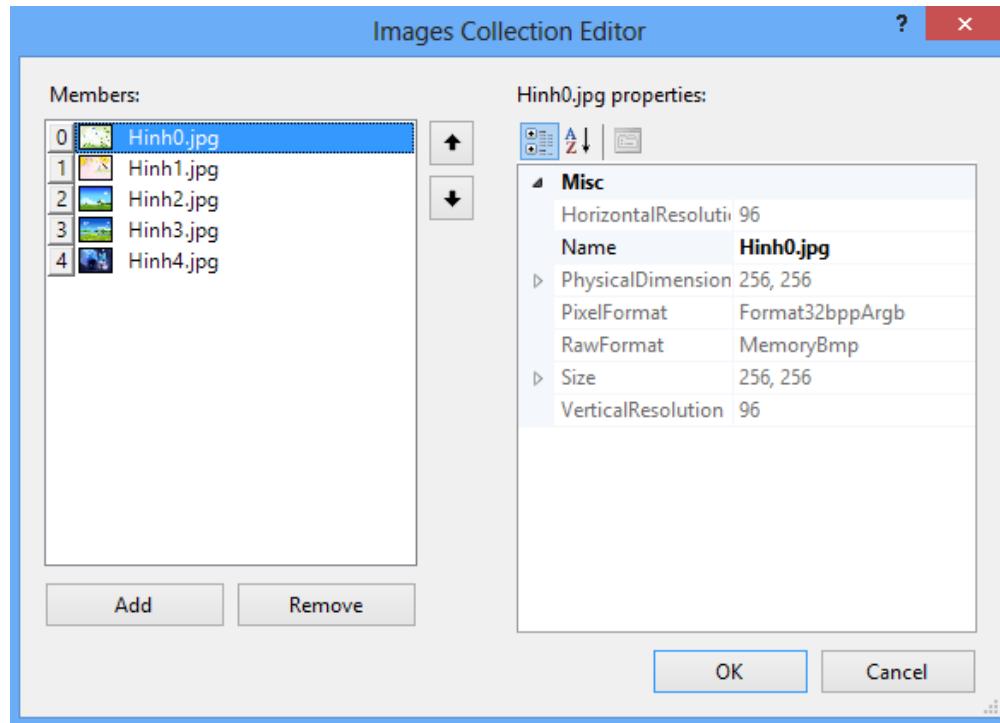
Thuộc tính WrapContents: False

- `imageList1`:

Thuộc tính Name: `myImageList`

Thuộc tính ImageSize: 256, 256

Thuộc tính Images: Thêm một số hình ảnh vào `myImageList` như hình 6.12



Hình 6.12: Cửa sổ thêm hình cho `myImageList`

Bước 3: Viết mã lệnh cho điều khiển

- Sự kiện `Load` của `Form1`:

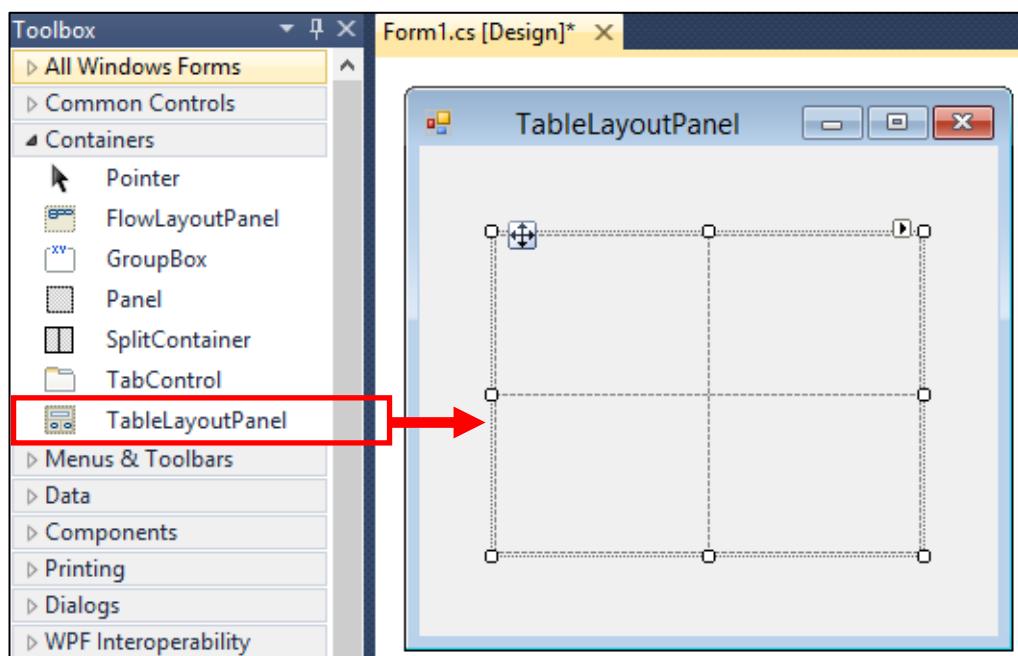
```
private void Form1_Load(object sender, EventArgs e)
{
    for (int i = 0; i < myImageList.Images.Count; i++)
    {
        PictureBox pic = new PictureBox();
        pic.Image = myImageList.Images[i];
        pic.Size = new Size(50, 50);
        pic.Click += new EventHandler(Form1_Click);
        myFlowLayoutPanel.Controls.Add(pic);
    }
}
```

- Sự kiện *Click* của Form1:

```
private void Form1_Click(object sender, EventArgs e)
{
    try
    {
        PictureBox pic = (PictureBox)sender;
        myPictureBox.Image = pic.Image;
    }catch (Exception ex) { }
}
```

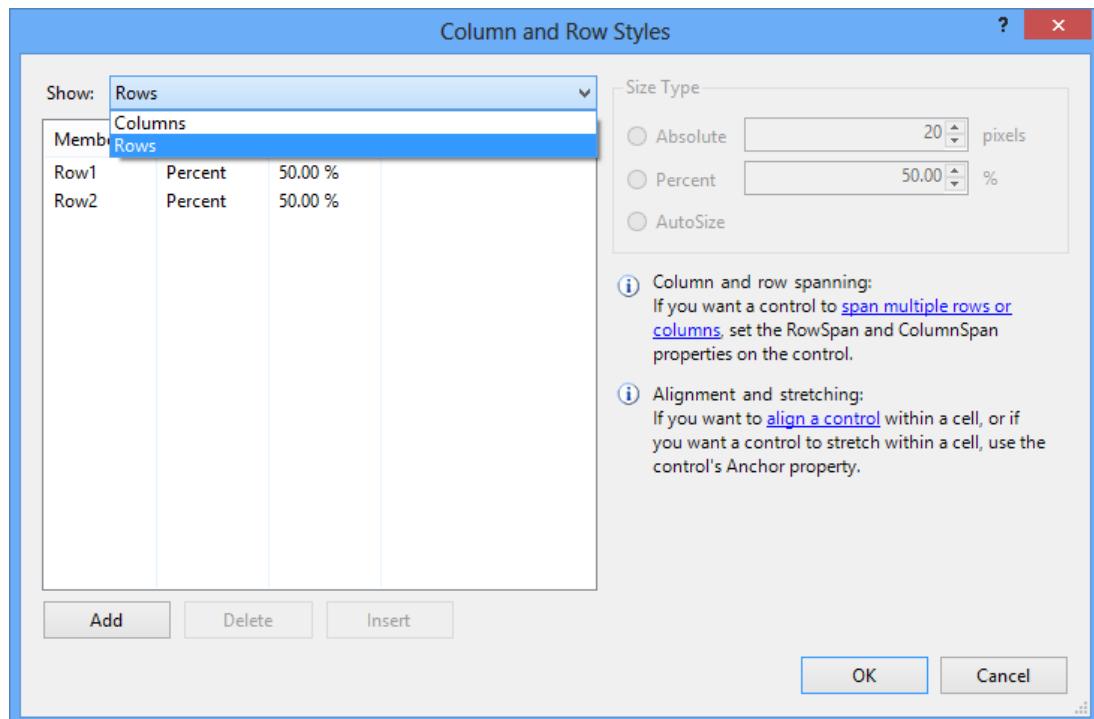
## 6.4. Điều khiển TableLayoutPanel

Cũng như điều khiển *FlowLayoutPanel*, *TableLayoutPanel* là điều khiển dẫn xuất từ điều khiển Panel và được dùng cho mục đích thiết kế giao diện form. *TableLayoutPanel* nằm trong nhóm Containers của cửa sổ Toolbox như hình 6.13.

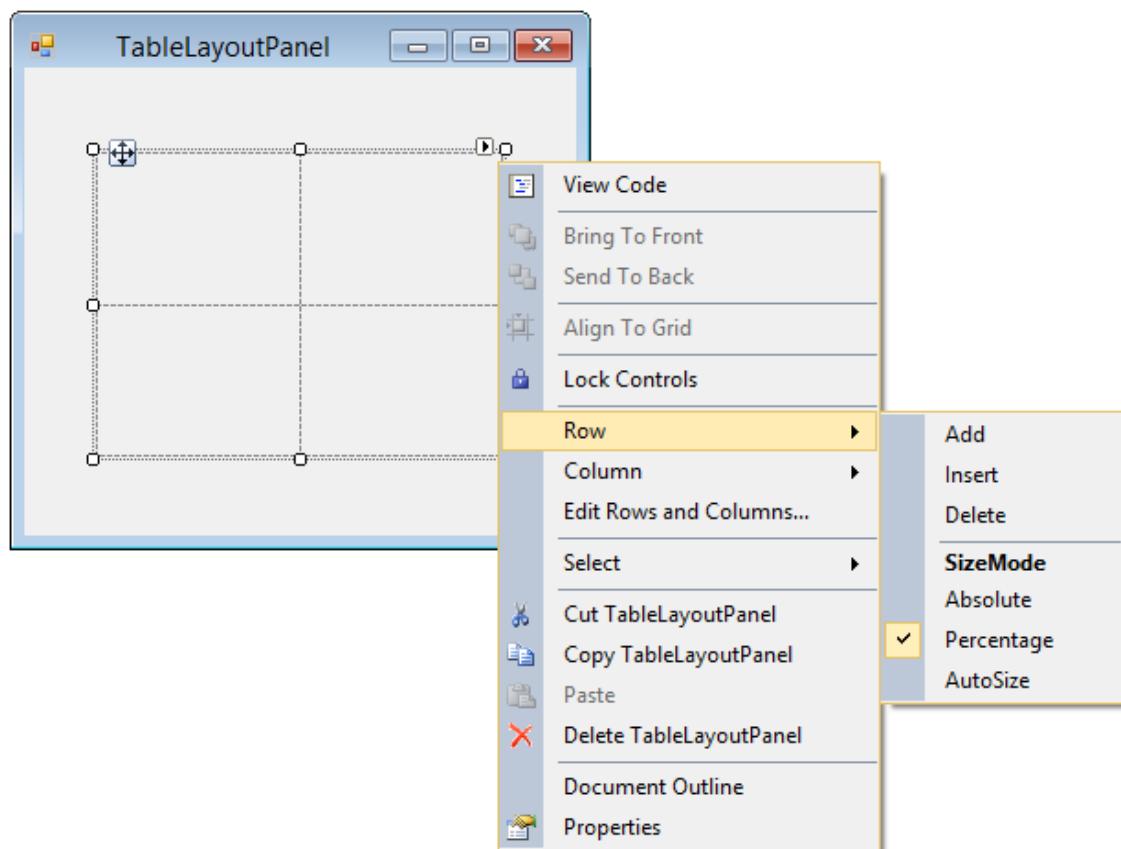


Hình 6.13: Điều khiển *TableLayoutPanel* trong cửa sổ Toolbox

*TableLayoutPanel* bao gồm các ô theo dòng và cột để thêm điều khiển vào. Lập trình viên có thể thêm các dòng và các cột cho *TableLayoutPanel* qua các thuộc tính *Columns* và *Rows* trong cửa sổ Properties như hình 6.14, hoặc thêm dòng và cột qua *ContextMenuStrip* khi nhấp chuột phải vào *TableLayoutPanel* như hình 6.15.



Hình 6.14: Thêm dòng và cột trên cửa sổ Column and Row Styles



Hình 6.15: Thêm dòng hoặc cột trên ContextMenuStrip

Trên cửa sổ Column and Row Styles như hình 6.14, các dòng và cột thêm vào có thể xác định kích thước bằng pixel (mục chọn *Absolute*), phần trăm (mục chọn *Percent*) hoặc tự động điều chỉnh kích thước (mục chọn *AutoSize*).

Với mỗi ô của *TableLayoutPanel*, chỉ có thể chứa được một điều khiển. Tuy nhiên lập trình viên có thể thêm nhiều điều khiển trong một ô bằng cách thêm một điều khiển loại Containers như: *GroupBox*, *Panel*, ... vào ô của *TableLayoutPanel*, khi đó lập trình viên có thể thêm nhiều điều khiển vào điều khiển loại Containers nằm trong ô của *TableLayoutPanel*.

*TableLayoutPanel* cũng được cung cấp thanh trượt (*ScrollBar*) khi thuộc tính *AutoScroll* là True.

*TableLayoutPanel* là điều khiển dạng bảng chia thành các ô (cell) do đó không có thuộc tính *BorderStyle* mà thay vào đó có thuộc tính *CellBorderStyle*. Thuộc tính *CellBorderStyle* chỉ định kiểu đường viền cho *TableLayoutPanel*. Thông thường, thuộc tính *TableLayoutStyle* có giá trị mặc định là *None*, nghĩa là không hiển thị đường viền quanh các ô của *TableLayoutPanel*. Lập trình có thể thiết lập các loại đường viền được hỗ trợ cho thuộc tính *CellBorderStyle* như: *Single*, *Inset*, *InsetDouble*, *Outset*, *OutsetDouble*, hoặc *OutsetPartial*.

- Một số thuộc tính thường dùng của *TableLayoutStyle*:

Bảng 6.5: Bảng mô tả các thuộc tính của *TableLayoutStyle*

Thuộc tính	Mô tả
<i>AutoScroll</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: <i>TableLayoutStyle</i> tự động xuất hiện thanh trượt khi kích thước Panel không thể hiển thị hết các điều khiển chứa bên trong</li> <li>- Nếu là False: <i>TableLayoutStyle</i> sẽ không hiển thị thanh trượt</li> </ul>
<i>CellBorderStyle</i>	Kiểu đường viền của <i>TableLayoutStyle</i> khi hiển thị. Có 3 giá trị: <i>None</i> , <i>Single</i> và <i>Inset</i> , <i>InsetDouble</i> , <i>Outset</i> , <i>OutsetDouble</i> , <i>OutsetPartial</i> .
<i>ColumnCount</i>	Số cột của <i>TableLayoutPanel</i> . Lập trình viên có thể thêm hoặc giảm số cột bằng cách thay đổi giá trị thuộc tính <i>ColumnCount</i>
<i>Columns</i>	Thuộc tính này giúp hiển thị bảng Column and Row Styles để thêm, sửa hoặc xóa cột.
<i>GrowStyle</i>	Thuộc tính chỉ định việc thay đổi kích thước của <i>TableLayoutPanel</i> như thế nào. Gồm các giá trị: <ul style="list-style-type: none"> <li>- <i>FixedSize</i>: Cố định kích thước, không</li> </ul>

	thay đổi. - AddRows: Thêm dòng mới - AddColumns: Thêm cột mới
RowCount	Số cột của <i>TableLayoutPanel</i> . Lập trình viên có thể thêm hoặc giảm số dòng bằng cách thay đổi giá trị thuộc tính RowCount
Rows	Thuộc tính này giúp hiển thị bảng Column and Row Styles để thêm, sửa hoặc xóa dòng.

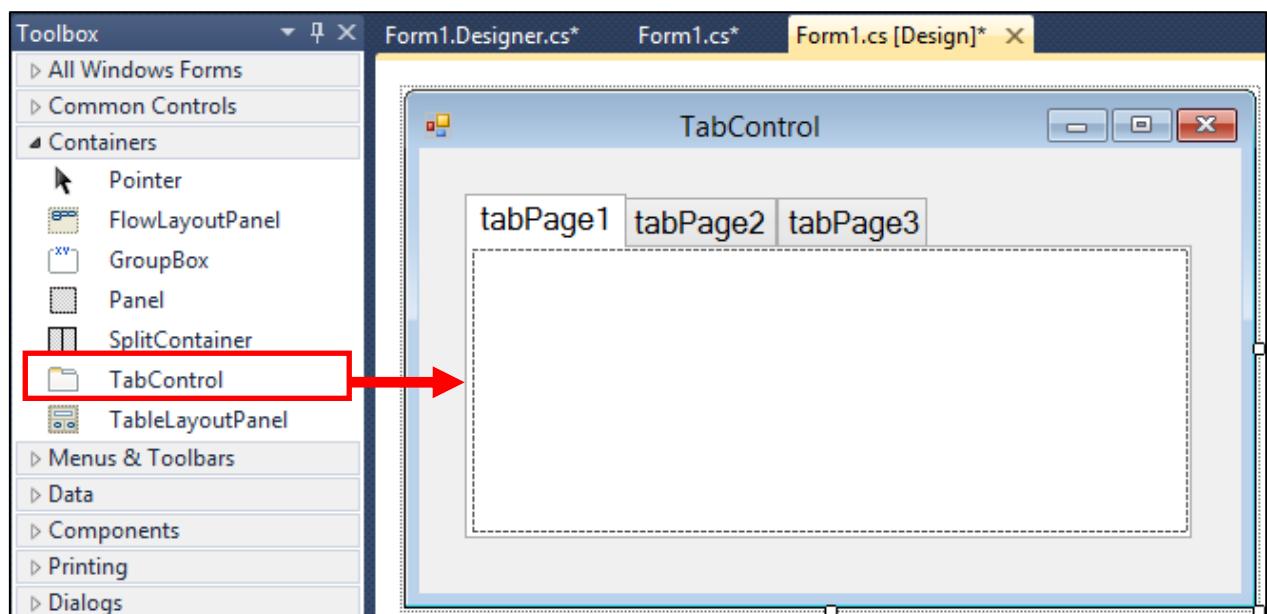
➤ Phương thức thường dùng của *TableLayoutStyle*:

Bảng 6.6: Bảng mô tả các phương thức của *TableLayoutStyle*

Phương thức	Mô tả
Controls.Add	Phương thức có chức năng thêm điều khiển vào <i>TableLayoutStyle</i>

## 6.5. Điều khiển TabControl

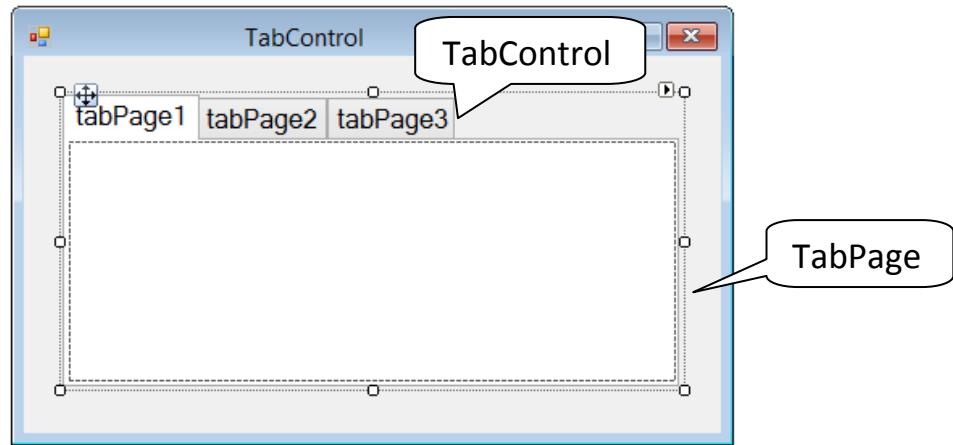
TabControl là điều khiển dạng Containers, do đó có thể chứa các điều khiển khác. Điểm đặc biệt của TabControl là cho phép thể hiện nhiều page trên một form duy nhất. Mỗi page có thể chứa nhiều điều khiển khác bên trong. Điều khiển TabControl nằm trong nhóm Containers của cửa sổ Toolbox như hình 6.16.



Hình 6.16: Điều khiển TabControl trong cửa sổ Toolbox

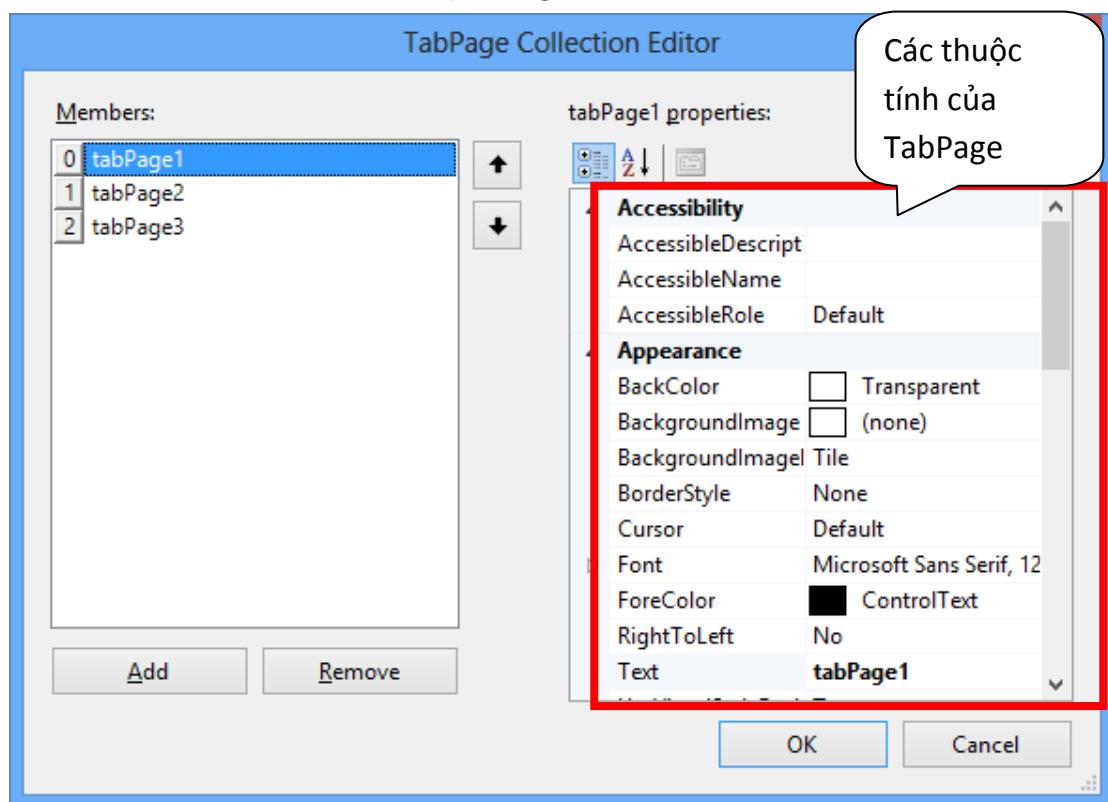
➤ TabPage:

Thuộc tính quan trọng nhất của TabControl là TabPage. Một TabControl có thể có nhiều TabPage như hình 6.17. Người dùng có thể nhấp vào các tab để chuyển đổi qua lại giữa các TabPage với nhau



Hình 6.17: TabControl chứa nhiều TabPage

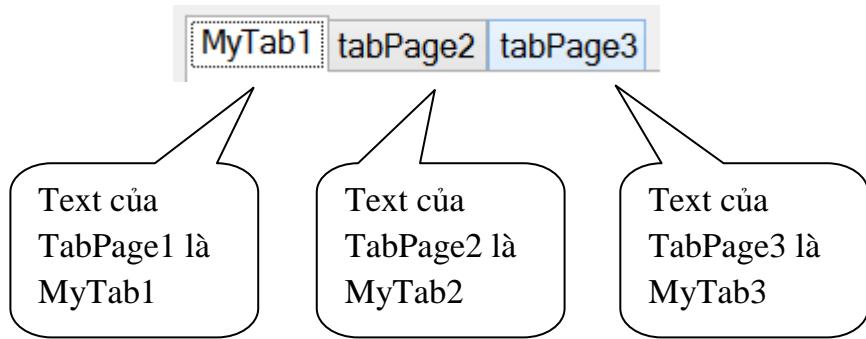
*TabPage* là điều khiển dạng Container nằm trong *TabControl* và có thể chứa các điều khiển khác bên trong. Mỗi *TabPage* có các thuộc tính riêng, lập trình viên có thể thiết lập giá trị thuộc tính khác nhau trên mỗi *TabPage* của *TabControl* bằng cách nhập chuột trái chọn thuộc tính *TabPage*s trên cửa sổ Properties. Khi đó một cửa sổ *TabPage Collection Editor* sẽ hiển thị như hình 6.18. Tại cửa sổ này, lập trình viên cũng có thể thêm hoặc xóa các *TabPage* bằng cách nhấn nút Add hoặc Remove.



Hình 6.18: Cửa sổ thiết lập giá trị thuộc tính cho TabPage

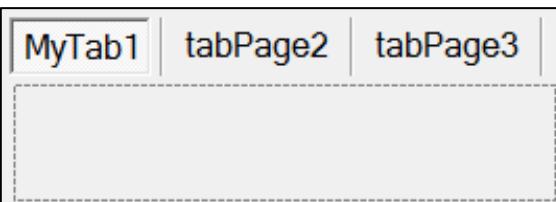
Điều khiển *TabPage* có nhiều điểm giống với điều khiển *Panel*. *TabPage* cũng hỗ trợ thanh trượt khi cần như thuộc tính *AutoScroll* được thiết lập là *True*, có thuộc tính *BorderStyle* để thiết lập đường viền quanh *TabPage* với 3 giá trị: *None*, *FixedSingle*, *Fixed3D*. Tuy nhiên có điểm khác biệt với *Panel* là *TabPage* hỗ trợ thuộc

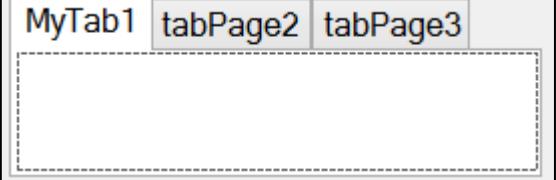
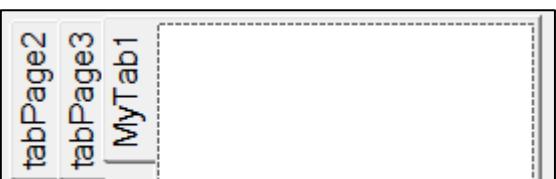
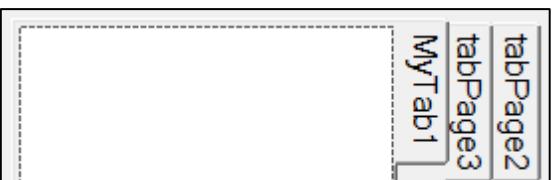
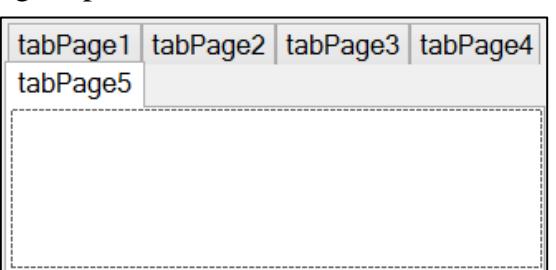
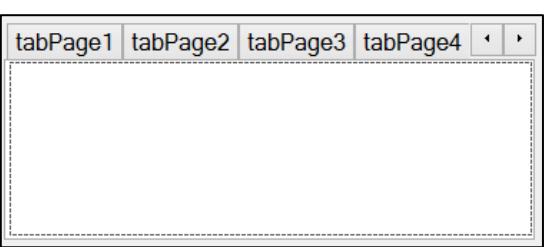
tính *Text*, chuỗi mô tả được thiết lập trong thuộc tính *Text* sẽ hiển thị trên tab của *TabPage*:



- Các thuộc tính thường dùng của *TabControl*:

*Bảng 6.7: Bảng mô tả các thuộc tính của TabControl*

Thuộc tính	Mô tả
<i>Appearance</i>	<p>Thuộc tính chỉ định <i>TabPage</i> sẽ hiển thị ở hình dạng nào. Có 3 giá trị:</p> <ul style="list-style-type: none"> <li>- <i>Normal</i>:</li>  </ul> <ul style="list-style-type: none"> <li>- <i>Button</i>:</li>  </ul> <ul style="list-style-type: none"> <li>- <i>FlatButtons</i>:</li>  </ul>
<i>Alignment</i>	<p>Thuộc tính xác định các tab sẽ hiển thị ở trên, dưới, trái hay phải của <i>TabControl</i>. Gồm các giá trị:</p> <ul style="list-style-type: none"> <li>- <i>Top</i>:</li> </ul>

	 <ul style="list-style-type: none"> <li>- <i>Bottom:</i></li> </ul>  <ul style="list-style-type: none"> <li>- <i>Left:</i></li> </ul>  <ul style="list-style-type: none"> <li>- <i>Right:</i></li> </ul> 
<i>Multiline</i>	<p>Mang hai giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép hiển thị nhiều dòng để chứa các tab nếu số lượng các tab vượt ngoài phạm vi kích thước của <i>TabControl</i>.</li> </ul>  <ul style="list-style-type: none"> <li>- Nếu là False: Chỉ cho phép tab hiển thị trên một dòng.</li> </ul> 

<i>TabPage</i>	Chứa tập các các <i>TabPage</i> có trong <i>TabControl</i>
<i>TabCount</i>	Trả về số lượng <i>TabPage</i> mà <i>TabControl</i> có
<i>SelectedTab</i>	Trả về điều khiển <i>TabPage</i> được chọn
<i>SelectedIndex</i>	Trả về vị trí của <i>TabPage</i> được chọn

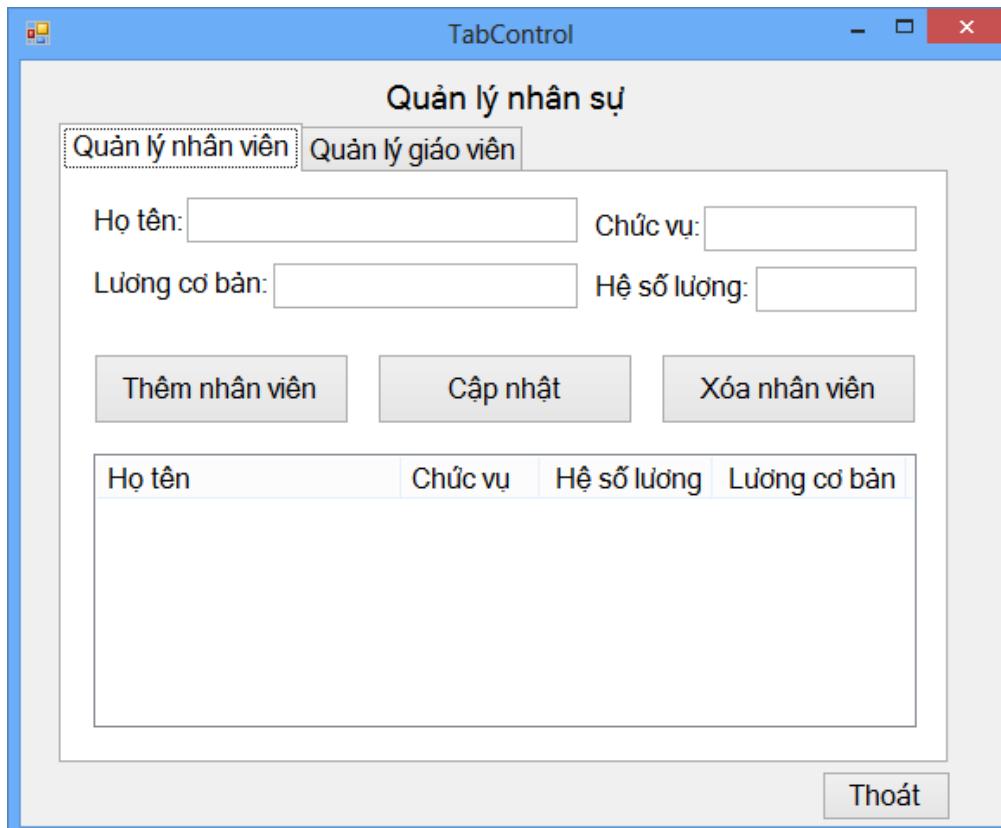
- Các sự kiện thường dùng của *TabControl*:

Bảng 6.8: Bảng mô tả các sự kiện của *TabControl*

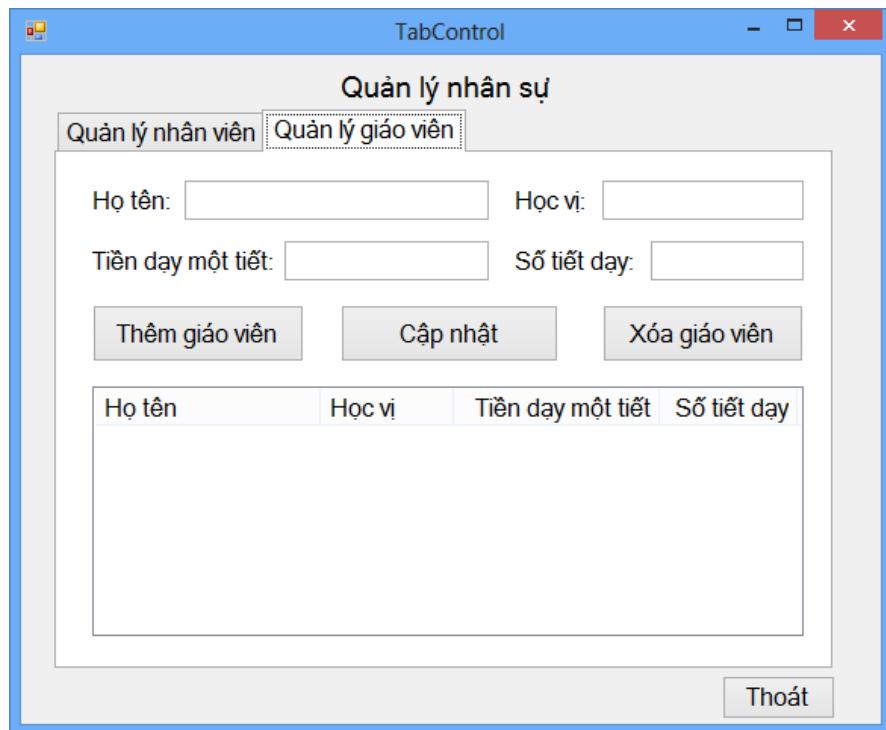
Sự kiện	Mô tả
<i>SelectedIndexChanged</i>	Phát sinh khi người dùng chọn một <i>TabPage</i> khác trên <i>TabControl</i>

Ví dụ 6.3: Viết chương trình quản lý nhân sự như hình 6.19 và 6.20. Chương trình gồm 2 *TabPage*: *TabPage* Quản lý nhân viên và *TabPage* Quản lý giáo viên.

- *TabPage* quản lý nhân viên: Cho phép thêm, sửa và xóa nhân viên. Thông tin nhân viên cần quản lý bao gồm: Họ tên nhân viên, chức vụ của nhân viên, hệ số lương và lương cơ bản.
- *TabPage* Quản lý giáo viên: Cho phép thêm sửa xóa giáo viên. Thông tin giáo viên cần quản lý gồm: Họ tên giáo viên, chức vụ của giáo viên, tiền giảng một tiết, số tiết dạy và học vị của giáo viên.



Hình 6.19: Giao diện *TabPage* Quản lý nhân viên

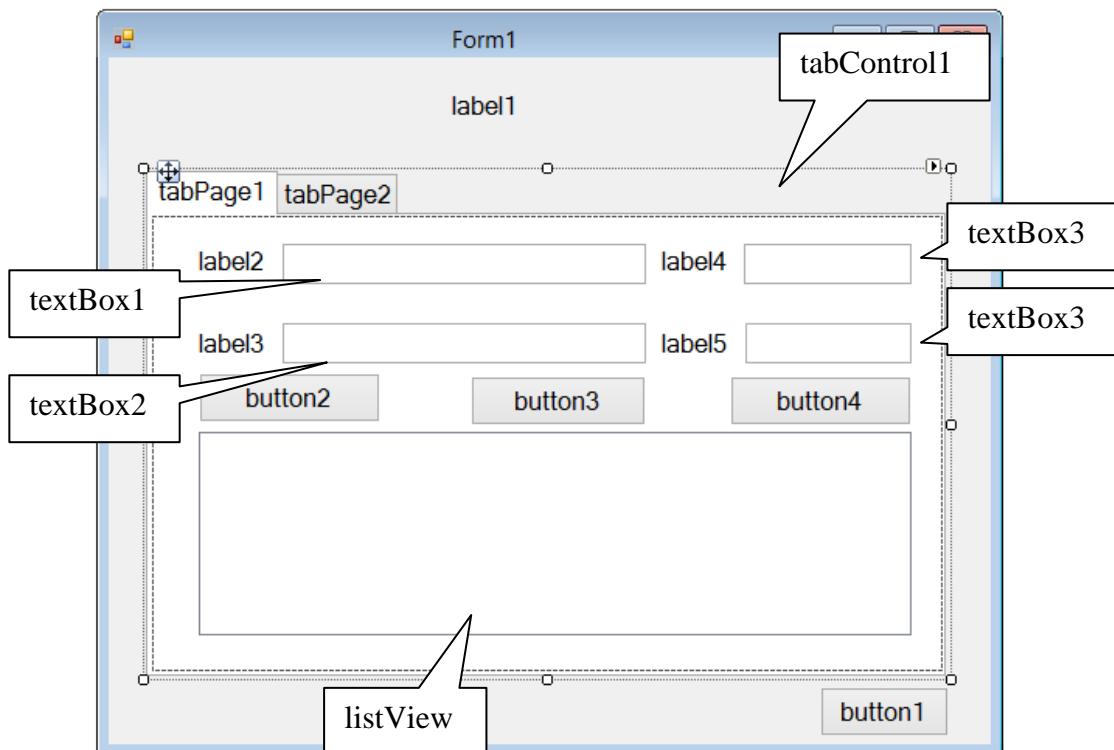


Hình 6.20: Giao diện TagPage Quản lý giáo viên

Hướng dẫn:

Tạo TagPage Quản lý nhân viên:

Bước 1: Thiết kế giao diện ban đầu. Thêm các điều khiển *Label*, *TextBox*, *TabControl* và *ListView* vào form như hình 6.21.



Hình 6.21: Giao diện TagPage Quản lý nhân viên sau khi thêm điều khiển

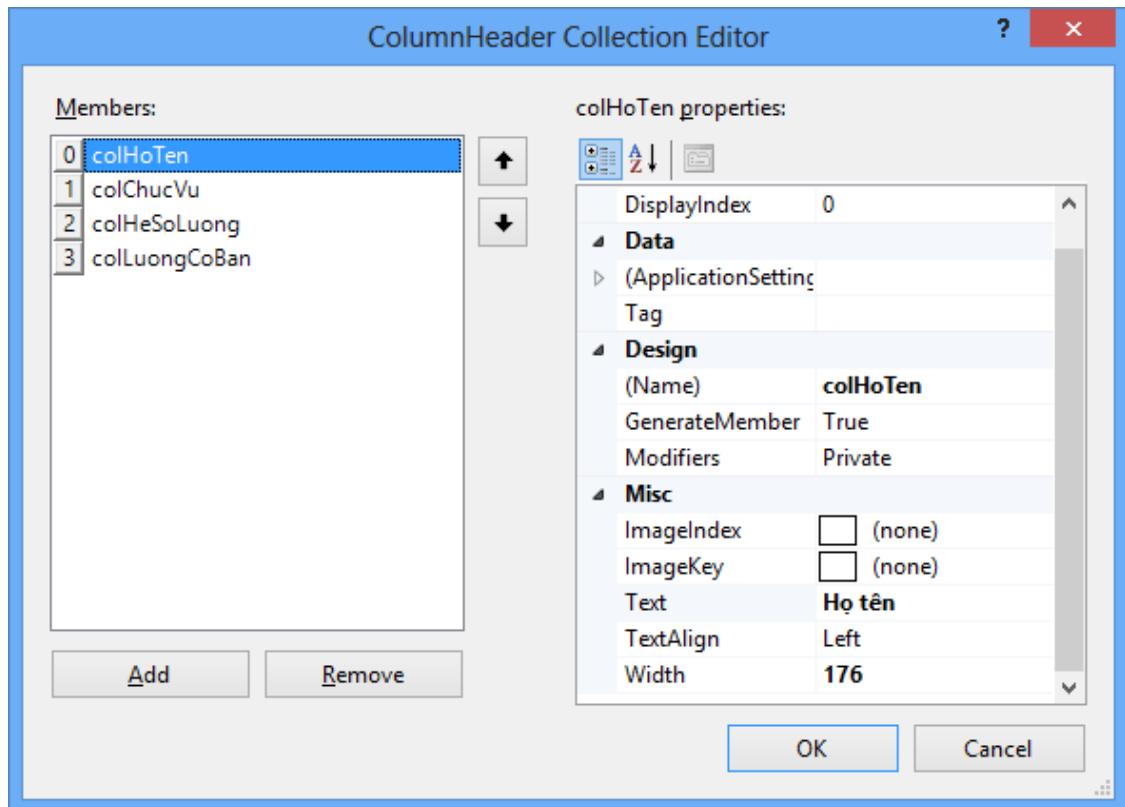
Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties

- Form1:

- Thuộc tính *Text*: “TabControl”
- label1:  
Thuộc tính *Text*: “Quản lý nhân sự”  
Thuộc tính *Size*: 14
  - label2:  
Thuộc tính *Text*: “Họ tên:”
  - label3:  
Thuộc tính *Text*: “Lương cơ bản:”
  - label4:  
Thuộc tính *Text*: “Chức vụ:”
  - label5:  
Thuộc tính *Text*: “Hệ số lương:”
  - textBox1:  
Thuộc tính *Name*: txtHoTenNV
  - textBox2:  
Thuộc tính *Name*: txtLuongCBNV
  - textBox3:  
Thuộc tính *Name*: txtChucVuNV
  - textBox4:  
Thuộc tính *Name*: txtHeSoLuongNV
  - button1:  
Thuộc tính *Name*: btnThoat  
Thuộc tính *Text*: “Thoát”
  - button2:  
Thuộc tính *Name*: btnThemNV  
Thuộc tính *Text*: “Thêm nhân viên”
  - button3:  
Thuộc tính *Name*: btnCapNhatNV  
Thuộc tính *Text*: “Cập nhật”
  - button4:  
Thuộc tính *Name*: btnXoaNV  
Thuộc tính *Text*: “Xóa nhân viên”
  - listView1:  
Thuộc tính *Name*: listNhanVien  
Thuộc tính *View*: Detail  
Thuộc tính *FullRowSelect*: True  
Thuộc tính *MultiSelect*: False

Thuộc tính *Columns*: Mở cửa sổ ColumnHeader Collection Editor, thêm 4 cột: colHoTen, colChucVu, colHeSoLuong, colLuongCoBan như hình 6.22.

- Cột colHoTen: Thiết lập thuộc tính *Text* là “Họ tên”
- Cột colChucVu: Thiết lập thuộc tính *Text* là “Chức vụ”
- Cột colHeSoLuong: Thiết lập thuộc tính *Text* là “Hệ số lương”
- Cột colLuongCoBan: Thiết lập thuộc tính *Text* là “Lương cơ bản”



Hình 6.22: Thêm cột cho listView1 trong cửa sổ ColumnHeader Collection Editor

- tabControl1:

Thuộc tính *Name*: myTabControl

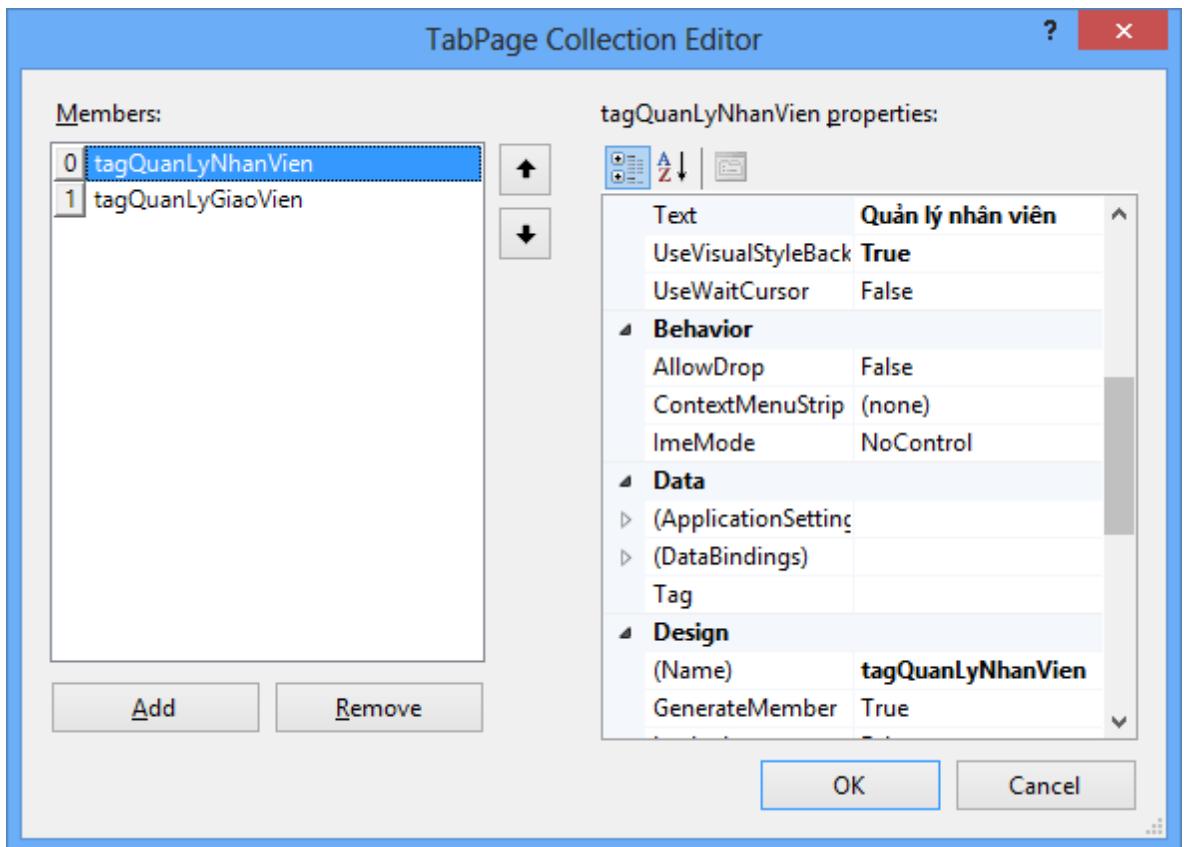
Thuộc tính *TabPage*: Mở cửa sổ TabPage Collection Editor, thêm 2 *TabPage*: Quản lý nhân viên và Quản lý giáo viên như hình 6.23.

Trên tabQuanLyNhanVien:

- Thiết lập thuộc tính *Text*: “Quản lý nhân viên”

Trên tabQuanLyGiaoVien:

- Thiết lập thuộc tính *Text*: “Quản lý giáo viên”



Hình 6.23: ThêmTabPage cho TabControl trên cửa sổ tabPage Collection Editor

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện Click của btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện Click của btnCapNhatNV:

```
private void btnCapNhatNV_Click(object sender, EventArgs e)
{
    foreach (ListViewItem lvi in listNhanVien.SelectedItems)
    {
        lvi.SubItems[0].Text = txtTenNV.Text;
        lvi.SubItems[1].Text = txtChucVuNV.Text;
        lvi.SubItems[2].Text = txtHeSoLuongNV.Text;
        lvi.SubItems[3].Text = txtLuongCBNV.Text;
    }
}
```

- Sự kiện *Click* của btnThemNV:

```
private void btnThemNV_Click(object sender, EventArgs e)
{
    if (txtLuongCBNV.Text != "" && txtTenNV.Text != "" &&
        txtHeSoLuongNV.Text != "" && txtChucVuNV.Text != "")
    {
        ListViewItem LVIItem = new ListViewItem(txtTenNV.Text);
        ListViewItem.ListViewSubItem LVSItemCV = new
            ListViewItem.ListViewSubItem(LVIItem,
            txtChucVuNV.Text);
        ListViewItem.ListViewSubItem LVSItemHSL = new
            ListViewItem.ListViewSubItem(LVIItem,
            txtHeSoLuongNV.Text);
        ListViewItem.ListViewSubItem LVSItemLCB = new
            ListViewItem.ListViewSubItem(LVIItem,
            txtLuongCBNV.Text);
        LVIItem.SubItems.Add(LVSItemCV);
        LVIItem.SubItems.Add(LVSItemHSL);
        LVIItem.SubItems.Add(LVSItemLCB);
        listNhanVien.Items.Add(LVIItem);
        txtLuongCBNV.Text = "";
        txtTenNV.Text = "";
        txtHeSoLuongNV.Text = "";
        txtChucVuNV.Text = "";
    }
    else
        MessageBox.Show("Phải nhập đầy đủ thông tin nhân viên");
}
```

- Sự kiện *Click* của btnXoaNV:

```
private void btnXoaNV_Click(object sender, EventArgs e)
{
    if (listNhanVien.SelectedIndices.Count > 0)
    {
        listNhanVien.Items.RemoveAt(listNhanVien.FocusedItem.Index);
    }
    else
        MessageBox.Show("Phải chọn nhân viên muốn ");
}
```

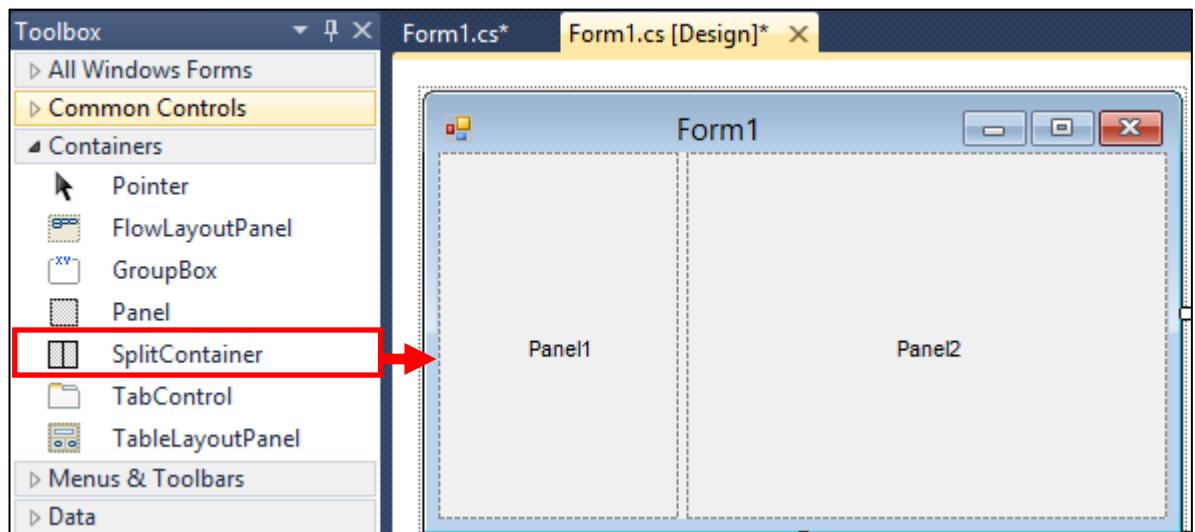
- Sự kiện *SelectedIndexChanged* của listNhanVien:

```
private void listNhanVien_SelectedIndexChanged(object sender,
EventArgs e)
{
    foreach (ListViewItem lvi in listNhanVien.SelectedItems)
    {
        txtTenNV.Text = lvi.SubItems[0].Text;
        txtChucVuNV.Text = lvi.SubItems[1].Text;
        txtHeSoLuongNV.Text = lvi.SubItems[2].Text;
        txtLuongCBNV.Text = lvi.SubItems[3].Text;
    }
}
```

Tạo TagPage Quản lý giáo viên: Thiết kế và viết mã lệnh tương tự như TagPage Quản lý nhân viên.

## 6.6. Điều khiển SplitContainer

Điều khiển *SplitContainer* giúp phân chia form thành hai phần. Cụ thể hơn *SplitContainer* được cấu tạo bởi hai điều khiển *Panel*, mỗi *Panel* trong điều khiển *SplitContainer* đều có chức năng như một điều khiển *Panel* thông thường. Khi thêm điều khiển *SplitContainer* từ cửa sổ Toolbox vào form thì mặc định *SplitContainer* có thuộc tính *Dock* mang giá trị *Fill*. Kích thước của *hai Panel* trong *SplitContainer* có thể thay đổi nhờ *Splitter*, *Splitter* là một vạch phân cách hai *Panel*. Điều khiển *SplitContainer* nằm trong nhóm *Containers* của cửa sổ Toolbox như hình 6.24.



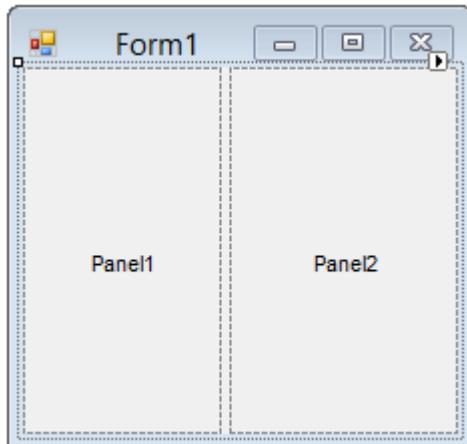
Hình 6.24: Điều khiển *SplitContainer* trong cửa sổ Toolbox

Các *Panel* đều hỗ trợ thanh trượt (ScrollBar) khi thuộc tính *AutoScroll* được thiết lập là True.

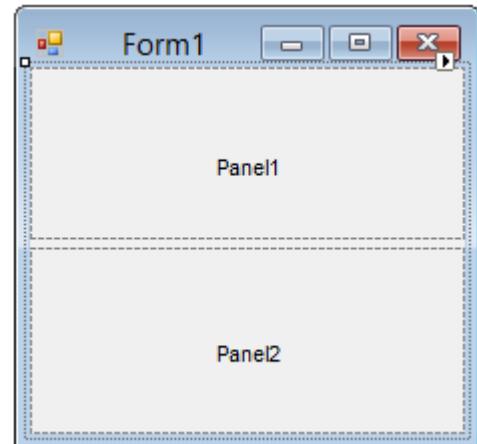
Tuy nhiên *Panel* trong *SplitContainer* không có thuộc tính *BorderStyle* để thiết lập đường viền, vì là điều khiển chứa trong *SplitContainer*, do đó thuộc tính *BorderStyle* được đặt ở điều khiển *SplitContainer*. Các giá trị của thuộc tính

*BorderStyle* của *SplitContainer* cũng gồm 3 giá trị như *Panel*: *None*, *Fixed3D*, *FixedSingle*.

Vạch phân cách *Splitter* có thể phân cách theo chiều dọc hoặc chiều ngang tùy thuộc vào thuộc tính *Orientation*. Thuộc tính *Orientation* mang hai giá trị để thiết lập *SplitContainer* là: *Vertical* và *Horizontal* như hình 6.25, hình 6.26.



Hình 6.25: Thuộc tính *Orientation* là  
*Vertical*



Hình 6.26: Thuộc tính *Orientation* là  
*Horizontal*

Nếu không muốn cho người dùng dịch chuyển vạch phân cách *Splitter* để thay đổi kích thước của hai *Panel*, lập trình viên có thể thiết lập thuộc tính *IsSplitterFixed* của *SplitContainer* là *True*. Ngoài ra một điểm đặc biệt là có thể chỉ định không cho phép thay đổi kích thước của Panel1 hoặc Panel2 bằng cách kết hợp thuộc tính *FixedPanel* và thuộc tính *IsSplitterFixed* như bảng 6.9.

Bảng 6.9: Bảng mô tả thuộc tính *IsSplitterFixed* và *FixedPanel*

Thuộc tính	Tác dụng
<i>IsSplitterFixed</i> = <i>False</i>	Thuộc tính <i>FixedPanel</i> không có hiệu lực. Người dùng có thể thay đổi kích thước của Panel1 và cả Panel2
<i>IsSplitterFixed</i> = <i>True</i>	<p>Thiết lập thuộc tính <i>FixedPanel</i>:</p> <ul style="list-style-type: none"> <li>- <i>FixedPanel</i> = <i>None</i>: Người dùng không thể sử dụng vạch phân cách <i>Splitter</i> để thay đổi kích thước của cả Panel1 và Panel2. Nhưng kích thước cả hai Panel sẽ thay đổi khi <i>SplitContainer</i> có thuộc tính <i>Dock</i> là <i>Fill</i> và người dùng thay đổi kích thước form.</li> <li>- <i>FixedPanel</i> = <i>Panel1</i>: Người dùng không thể thay đổi kích thước Panel1 (Khi thay đổi kích thước form thì kích thước Panel2 thay đổi, kích thước Panel1 là không đổi).</li> <li>- <i>FixedPanel</i> = <i>Panel2</i>: Người dùng không thể thay đổi kích thước Panel2 (Khi thay đổi kích thước form</li> </ul>

	thì kích thước Panel1 thay đổi, kích thước Panel2 là không đổi).
--	--

Một trong hai *Panel* của *SplitContainer* có thể ẩn đi bằng cách thiết lập thuộc tính *Panel1Collapsed* và *Panel2Collapsed* là True. Việc ẩn hai *Panel* chỉ có tác dụng với một *Panel*. Nghĩa là chỉ có thể thiết lập một trong hai thuộc tính *Panel1Collapsed* và *Panel2Collapsed* là True. Khi *Panel1Collapsed* là True thì *Panel2Collapsed* là False và ngược lại.

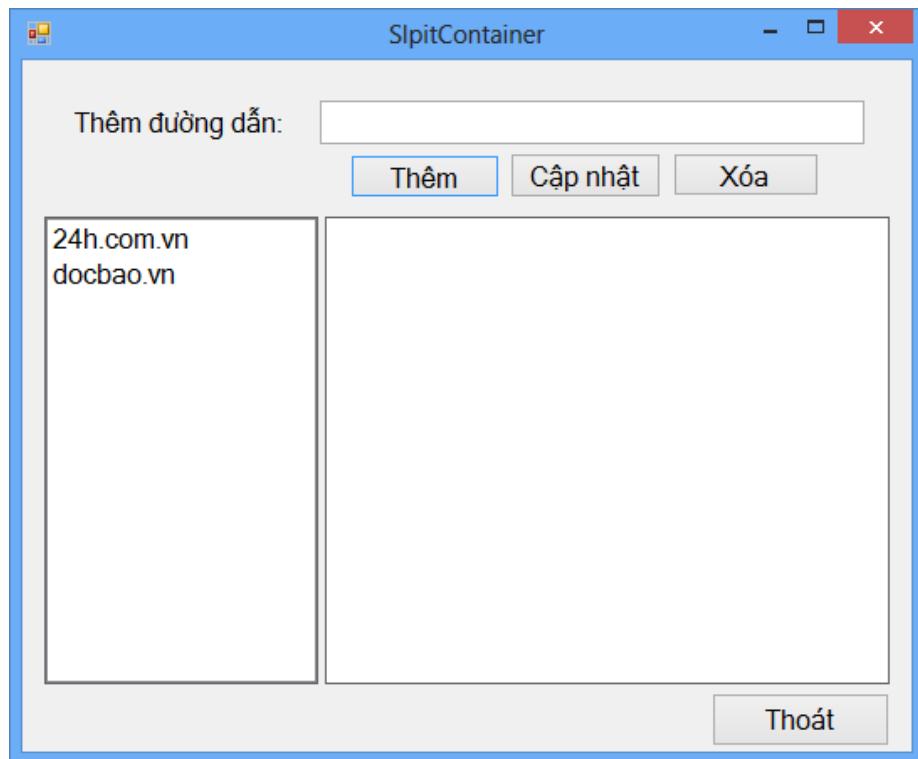
- Các thuộc tính thường dùng của *SplitContainer*:

Bảng 6.10: Bảng mô tả các thuộc tính của *SplitContainer*

Thuộc tính	Mô tả
<i>BorderStyle</i>	Thiết lập đường viền cho <i>SplitContainer</i> . Gồm 3 giá trị: <i>None</i> , <i>FixedSingle</i> , <i>Fixed3D</i> .
<i>FixedPanel</i>	Cố định kích thước của các <i>Panel</i> trong <i>SplitContainer</i> . Gồm 3 giá trị: <i>None</i> , <i>Panel1</i> , <i>Panel2</i> .
<i>IsSplitterFixed</i>	Mang hai giá trị True và False. Nếu là True, cố định vạch phân cách Splitter
<i>Orientation</i>	Xác định vạch phân cách Splitter sẽ phân cách theo chiều ngang hay dọc. Gồm 2 giá trị: <i>Vertical</i> , <i>Horizontal</i> .
<i>Panel1Collapsed</i>	Mang hai giá trị True hoặc False. Nếu là True sẽ ẩn Panel1
<i>Panel1MinSize</i>	Lấy kích thước nhỏ nhất hoặc thiết lập kích thước nhỏ nhất cho Panel1
<i>Panel2Collapsed</i>	Mang hai giá trị True hoặc False. Nếu là True sẽ ẩn Panel2
<i>Panel2MinSize</i>	Lấy kích thước nhỏ nhất hoặc thiết lập kích thước nhỏ nhất cho Panel2
<i>SplitterDistance</i>	Trả về khoảng cách bằng pixel từ Splitter đến cạnh bên trái (nếu hai Panel nằm dọc) hay đến cạnh trên (nếu hai Panel nằm ngang)
<i>SplitterWidth</i>	thiết lập độ rộng của vạch phân cách Splitter

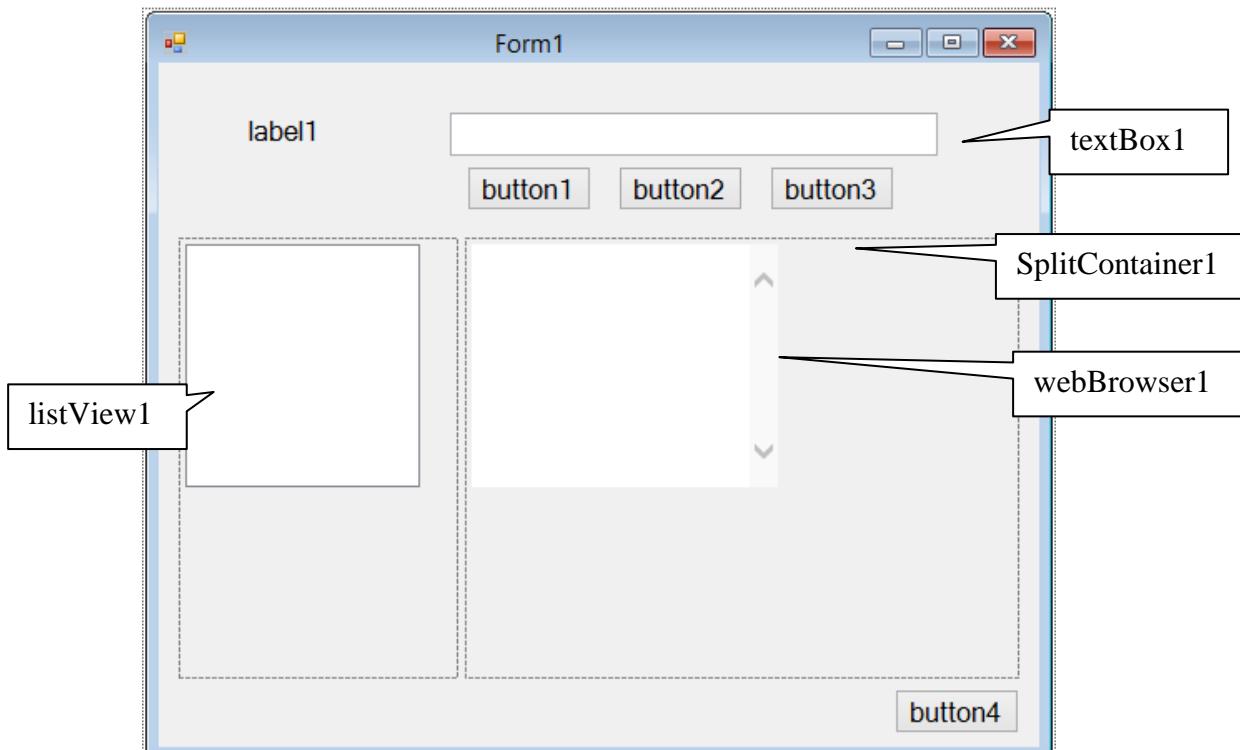
Ví dụ 6.4: Viết chương trình có giao diện như hình 6.27. Gồm: 1 điều khiển *SplitContainer* với Panel1 chứa 1 điều khiển *ListView* và Panel2 chứa 1 điều khiển *WebBrowser*.

Yêu cầu: Người dùng nhập đường dẫn website vào TextBox và nhấn nút thêm. Đường dẫn website vừa nhập sẽ được đưa vào ListView. Người dùng có thể hiển thị bất cứ website nào trên WebBrowser bằng cách nhấp chuột vào đường dẫn chứa trong ListView.



Hình 6.27: Giao diện quản lý địa chỉ website

Bước 1: Thiết kế giao diện ban đầu cho form. Thêm các điều khiển Label, TextBox, Button, SplitContainer và ListView, WebBrowser từ cửa sổ Toolbox vào form như hình 6.28.



### *Hình 6.28: Giao diện form sau khi thêm điều khiển*

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties

- Form1:  
Thuộc tính Text: “SplitContainer”
- label1:  
Thuộc tính Text: “Thêm đường dẫn:”  
textBox1:  
Thuộc tính Name: txtLink
- button1:  
Thuộc tính Name: btnThem  
Thuộc tính Text: “Thêm”
- button2:  
Thuộc tính Name: btnCapNhat  
Thuộc tính Text: “Cập nhật”
- button3:  
Thuộc tính Name: btnXoa  
Thuộc tính Text: “Xóa”
- button4:  
Thuộc tính Name: btnThoat  
Thuộc tính Text: “Thoát”
- listView1:  
Thuộc tính Name: listLinkWebsite  
Thuộc tính Dock: Fill
- webBrowser1:  
Thuộc tính Name: myWebsite  
Thuộc tính Dock: Fill
- splitContainer1:  
Thuộc tính BorderStyle: FixedSingle  
Thuộc tính Dock: None  
Thuộc tính Orientation: Vertical  
Thuộc tính IsSplitterFixed: True

Bước 3: Viết mã lệnh cho điều khiển

- Sự kiện Click của btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Load* của Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    myWebsite.Navigate("www.google.com");
}
```

- Sự kiện *Click* của nút btnThem:

```
private void btnThem_Click(object sender, EventArgs e)
{
    ListViewItem lvi = new ListViewItem(txtLink.Text);
    listLinkWebsite.Items.Add(lvi);
    txtLink.Text = "";
}
```

- Sự kiện *Click* của nút btnCapNhat:

```
private void btnCapNhat_Click(object sender, EventArgs e)
{
    listLinkWebsite.FocusedItem.Text = txtLink.Text;
}
```

- Sự kiện *Click* của nút btnXoa:

```
private void btnXoa_Click(object sender, EventArgs e)
{
    int i = listLinkWebsite.FocusedItem.Index;
    listLinkWebsite.Items.RemoveAt(i);
    txtLink.Text = "";
    myWebsite.Navigate("www.google.com");
}
```

- Sự kiện *MouseClick* của listLinkWebsite:

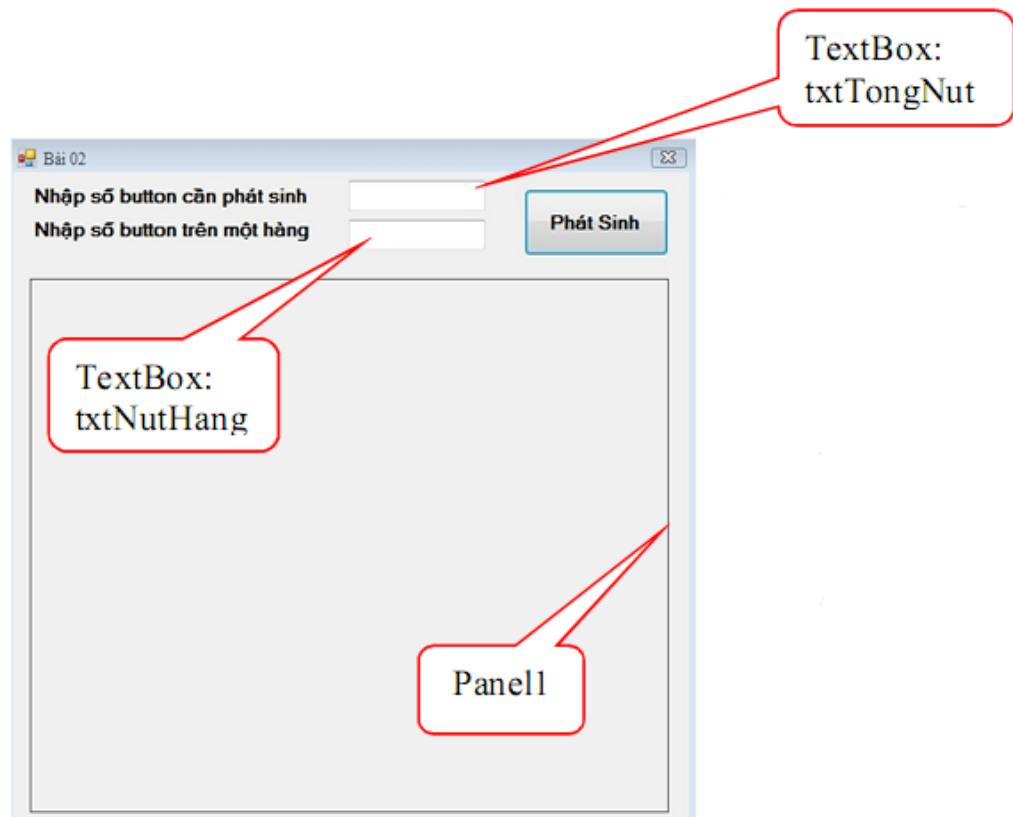
```
private void listLinkWebsite_MouseClick(object sender,
MouseEventArgs e)
{
    txtLink.Text = listLinkWebsite.FocusedItem.Text;
    if (e.Button == MouseButtons.Left)
        myWebsite.Navigate(listLinkWebsite.FocusedItem.Text);
}
```

## 6.7. Bài tập cuối chương

Câu 1: Thiết kế chương trình tạo Button có giao diện như hình 6.29.

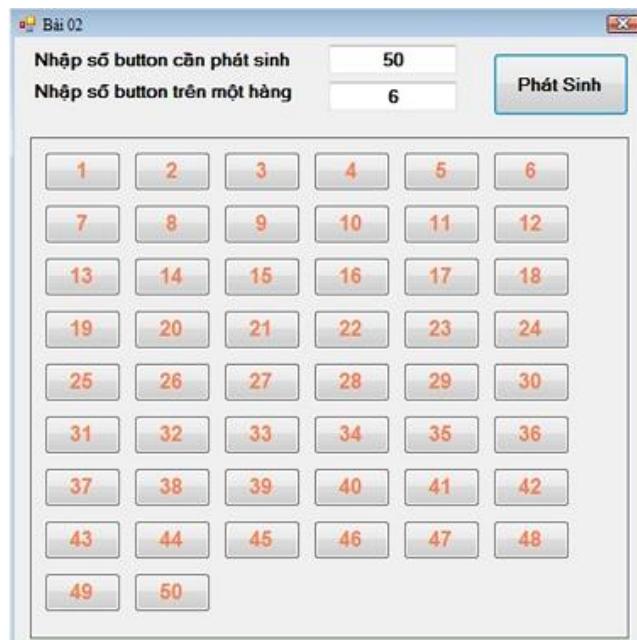
Yêu cầu:

- Khi nhấn F5 chạy chương trình xuất hiện giao diện như hình 629.
- Người dùng nhập tổng số Button cần phát sinh trong TextBox txtTongNut; nhập số Button muốn phát sinh trên một hàng trong TextBox txtNutHang.



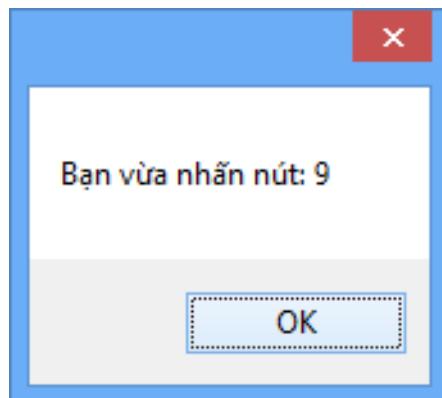
Hình 6.29: Giao diện chương trình tạo Button

- Khi người dùng nhấn nút Phát Sinh sẽ phát sinh các Button như yêu cầu vào Panel1 như hình 6.30.

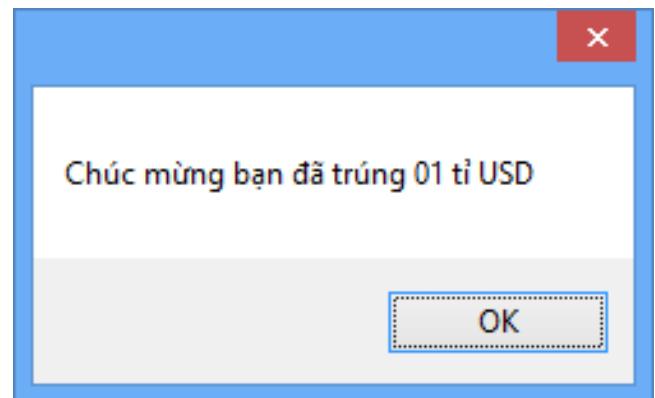


Hình 6.30: Giao diện chương trình sau khi phát sinh Button

- Khi người dùng nhấn vào một nút nhấn xuất thông báo “Bạn vừa nhấn nút: \_” như hình 6.31. Riêng nút số 10 khi người dùng nhấn sẽ xuất thông báo “Chúc mừng bạn đã trúng 01 tỉ USD” như hình 6.32.



Hình 6.31: MessageBox thông báo nút người dùng vừa nhấn



Hình 6.32: MessageBox thông báo người dùng vừa nhấn nút 10

Câu 2: Viết chương trình tạo câu hỏi trắc nghiệm như hình 6.33.

Hình 6.33: Giao diện tạo câu hỏi trắc nghiệm

Yêu cầu:

- Người dùng nhập nội dung câu hỏi trong textBox1
- Người dùng nhập đáp án trả lời trong textBox2. Nếu đáp án đúng thì checkBox1 sẽ được chọn tương ứng với giá trị 1. Nếu đáp án sai thì checkBox1 sẽ không được chọn tương ứng với giá trị 0.
- Người dùng nhấn Button “Thêm” để thêm đáp án vào listView1
- Nhấn Button “Thoát” để đóng chương trình.

- Nhấn Button “Hoàn thành” để hoàn tất việc tạo 1 câu hỏi trắc nghiệm. Khi đó sẽ hiển thị form như hình 6.34.



Hình 6.34: Giao diện trả lời trắc nghiệm

Trên giao diện trả lời trắc nghiệm như hình 6.34:

- Nội dung câu hỏi nhập trong textBox1 ở hình 6.33 sẽ hiển thị trên label1.
- Các đáp án trả lời trong listView1 ở hình 6.33 tương ứng sẽ tạo thành các radioButton đáp án để người dùng chọn trong flowLayoutPanel1.
- Nhấn Button “Trả lời” để hoàn thành việc chọn đáp án đúng, nếu chọn đúng thì MessageBox với nội dung “Bạn đã trả lời đúng” được hiển thị, nếu chọn sai thì MessageBox với nội dung “Bạn đã trả lời sai” được hiển thị.

# CHƯƠNG 7: ĐIỀU KHIỂN DIALOG VÀ PHƯƠNG THỨC MESSAGE

## 7.1. Lớp MessageBox

Lớp *MessageBox* giúp hiển thị một hộp thoại trên màn hình. Khi hộp thoại xuất hiện thì người dùng bắt buộc phải thao tác trên hộp thoại trước thì mới có tiếp tục thực hiện công việc trên ứng dụng. Việc hiển thị như thế hữu ích khi muốn cảnh báo một lỗi hoặc hướng dẫn cho người dùng. Sử dụng lớp *MessageBox* thì cần khai báo không gian tên:

```
System.Windows.Forms
```

Để hiển thị hộp thoại, người dùng chỉ cần sử dụng phương thức tĩnh *Show()* của lớp *MessageBox* mà không cần phải có bất cứ khai báo nào để tạo đối tượng lớp *MessageBox*.

- Phương thức *Show()*:

```
DialogResult Show(string text, string caption,  
                  MessageBoxButtons buttons,  
                  MessageBoxIcon icon,  
                  MessageBoxDefaultButton defaultButton,  
                  MessageBoxOptions options);
```

- *Text*: Chuỗi hiển thị trên hộp thoại *MessageBox*
- *Caption*: Chuỗi hiển thị trên thanh titlebar của hộp thoại *MessageBox*
- Kiểu liệt kê *MessageBoxIcon* gồm các giá trị xác định biểu tượng hiển thị trên *MessageBox*:

```
public enum MessageBoxIcon  
{  
    Asterisk = 0x40,  
    Error = 0x10,  
    Exclamation = 0x30,  
    Hand = 0x10,  
    Information = 0x40,  
    None = 0,  
    Question = 0x20,  
    Stop = 0x10,  
    Warning = 0x30  
}
```

- Kiểu liệt kê *MessageBoxDefaultButtons* gồm các giá trị xác định nút được *Focus* trên *MessageBox*:

```
public enum MessageBoxDefaultButton
{
    Button1 = 0,
    Button2 = 0x100,
    Button3 = 0x200
}
```

- Kiểu liệt kê *MessageBoxButtons* gồm các giá trị xác định nút hiển thị trên *MessageBox*:

```
public enum MessageBoxButtons
{
    OK,
    OKCancel,
    AbortRetryIgnore,
    YesNoCancel,
    YesNo,
    RetryCancel
}
```

- Kiểu liệt kê *MessageBoxOption* gồm các giá trị xác định nút hiển thị bên phải hoặc bên trái của *MessageBox*:

```
public enum MessageBoxOptions
{
    DefaultDesktopOnly = 0x20000,
    RightAlign = 0x80000,
    RtlReading = 0x100000,
    ServiceNotification = 0x200000
}
```

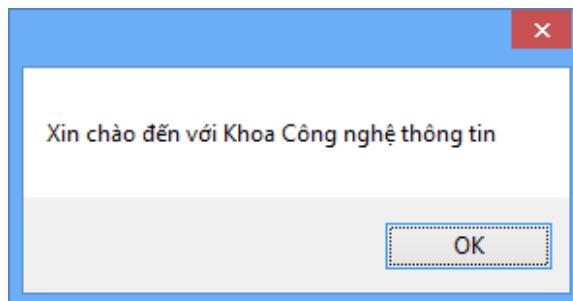
Lưu ý: Khi hộp thoại hiển thị từ phương thức *Show()* có sử dụng tham số *MessageBoxButton* thì phương thức này sẽ trả về một giá trị có kiểu *DialogResult*. *DialogResult* không phải là lớp mà là một kiểu liệt kê bao gồm các giá trị:

```
public enum DialogResult
{
    None,
    OK,
    Cancel,
    Abort,
    Retry,
    Ignore,
    Yes,
    No
}
```

Lập trình viên có thể sử dụng *DialogResult* để xác định xem người dùng đã nhấn nút nào trên hộp thoại *MessageBox*.

Ví dụ 7.1: Hiển thị các dạng khác nhau của hộp thoại *MessageBox*.

- Hiển thị hộp thoại hiển thị dòng thông báo đơn giản như hình 7.1:

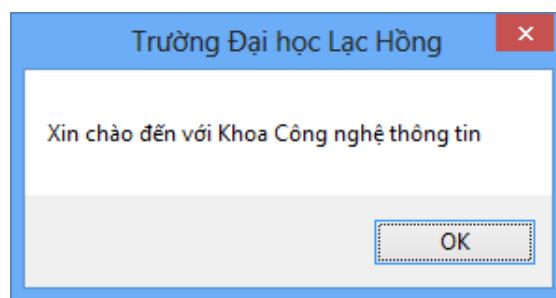


Hình 7.1: Hộp thoại *MessageBox* đơn giản

Mã lệnh:

```
MessageBox.Show("Xin chào đến với Khoa Công nghệ thông tin");
```

- Hiển thị hộp thoại với dòng thông báo trên *MessageBox* và dòng tiêu đề trên titlebar của hộp thoại như hình 7.2.

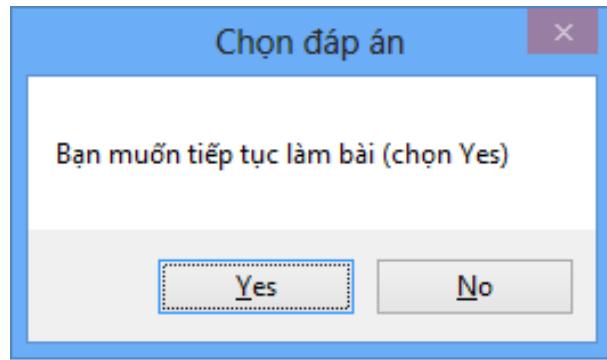


Hình 7.2: Hộp thoại *MessageBox* với dòng tiêu đề trên titlebar

Mã lệnh:

```
MessageBox.Show("Xin chào đến với Khoa Công nghệ thông tin",  
    "Trường Đại học Lạc Hồng");
```

- Hiển thị hộp thoại với hai nút Yes/ No như hình 7.3. Muốn hiển thị nút trong hộp thoại cần sử dụng tham số thuộc kiểu *MessageBoxButton*, đồng thời để biết người dùng nhấn nút Yes hay No thì cần sử dụng biến thuộc kiểu *DialogResult* để nhận kết quả trả về từ hộp thoại.

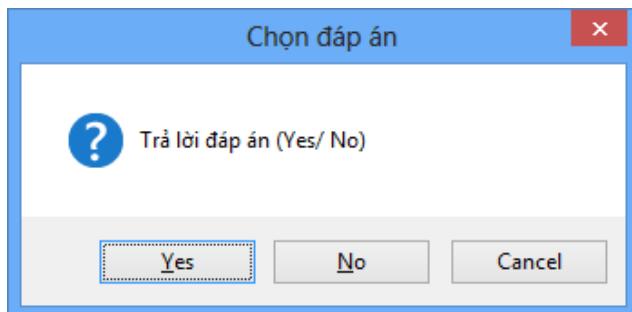


Hình 7.3: Hộp thoại MessageBox với hai nút Yes/ No

Mã lệnh:

```
DialogResult r = MessageBox.Show("Bạn muốn tiếp tục làm bài",
                                "Chọn đáp án",
                                MessageBoxButtons.YesNo);
if (r == DialogResult.Yes)
    MessageBox.Show("Bạn đã chọn Yes");
else
    MessageBox.Show("Bạn đã chọn No");
```

- Hiển thị hộp thoại có 3 nút Yes/ No/ Cancel và có thêm biểu tượng dạng câu hỏi cạnh chuỗi hiển thị trên MessageBox như hình 7.4.

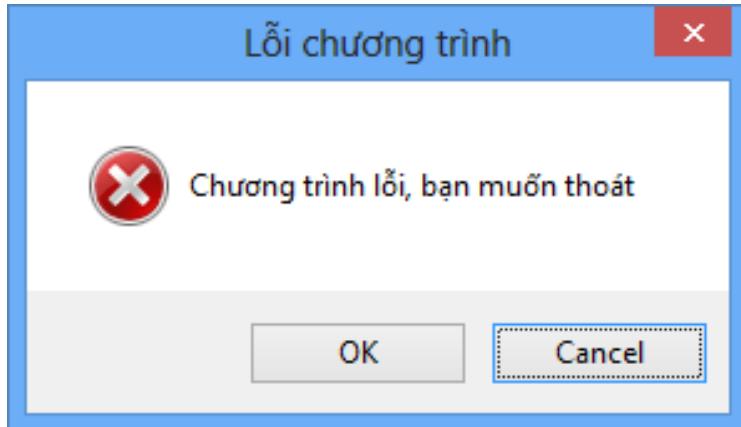


Hình 7.4: Hộp thoại MessageBox với biểu tượng dạng câu hỏi

Mã lệnh:

```
DialogResult r = MessageBox.Show("Trả lời đáp án (Yes/ No)",
                                "Chọn đáp án",
                                MessageBoxButtons.YesNoCancel,
                                MessageBoxIcon.Question);
if (r == DialogResult.Yes)
    MessageBox.Show("Bạn đã chọn Yes");
else
    if(r==DialogResult.No)
        MessageBox.Show("Bạn đã chọn No");
    else
        MessageBox.Show("Bạn đã thoát chương trình");
```

- Hiển thị hộp thoại có 2 nút Ok/ Cancel và có thêm biểu tượng dạng lỗi cạnh chuỗi hiển thị trên *MessageBox* như hình 7.5. Nút mặc định được Focus khi hộp thoại hiển thị là nút Cancel.



Hình 7.5: Hộp thoại *MessageBox* báo lỗi

Mã lệnh:

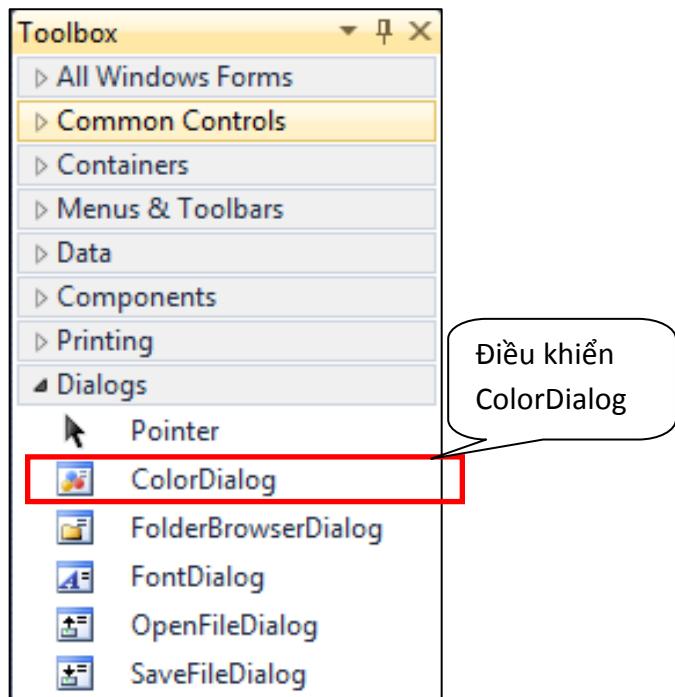
```
DialogResult r = MessageBox.Show("Chương trình lỗi, bạn muốn  
thoát", "Lỗi chương trình",  
MessageBoxButtons.OKCancel,  
MessageBoxIcon.Error,  
MessageBoxDefaultButton.Button2);  
  
if (r == DialogResult.OK)  
    MessageBox.Show("Bạn đã chọn OK, thoát chương trình");  
else  
    if(r==DialogResult.No)  
        MessageBox.Show("Bạn đã chọn Cancel, tiếp tục chạy  
chương trình");
```

## 7.2. Điều khiển ColorDialog

Để thiết lập màu cho các điều khiển, C# hỗ trợ lớp *ColorDialog* để người dùng sử dụng và hiển thị trực quan qua một hộp thoại gọi là *ColorDialog Box*.

*ColorDialog Box* chứa một danh sách các màu sắc được xác định cho hệ thống hiển thị. Người dùng có thể lựa chọn hoặc tạo ra một màu sắc đặc biệt từ danh sách, sau đó áp dụng khi thoát khỏi hộp thoại.

*ColorDialog* nằm trong nhóm *Dialog* của cửa sổ *Toolbox* như hình 7.6.

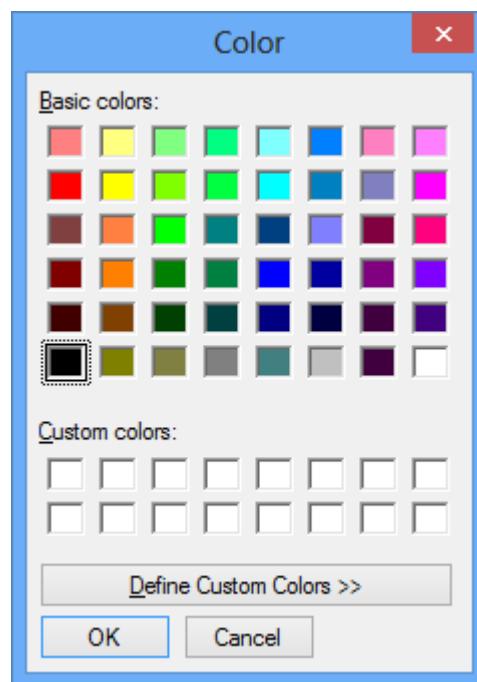


Hình 7.6: Điều khiển *ColorDialog* trong cửa sổ *Toolbox*

Lập trình có thể tạo ra đối tượng *ColorDialog* bằng cách kéo điều khiển vào form từ cửa sổ *Toolbox* hoặc sử dụng mã lệnh tạo đối tượng và hiển thị *ColorDialog Box* bằng cách gọi phương thức *ShowDialog()* để hiển thị hộp thoại như hình 7.7.

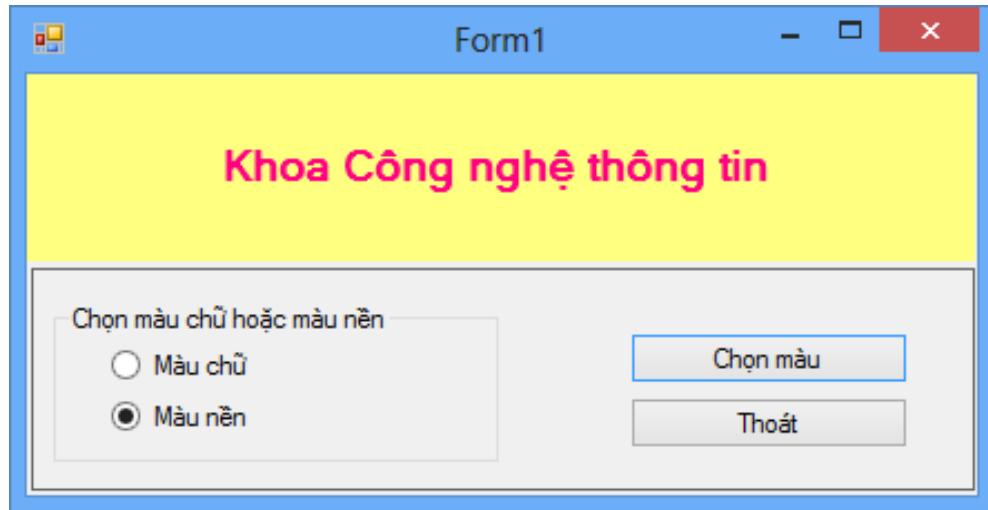
Mã lệnh:

```
ColorDialog clbox= new ColorDialog();
clbox.ShowDialog();
```



Hình 7.7: Giao diện *ColorDialog Box*

Ví dụ 7.2: Viết chương trình chọn màu chữ và màu nền cho Label có giao diện như hình 7.8.



Hình 7.8: Giao diện form lựa chọn màu cho Label

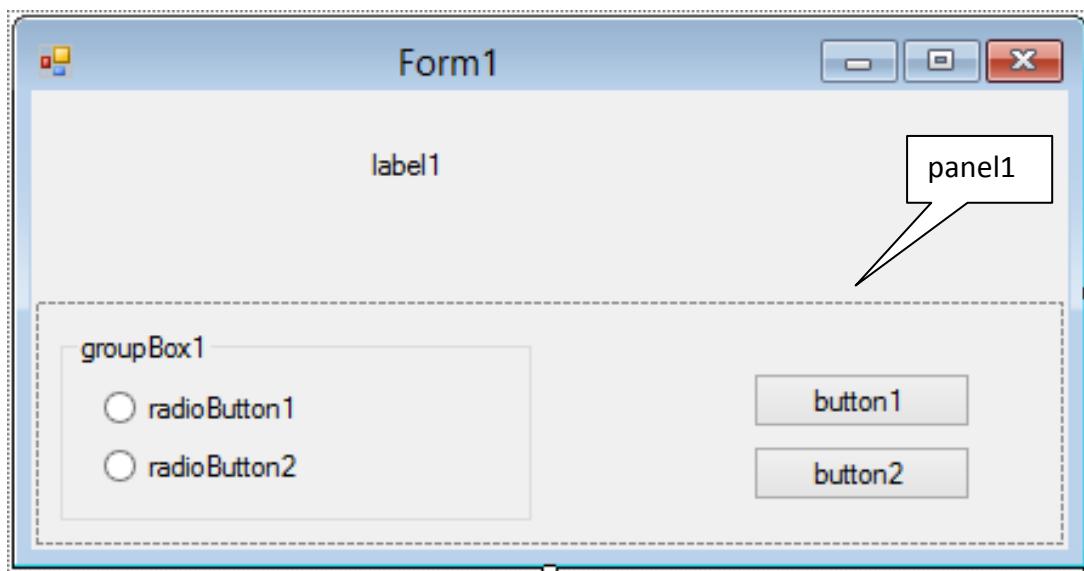
Yêu cầu:

Người dùng chọn thay đổi màu nền hoặc màu chữ của Label có dòng chữ “Khoa Công nghệ thông tin” bằng cách chọn một trong hai *RadioButton*: “Màu chữ”, “Màu nền”.

Sau khi đã có lựa chọn, người dùng nhấn vào nút “Chọn màu” để hiện thị *ColorDialog* Box và chọn màu muốn thay đổi cho *Label*.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển *Label*, *Button*, *Panel*, *GroupBox* và *RadioButton* từ cửa sổ *Toolbox* vào form như hình 7.9.



Hình 7.9: Giao diện form sau khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ *Properties*

- *label1*:

Thuộc tính *Text*: “Khoa Công nghệ thông tin”

- Thuộc tính *FontSize*: 14
- Thuộc tính *FontStyle*: Bold
- Thuộc tính *AutoSize*: False
- Thuộc tính *Size*: 428, 86
- Thuộc tính *TextAlign*: MiddleCenter
- panel1:
  - Thuộc tính *BorderStyle*: FixedSingle
- groupBox1:
  - Thuộc tính *Text*: “Chọn màu chữ hoặc màu nền”
- radioButton1:
  - Thuộc tính *Text*: “Màu chữ”
  - Thuộc tính *Name*: radMauChu
- radioButton2:
  - Thuộc tính *Text*: “Màu nền”
  - Thuộc tính *Name*: radMauNen
- button1:
  - Thuộc tính *Text*: “Chọn màu”
  - Thuộc tính *Name*: btnChonMau
- button2:
  - Thuộc tính *Text*: “Thoát”
  - Thuộc tính *Name*: btnThoat

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* nút btnChonMau:

```
private void btnChonMau_Click(object sender, EventArgs e)
{
    ColorDialog cldLog = new ColorDialog();
    DialogResult rs = cldLog.ShowDialog();
    if (rs == DialogResult.OK)
    {
        if (radMauChu.Checked == true)
        {
            lblHienThi.ForeColor = cldLog.Color;
        }
        if (radMauNen.Checked == true)
        {
            lblHienThi.BackColor = cldLog.Color;
        }
    }
}
```

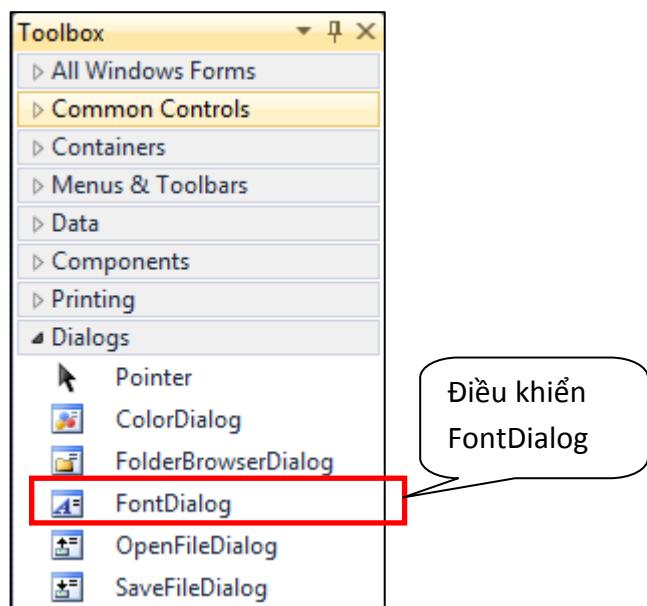
- Sự kiện Click nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

### 7.3. Điều khiển FontDialog

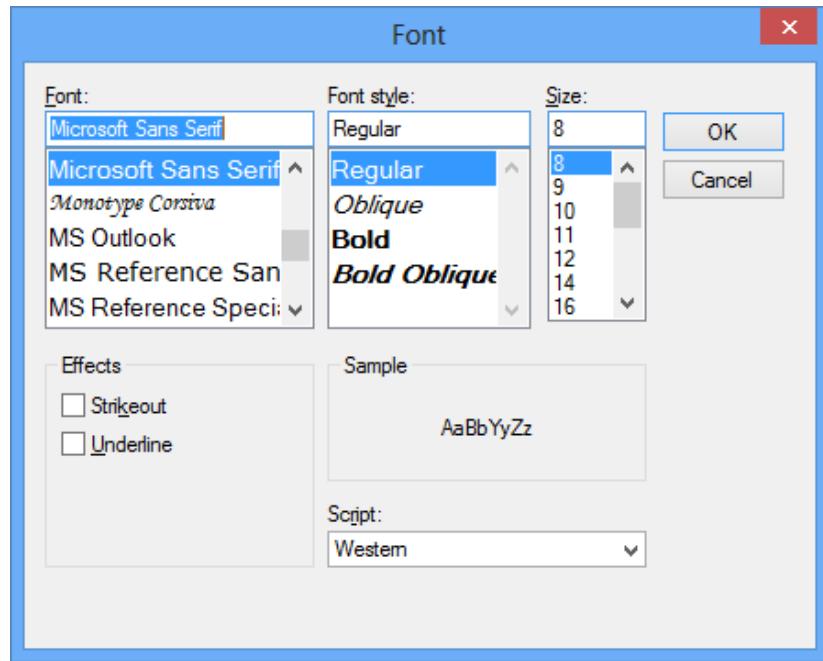
Tương tự như *ColorDialog Box*, để thiết lập font chữ hiển thị, C# cũng hỗ trợ hộp thoại gọi là *FontDialog Box*. *FontDialog Box* sẽ hiển thị một danh sách các font chữ hiện đang được cài đặt trên hệ thống; Cho phép người dùng lựa chọn các thuộc tính cho một font chữ hợp lý, như kiểu font chữ, kích cỡ chữ, hiệu ứng, ....

*FontDialog* nằm trong nhóm *Dialogs* của cửa sổ *Toolbox* như hình 7.10.



Hình 7.10: Điều khiển *FontDialog* trong cửa sổ *Toolbox*

Lập trình có thể tạo ra đối tượng *FontDialog* bằng cách kéo điều khiển vào form từ cửa sổ *Toolbox* hoặc sử dụng mã lệnh tạo đối tượng và hiển thị *FontDialog Box* bằng cách gọi phương thức *ShowDialog()* để hiển thị hộp thoại như hình 7.11.

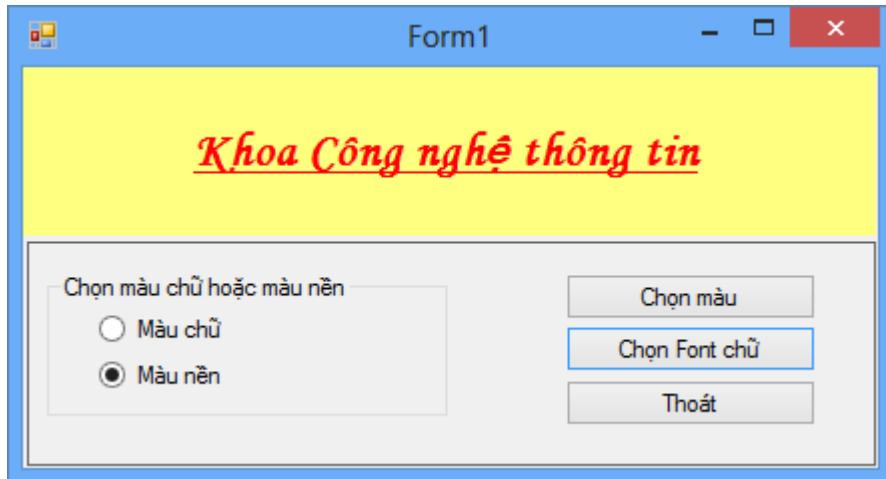


Hình 7.11: Giao diện FontDialog Box

Mã lệnh hiển thị *FontDialog Box*:

```
FontDialog fontbox= new FontDialog();
fontbox.ShowDialog();
```

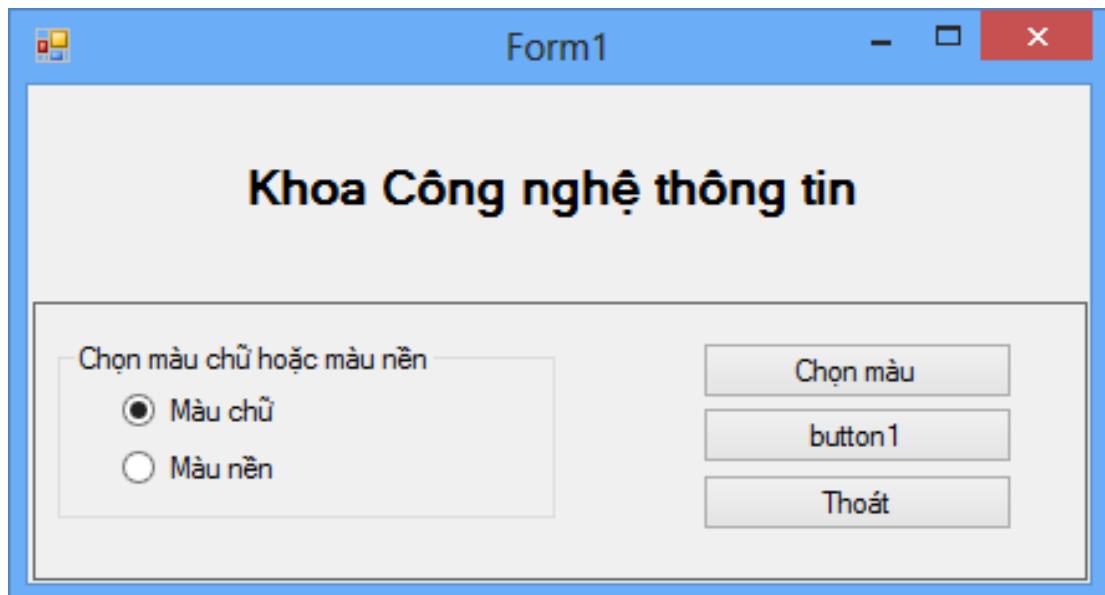
Ví dụ 7.3: Dựa vào ví dụ 7.2. Thêm nút lệnh “Chọn Font chữ” như hình 7.12



Hình 7.12: Giao diện form sau khi thêm nút “Chọn Font chữ”

Yêu cầu: Khi người dùng nhấn vào nút “Chọn Font chữ” sẽ hiển thị hộp thoại *FontDialog Box* cho phép người dùng thay đổi font chữ, kiểu chữ và kích thước của dòng chữ “Khoa Công nghệ thông tin”.

Bước 1: Thêm một điều khiển *Button* từ cửa sổ Toolbox vào frm như hình 7.13



Hình 7.13: Giao diện form sau khi thêm Button

Bước 2: Thiết lập giá trị thuộc tính cho nút button1:

Thuộc tính *Name*: btnChonFont

Thuộc tính *Text*: “Chọn Font chữ”

Bước 3: Viết mã lệnh cho sự kiện.

- Khai báo biến fontbox có kiểu *FontDialog*:

```
FontDialog fontbox = new FontDialog();
```

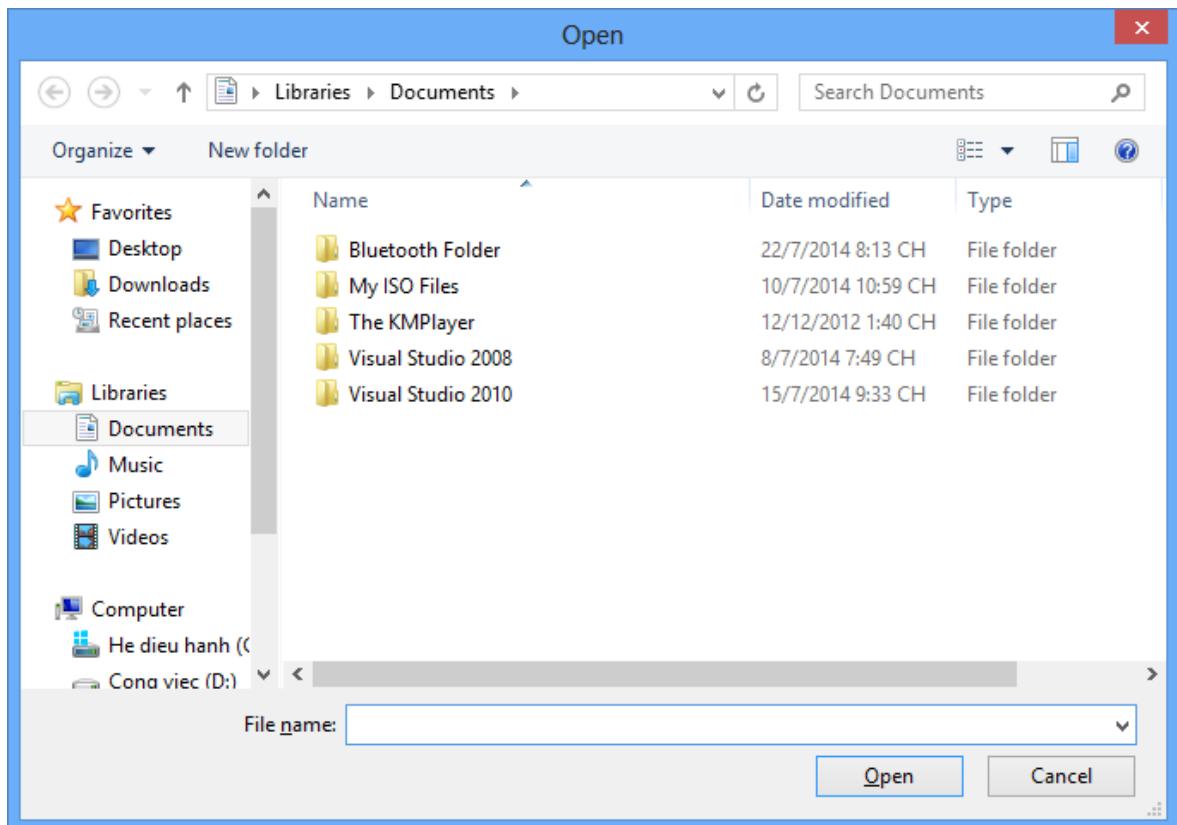
- Sự kiện *Click* nút btnChonFont

```
private void btnChonFont_Click(object sender, EventArgs e)
{
    DialogResult rs = fontbox.ShowDialog();
    if (rs == DialogResult.OK)
    {
        lblHienThi.Font = fontbox.Font;
    }
}
```

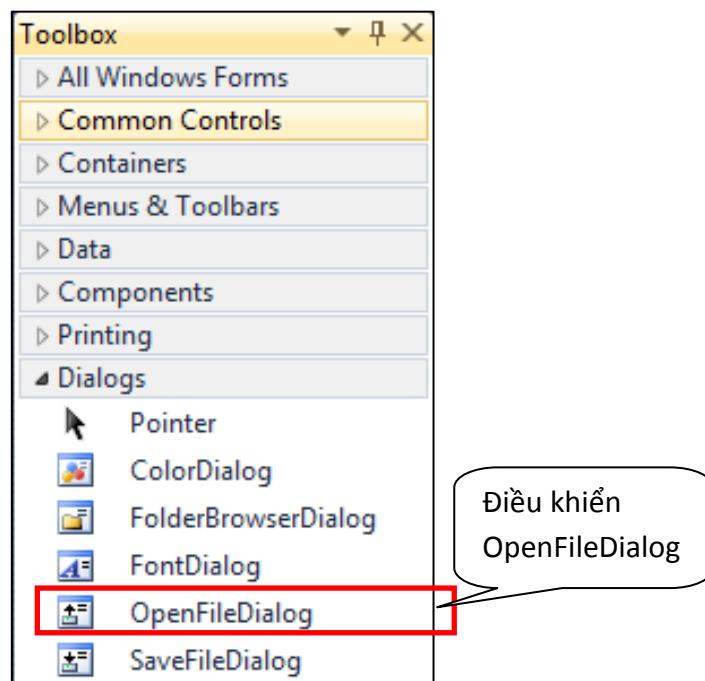
## 7.4. Điều khiển OpenFileDialog

Điều khiển *OpenFileDialog* cho phép hiển thị hộp thoại *OpenFileDialog* để lấy về ổ đĩa, tên thư mục, tên tập tin đối với một tập tin hay thư mục đang tồn tại. Thông thường các ứng dụng Windows Forms sử dụng hộp thoại *OpenFileDialog* để mở một tập tin nào đó lưu trên bộ nhớ máy tính.

Hộp thoại *OpenFileDialog* có giao diện như hình 7.14 và đặt trong nhóm *Dialogs* của cửa sổ *Toolbox* như hình 7.15.



Hình 7.14: Hộp thoại OpenFileDialog

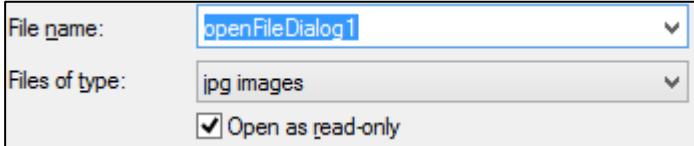


Hình 7.15: Điều khiển OpenFileDialog trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của OpenFileDialog:

Thuộc tính	Mô tả
Title	Thiết lập chuỗi hiển thị trên titlebar của hộp thoại OpenFileDialog

<i>Filter</i>	Thuộc tính nhận giá trị là một chuỗi. Chuỗi này chứa các đuôi mở rộng của tập tin, giúp lọc ra trên hộp thoại OpenFileDialog chỉ hiển thị đúng loại đuôi có trong chuỗi. Ví dụ: Thiết lập chuỗi chỉ hiển thị file hình ảnh có đuôi JPG và đuôi GIF: <b>“jpg images *.jpg gif images *.gif”</b> Thiết lập chuỗi hiển thị tất cả những gì có trong thư mục trên hộp thoại: <b>“All *.*”</b>
<i>FileName</i>	Thuộc tính trả về đường dẫn cùng tên tập tin được chọn
<i>FileNames</i>	Thuộc tính trả về một mảng các đường dẫn cùng tên tập tin được chọn. Lưu ý: Thuộc tính thường được sử dụng khi thuộc tính MultiSelect được thiết lập là True
<i>FilterIndex</i>	Thuộc tính sử dụng cùng với thuộc tính Filter. Có ý nghĩa sẽ ưu tiên lọc loại tập tin nào trước trong chuỗi Filter. Ví dụ: Chuỗi Filter lọc hai loại tập tin JPG và GIF: <b>“jpg images *.jpg gif images *.gif”</b> Nếu muốn hộp thoại khi hiển thị lọc hình có đuôi JPG đầu tiên thiết lập thuộc tính FilterIndex có giá trị 1. Nếu muốn hộp thoại khi hiển thị lọc hình có đuôi GIF đầu tiên thiết lập thuộc tính FilterIndex có giá trị 2.
<i>InitialDirectory</i>	Giá trị thuộc tính là một đường dẫn đến một thư mục hoặc ô đĩa. Khi hộp thoại OpenFileDialog được mở lên sẽ hiển thị nội dung của đường dẫn này đầu tiên.
<i>RestoreDirectory</i>	Mang hai giá trị True hoặc False, cho biết hộp thoại có trả lại đường dẫn thư mục vừa mở trước đó. Lưu ý: Thuộc tính chỉ có hiệu lực khi thuộc tính InitialDirectory mang giá trị rỗng
<i>MultiSelect</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép người dùng chọn nhiều tập tin hoặc thư mục</li> <li>- Nếu là False: Chỉ được chọn 1 tập tin hoặc một thư mục</li> </ul>
<i>CheckFileExists</i>	Mang hai giá trị True hoặc False.

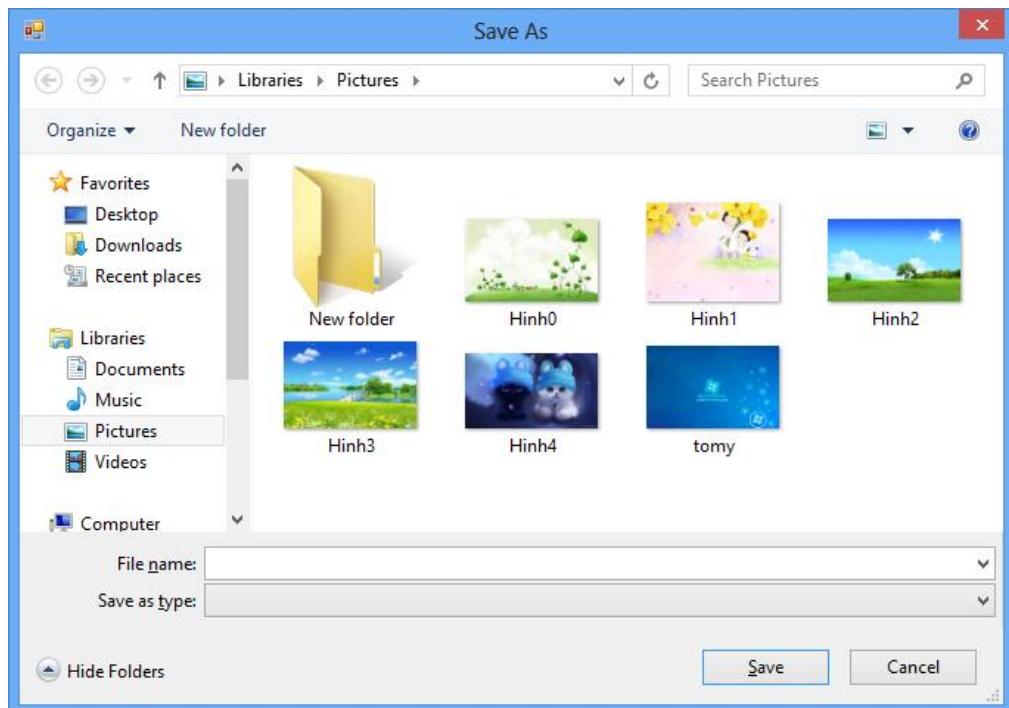
	Nếu là True sẽ cho phép hộp thoại hiển thị một cảnh báo nếu người dùng chỉ định một tập tin không tồn tại và nhấn nút Open.
<i>ReadOnlyChecked</i>	Mang giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Hiển thị một CheckBox với tiêu đề Open as Read Only phía dưới ComboBox Files Of Types</li> </ul>  <ul style="list-style-type: none"> <li>- Nếu là False: Không hiển thị CheckBox.</li> </ul> <p>Lưu ý: Thuộc tính chỉ có hiệu lực khi thuộc tính ShowReadOnly mang giá trị True</p>
<i>AddExtension</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True sẽ cho phép thêm vào tên mở rộng (jpg, gif, ...) vào tập tin.</li> <li>- Nếu là False: Không cho phép</li> </ul>
<i>DefaultExt</i>	Thêm tên mở rộng (.jpg, .gif, ...) cho tập tin nếu người dùng không cung cấp tên mở rộng.

- Một số phương thức thường dùng của *OpenFileDialog*:

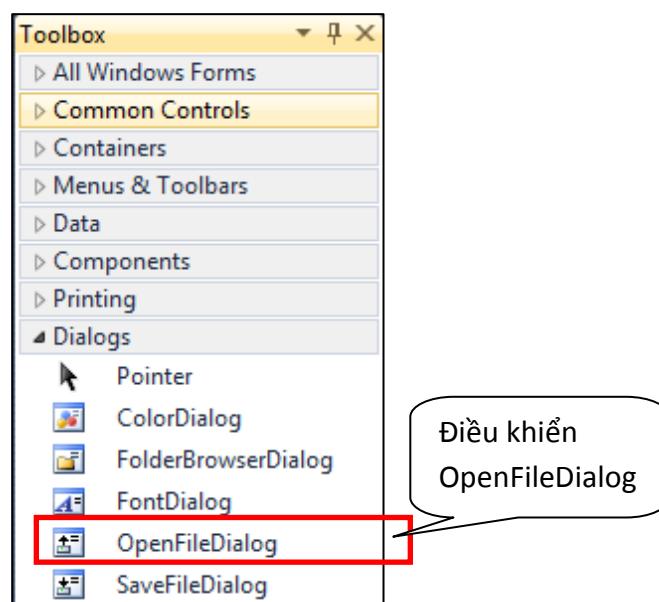
Phương thức	Mô tả
<i>Reset()</i>	Thiết lập giá trị các thuộc tính trở lại giá trị mặc định
<i>ShowDialog()</i>	Hiển thị hộp thoại, bắt buộc người dùng phải thao tác và đóng hộp thoại lại mới có thể thực hiện công việc khác.

## 7.5. Điều khiển SaveFileDialog

Điều khiển *SaveFileDialog* cho phép hiển thị hộp thoại *SaveFileDialog* để người dùng lựa chọn đường dẫn đến một thư mục hoặc ổ đĩa trên hệ thống để ghi một tập tin mới hoặc ghi đè tập tin đã có. Hộp thoại *SaveFileDialog* có giao diện như hình 7.16 và đặt trong nhóm Dialogs của cửa sổ Toolbox như hình 7.17.



Hình 7.16: Giao diện hộp thoại SaveFileDialog



Hình 7.17: Điều khiển SaveFileDialog trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của SaveFileDialog:

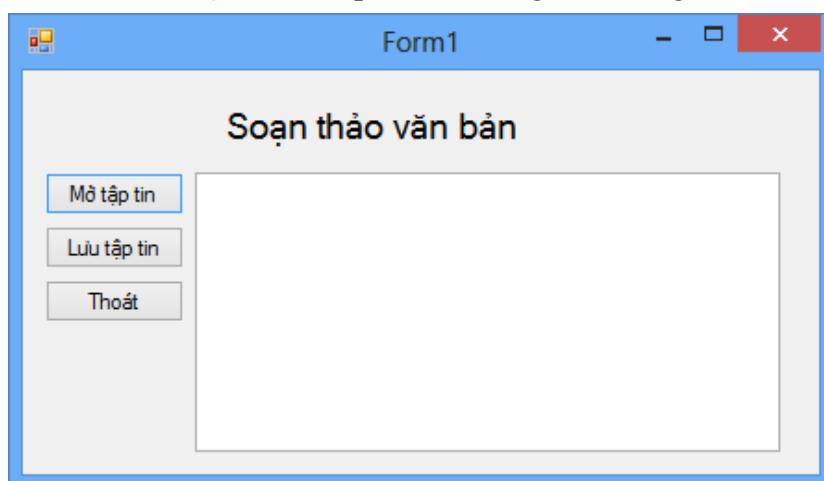
*SaveFileDialog* và *OpenFileDialog* đều thừa kế từ lớp cơ sở là *FileDialog* do đó *SaveFileDialog* có nhiều thuộc tính giống như *OpenFileDialog*. Một số thuộc tính *OpenFileDialog* có nhưng *SaveFileDialog* không có như: *Multiselect*, *ReadOnlyChecked*, *ShowReadOnly*. Do là hộp thoại giúp lưu tập tin do đó *SaveFileDialog* hỗ trợ hai thuộc tính mới là: *CreatePrompt* và *OverwritePrompt*.

Thuộc tính	Mô tả
<i>Title</i>	Thiết lập chuỗi hiển thị trên titlebar của hộp thoại OpenFileDialog

<i>Filter</i>	<p>Thuộc tính nhận giá trị là một chuỗi. Chuỗi này chứa các đuôi mở rộng của tập tin, giúp lọc ra trên hộp thoại OpenFileDialog chỉ hiển thị đúng loại đuôi có trong chuỗi.</p> <p>Ví dụ: Thiết lập chuỗi chỉ hiển thị file hình ảnh có đuôi JPG và đuôi GIF:</p> <p style="color: red;">“jpg images *.jpg gif images *.gif”</p> <p>Thiết lập chuỗi hiển thị tất cả những gì có trong thư mục trên hộp thoại:</p> <p style="color: red;">“All *.*”</p>
<i>FileName</i>	Thuộc tính trả về đường dẫn cùng tên tập tin được chọn
<i>FilterIndex</i>	<p>Thuộc tính sử dụng cùng với thuộc tính Filter. Có ý nghĩa sẽ ưu tiên lọc loại tập tin nào trước trong chuỗi Filter.</p> <p>Ví dụ: Chuỗi Filter lọc hai loại tập tin JPG và GIF:</p> <p style="color: red;">“jpg images *.jpg gif images *.gif”</p> <p>Nếu muốn hộp thoại khi hiển thị lọc hình có đuôi JPG đầu tiên thiết lập thuộc tính FilterIndex có giá trị 1.</p> <p>Nếu muốn hộp thoại khi hiển thị lọc hình có đuôi GIF đầu tiên thiết lập thuộc tính FilterIndex có giá trị 2.</p>
<i>InitialDirectory</i>	Giá trị thuộc tính là một đường dẫn đến một thư mục hoặc ổ đĩa. Khi hộp thoại OpenFileDialog được mở lên sẽ hiển thị nội dung của đường dẫn này đầu tiên.
<i>RestoreDirectory</i>	<p>Mang hai giá trị True hoặc False, cho biết hộp thoại có trả lại đường dẫn thư mục vừa mở trước đó.</p> <p>Lưu ý: Thuộc tính chỉ có hiệu lực khi thuộc tính InitialDirectory mang giá trị rỗng</p>
<i>CheckFileExists</i>	<p>Mang hai giá trị True hoặc False.</p> <p>Nếu là True sẽ cho phép hộp thoại hiển thị một cảnh báo nếu người dùng chỉ định một tập tin không tồn tại và nhấn nút Open.</p>
<i>CheckPathExists</i>	<p>Mang hai giá trị True hoặc False.</p> <p>Nếu là True sẽ kiểm tra đường dẫn tới tập tin có hợp lệ hay không.</p>
<i>AddExtension</i>	<p>Mang hai giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>- Nếu là True sẽ cho phép thêm vào tên mở rộng (jpg, gif, ..) vào tập tin.</li> </ul>

	<ul style="list-style-type: none"> <li>Nếu là False: Không cho phép</li> </ul>
<i>DefaultExt</i>	Thêm tên mở rộng (.jpg, .gif, ...) cho tập tin nếu người dùng không cung cấp tên mở rộng.
<i>OverwritePrompt</i>	Mang hai giá trị True và False. Nếu là True sẽ hiện cảnh báo khi người dùng ghi đè vào một tập tin đã tồn tại
<i>CreatePrompt</i>	Mang hai giá trị True và False. Nếu là True sẽ hiện thông báo khi người dùng lưu một tập tin mới

Ví dụ 7.4: Viết chương trình soạn thảo văn bản đơn giản sử dụng *OpenFileDialog* để mở tập tin và *SaveFileDialog* để lưu tập tin. Chương trình có giao diện như hình 7.18.



Hình 7.18: Giao diện chương trình soạn thảo văn bản

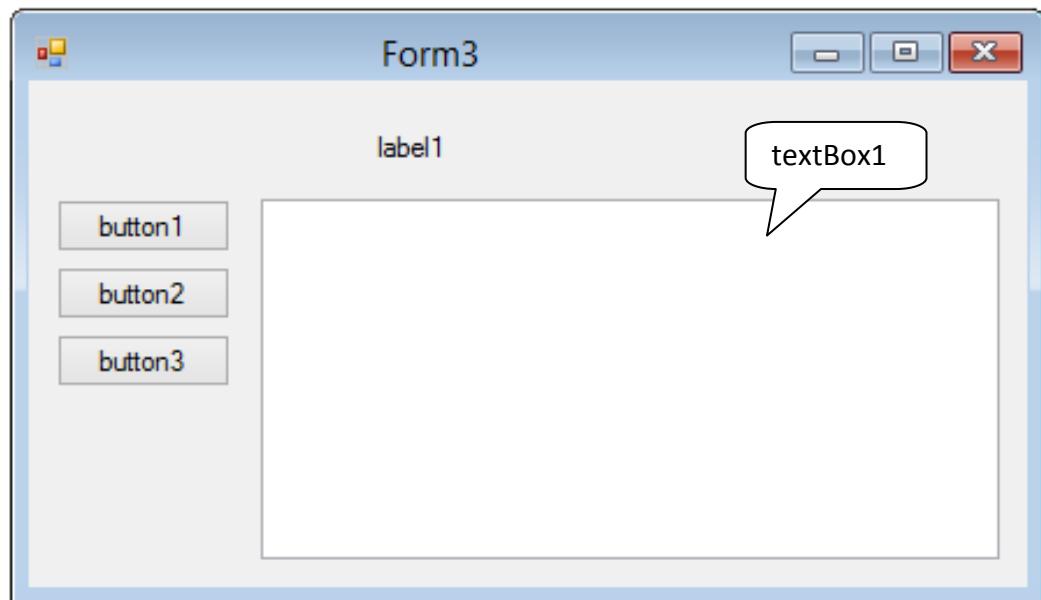
Yêu cầu:

Người dùng nhấn nút “Mở tập tin” để mở một tập tin có định dạng .txt. Nội dung của tập tin này sẽ hiển thị trên *TextBox* (chỉ cho phép hiển thị tập tin .txt)

Người dùng nhấn nút “Lưu tập tin” để lưu nội dung vừa soạn thảo trong *TextBox* vào một thư mục nào đó trên hệ thống.

Hướng dẫn:

Bước 1: Thiết kế giao diện chương trình. Lập trình viên thêm các điều khiển: *Label*, *TextBox*, *Button*, *OpenFileDialog* và *SaveFileDialog* từ cửa sổ Toolbox vào form như hình 7.19



Hình 7.19: Giao diện form soạn thảo văn bản khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties

- label1:

Thuộc tính *Text*: “Soạn thảo văn bản”

Thuộc tính *Font Size*: 14

- button1:

Thuộc tính *Name*: btnMoTapTin

Thuộc tính *Text*: “Mở tập tin”

- button2:

Thuộc tính *Name*: btnLuuTapTin

Thuộc tính *Text*: “Lưu tập tin”

- button3:

Thuộc tính *Name*: btnThoat

Thuộc tính *Text*: “Thoát”

- textBox1:

Thuộc tính *Name*: txtNoiDung

Thuộc tính *MultiLine*: True

Thuộc tính *Size*: 93, 55

- openFileDialog1:

Thuộc tính *RestoreDirectory*: True

Thuộc tính *Title*: “Mở tập tin Text để soạn thảo”

Thuộc tính *Filter*: “File Text|\*.txt”

- saveFileDialog1:
    - Thuộc tính *RestoreDirectory*: True
    - Thuộc tính *Title*: “Lưu tập tin Text”
    - Thuộc tính *Filter*: “File Text|\*.txt”
    - Thuộc tính *OverwritePrompt*: True
- Bước 3: Viết mã lệnh cho các điều khiển
- Sự kiện *Click* của nút btnMoTapTin:

```
private void btnMoTapTin_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string tentaptin = openFileDialog1.FileName;
        StreamReader rd = new StreamReader(tentaptin);
        txtNoiDung.Text = rd.ReadToEnd();
        rd.Close();
    }
}
```

- Sự kiện *Click* của nút btnThoat:

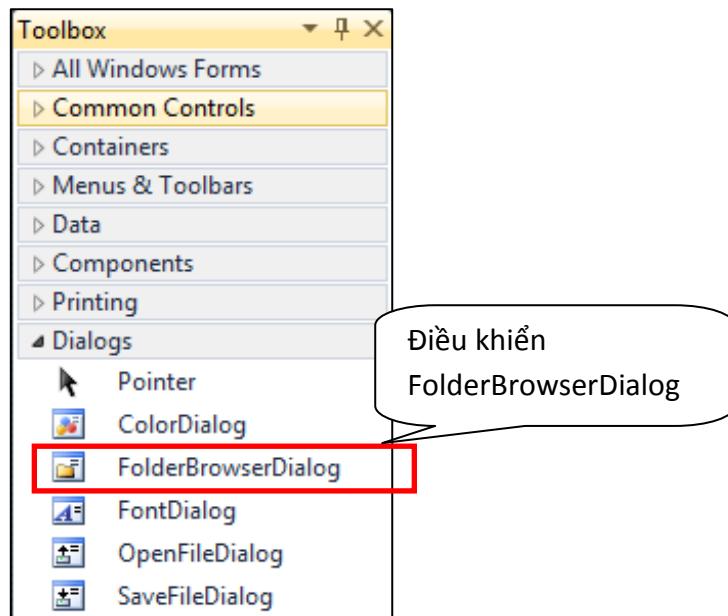
```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Click* của nút btnLuuTapTin:

```
private void btnLuuTapTin_Click(object sender, EventArgs e)
{
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string noidung = txtNoiDung.Text;
        string tentaptin = saveFileDialog1.FileName;
        StreamWriter wt = new StreamWriter(tentaptin);
        wt.WriteLine(noidung);
        wt.Close();
    }
}
```

## 7.6. Điều khiển FolderBrowserDialog

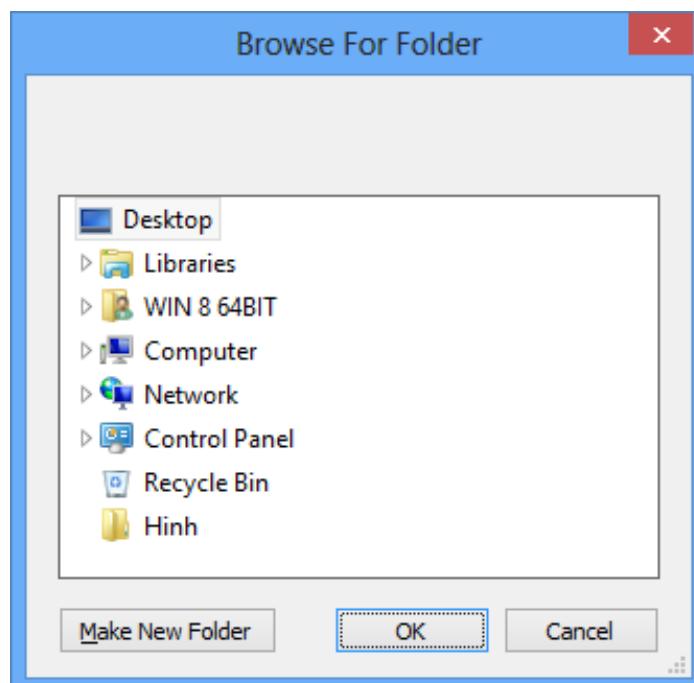
Điều khiển *FolderBrowserDialog* cho phép người dùng chọn một thư mục nào đó đang tồn tại trong hệ thống. Ngoài ra trong quá trình chọn người dùng cũng có thể tạo thư mục mới trên thư mục đang chọn. Điều khiển *ForlderBrowserDialog* nằm trong nhóm Dialogs của cửa sổ Toolbox như hình 7.20



Hình 7.20: Điều khiển *FolderBrowserDialog* trong cửa sổ *Toolbox*

Sự khác biệt cơ bản *FolderBrowserDialog* và *OpenFileDialog* là *FolderBrowserDialog* chỉ cho phép chọn thư mục mà không phải là chọn tập tin và mở như *OpenFileDialog*.

Lập trình viên có thể tạo đối tượng *FolderBrowserDialog* bằng cách kéo điều khiển từ cửa sổ *Toolbox* vào form hoặc viết mã lệnh để tạo đối tượng *FolderBrowserDialog* và hiển thị hộp thoại *FolderBrowserDialog* bằng phương thức *ShowDialog*. Giao diện hộp thoại *FolderBrowserDialog* như hình 7.21

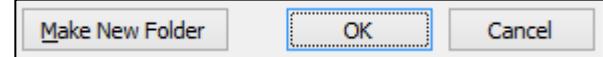


Hình 7.21: Giao diện hộp thoại *FolderBrowserDialog*

Mã lệnh tạo đối tượng và hiển thị hộp thoại *FolderBrowserDialog*:

```
FolderBrowserDialog fd = new FolderBrowserDialog();
fd.ShowDialog();
```

- Một số thuộc tính thường dùng của *FolderBrowserDialog*:

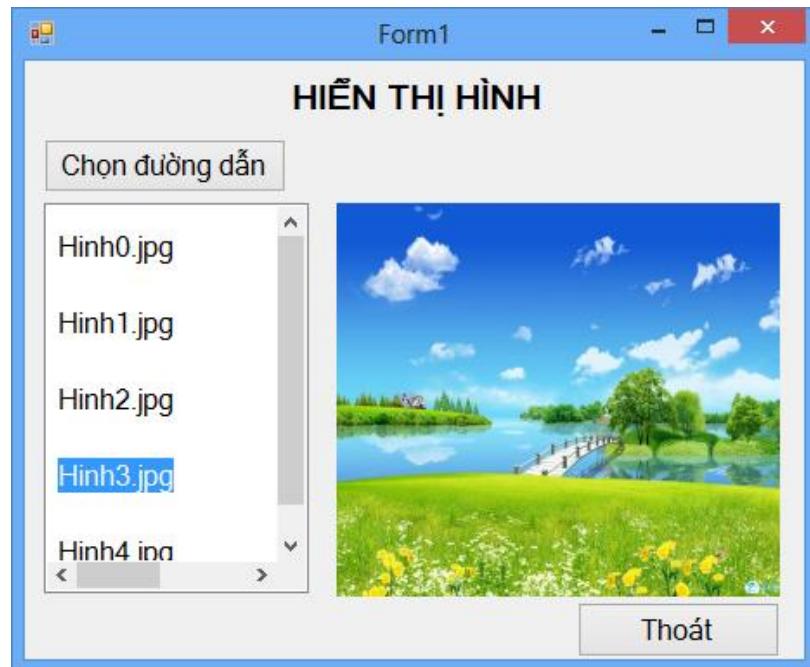
Thuộc tính	Mô tả
<i>Name</i>	Thiết lập tên cho <i>FolderBrowserDialog</i>
<i>SelectedPath</i>	Trả về đường dẫn cùng tên của thư mục mà người dùng chọn
<i>Description</i>	Thiết lập chuỗi mô tả hộp thoại, chuỗi mô tả này được hiển thị dưới titlebar của <i>FolderBrowserDialog</i>
<i>ShowNewFolderButton</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Sẽ xuất hiện nút lệnh “Make New Folder” phía dưới bên trái nút lệnh “OK”. Nút lệnh này cho phép người dùng tạo một thư mục mới bên trong thư mục đang chọn.</li> <li>- Nếu là False: Không hiển thị nút lệnh “Make New Folder”</li> </ul> 
<i>RootFolder</i>	Thiết lập thư mục mặc định sẽ mở ra khi hiển thị hộp thoại <i>FolderBrowserDialog</i> . Các vị trí thư mục mặc định sẽ mở đã được thiết lập sẵn để lập trình viên lựa chọn.

Ví dụ 7.5: Viết chương trình hiển thị hình ảnh có trên hệ thống máy tính. Giao diện chương trình như hình 7.21.

Yêu cầu:

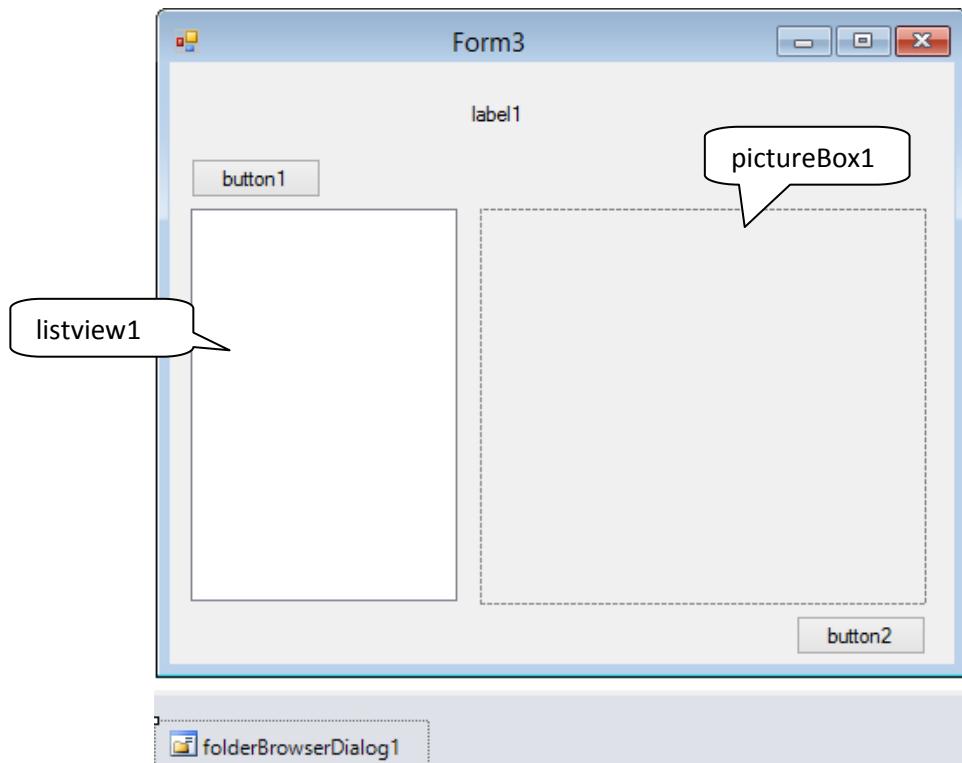
Chương trình cho phép người dùng chọn đường dẫn đến thư mục chứa hình. Một *ListView* trên form sẽ hiển thị danh sách tên các hình có đuôi tập tin dạng .JPG, Khi

người dùng chọn một hình trên *ListView* thì hình đó sẽ tương ứng hiển thị trên *PictureBox*.



Hình 7.21: Giao diện chương trình hiển thị hình ảnh

Bước 1: Thiết kế giao diện chương trình. Thêm các điều khiển: *Label*, *Button*, *PictureBox*, *ListView* và *FolderBrowserDialog* vào form như hình 7.22.



Hình 7.22: Giao diện form sau khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties  
- *label1*:

Thuộc tính *Text*: “HIỀN THỊ HÌNH”

Thuộc tính *Font Size*: 14

Thuộc tính *Font Style*: Bold

- button1:

Thuộc tính *Text*: “Chọn đường dẫn”

Thuộc tính *Name*: btnChon

- button2:

Thuộc tính *Text*: “Thoát”

Thuộc tính *Name*: btnThoat

- listView1:

Thuộc tính *Name*: listPath

Thuộc tính *View*: Tile

Thuộc tính *MultiSelect*: False

Thuộc tính *AllowDrop*: True

- pictureBox1:

Thuộc tính *Name*: picboxHienThi

Thuộc tính *SizeMode*: StretchImage

- folderBrowserDialog1:

Thuộc tính *Name*: myFolderBrowser

Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của nút btnChon:

```
private void btnChon_Click(object sender, EventArgs e)
{
    DialogResult dr = myFolderBrowser.ShowDialog();
    string[] path =
        Directory.GetFiles(myFolderBrowser.SelectedPath, "*.jpg");
    if (dr == DialogResult.OK)
    {
        foreach (string i in path)
        {
            int vt = i.LastIndexOf('\\');
            listPath.Items.Add(i.Substring(vt + 1, i.Length - vt - 1));
        }
    }
}
```

- Sự kiện *Click* của nút btnThoat:

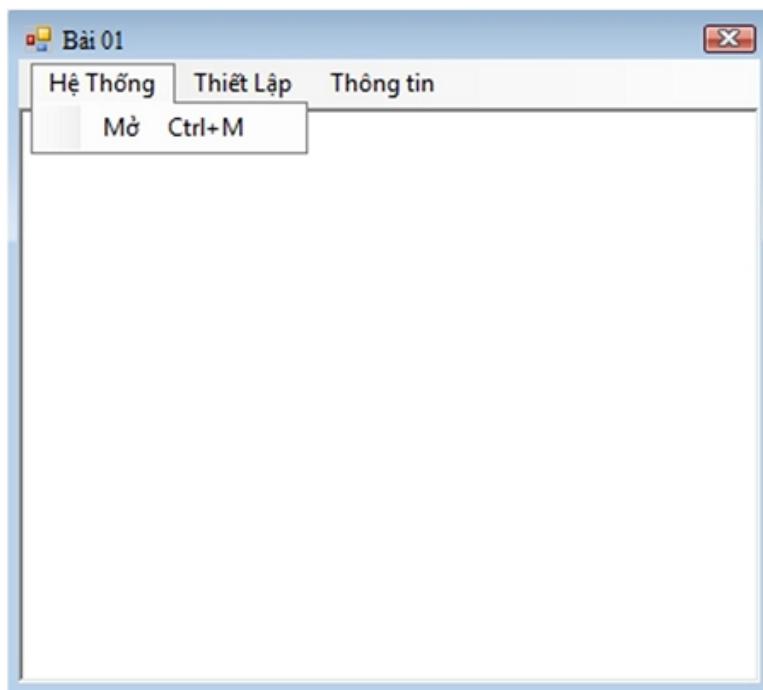
```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *SelectedIndexChanged* của nút *listPath*:

```
private void listPath_SelectedIndexChanged(object sender,
EventArgs e)
{
    picboxHienThi.ImageLocation =
        folderBrowserDialog1.SelectedPath
        + "\\\" + listPath.FocusedItem.Text;
}
```

## 7.7. Bài tập cuối chương

Câu 1: Thiết kế chương trình xử lý văn bản có giao diện như hình 7.23.



Hình 7.23: Giao diện chương trình xử lý văn bản

Xử lý menu “Hệ Thống”:

- Mở (Ctr+M): dùng OpenFileDialog để mở một file text bất kỳ hiện có trên ổ cứng máy tính và hiển thị nội dung của file text đó. Nếu không có sinh viên tự tạo ra một file text với nội dung bất kỳ.

Xử lý menu “Thiết Lập”:

- Đổi (Ctrl+D): để viết IN kí tự đầu của mỗi từ và cắt bỏ những kí tự trắng hiện có trong đoạn văn bản đang hiển thị.
- Font (Ctrl+F): dùng FontDialog và ColorDialog để thay đổi font, size, color của vùng văn bản đang được chọn khỏi trong khung hiển thị.

Xử lý menu “Thông tin”:

- Sinh viên: Hiển thị MessageBox bao gồm các thông tin Họ tên – Mã số sinh viên của sinh viên.

Gợi ý: khoảng trắng thừa là những khoảng trắng đứng đầu và cuối đoạn văn bản, những khoảng trắng đứng sau một khoảng trắng khác, những khoảng trắng đứng trước các dấu ngắt câu như dấu phẩy “,” dấu chấm “.” dấu chấm phẩy “;” dấu hai chấm “::” dấu hỏi “?” dấu chấm than “!”

Câu 2: Thiết kế chương trình tính toán có giao diện như hình 7.24.



Hình 7.24: Giao diện chương trình máy tính

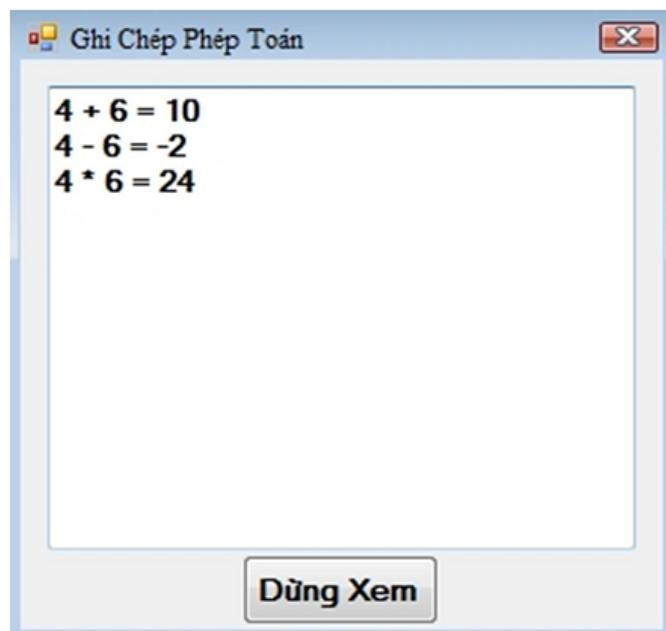
Yêu cầu:

- Xử lý sự kiện điều khiển form hình 7.24:

- Khi chưa nhấn lên Button “On/Off” một lần nào hoặc số lần kích lên nút là chẵn thì máy tính trong điều kiện không sẵn sàng sử dụng, nghĩa là không cho tác động lên các thành phần khác trên form máy tính.
- Mặc định, RadioButton “Có” đang được chọn.
- Bẫy lỗi nhập, chỉ cho nhập số vào hai TextBox txtSoA và txtSoB.
- TextBox kết quả không cho người dùng nhập liệu mà chỉ để xuất kết quả của phép toán khi người dùng nhấn các Button “Cộng”, “Trừ”, “Nhân”, “Chia” tương ứng. Khi kích nút phép toán nào thì nút đang được chọn sẽ đổi màu nền. Nếu kích lên nút phép toán khác thì nút phép toán đã được kích trước đó sẽ trở về màu nền ban đầu.
- Khi RadioButton “Có” đang được chọn, cứ mỗi phép toán ra được kết quả thì ghi toàn bộ phép toán đó thành một dòng dạng Giá trị A Tên phép toán Giá trị B = Kết quả tính vào tập tin GhiText.txt (xem hình 7.25), đồng thời, cho tác

động Button “Xem File”. File GhiText.txt phải đặt cùng vị trí với file chạy (.exe) của chương trình.

- Khi chọn RadioButton “Không” thì tập tin GhiText.txt hiện có sẽ bị xóa, đồng thời, không cho tác động lên Button “Xem File”.
- Kích Button “Xem File” cho phép hiển thị form ghi chép phép toán (hình 7.25)
- Kích Button “Làm lại” sẽ xóa sạch nội dung các ô nhập, ô kết quả và nội dung tập tin GhiText.txt, chờ thực hiện bài toán mới



Hình 7.25: Giao diện form ghi chép phép toán

➤ Xử lý sự kiện điều khiển form hình 7.25:

- Khi load form, xuất hiện OpenFileDialog cho chọn tập tin GhiText.txt và hiển thị nội dung file lên form.
- Button “Dừng Xem” thì đóng form ghi chép phép toán để trở về màn hình form máy tính.

Câu 3: Thiết kế chương trình tính toán Fibonacy có giao diện như hình 7.26.



Hình 7.26: Giao diện chương trình tính toán Fibonacy

Yêu cầu:

- Button “Mở/Tắt”: Khi chưa kích lên nút một lần nào hoặc số lần kích lên nút là chẵn thì tắt máy, nghĩa là không cho tác động lên các thành phần khác trên form. Nếu số lần kích lên nút là lẻ thì mở máy, nghĩa là cho tác động lên các thành phần còn lại trên form.
- Nhập và tính kết quả:  
Cách tính giá trị từng số hạng trong dãy Fibonaci:  $F_0=1$ ,  $F_1=1$ ,  $F_n=F_{n-1}+F_{n-2}$  (với  $n \geq 2$ )  
  - o Hình 7.27: Giao diện ban đầu khi chưa nhập số n, chờ nhập số n. Bẫy lỗi nhập, chỉ cho nhập vào một số n thỏa  $2 \leq n \leq 10$ .

Hình 7.27: Giao diện ban đầu khi chưa nhập n

- o Hình 7.28: khi vừa nhập vào số n thỏa điều kiện, hiển thị giá trị của  $F_0$  và  $F_1$ . Đồng thời, điền thêm giá trị n vào nhãn của số hạng F đang để trống.

Các số hạng của dãy Fibonaci

Số hạng thứ 0 (F0) =	1
Số hạng thứ 1 (F1) =	1
Số hạng thứ 5 (F5) =	

Hình 7.28: Giao diện khi nhập n thỏa điều kiện

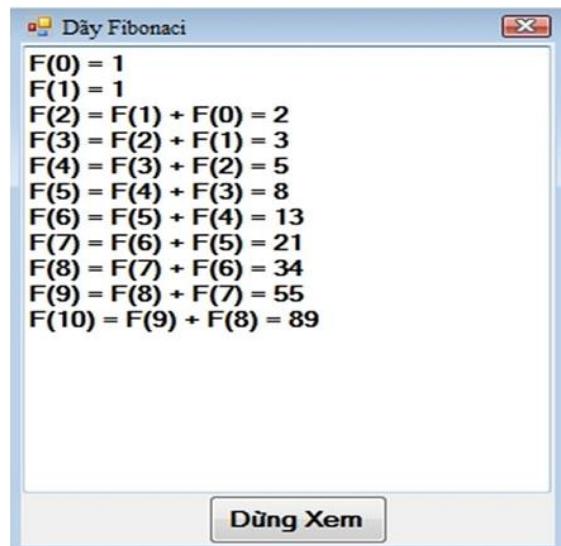
- Hình 7.29: khi nhấp chọn Button “=”, in giá trị số hạng  $F_n$  vào ô kết quả. Khi đã in kết quả thì không cho thay đổi giá trị trong ô nhập n.

Các số hạng của dãy Fibonaci

Số hạng thứ 0 (F0) =	1
Số hạng thứ 1 (F1) =	1
Số hạng thứ 5 (F5) =	8

Hình 7.29: Kết quả tính Fibonacy khi  $n = 5$

- GroupBox “Ghi từng số hạng vào file văn bản”:
  - Khi radioButton “Có” đang được chọn, cứ mỗi giá trị n nhập hợp lệ và ra được kết quả thì ghi vào tập tin GhiChep.txt đầy đủ từng số hạng từ F0 → Fn, mỗi số hạng ghi thành một dòng dạng giống như trong Hình 2, đồng thời, cho tác động Button “Xem File”.  
(Tập tin GhiChep.txt phải đặt cùng vị trí với file chạy (.exe) của chương trình.)
  - Khi chọn vào radioButton “Không” thì tập tin GhiChep.txt hiện có sẽ bị xóa, đồng thời, không cho tác động lên Button “Xem File”.
  - Nhấn chọn Button “Xem File” cho hiển thị form Dãy Fibonaci trên form như hình 7.30.



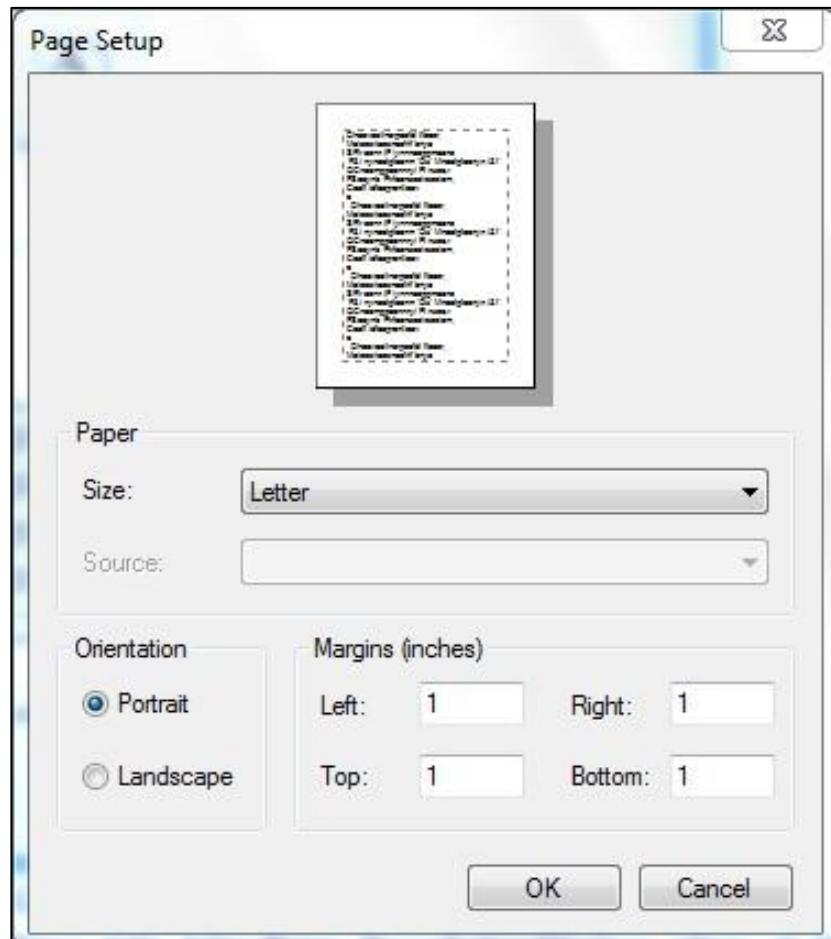
- *Hình 7.30: Giao diện form hiển thị kết quả Fibonacy từ  $F_0$  đến  $F_{10}$*

- Xử lý form Dãy Fibonaci:
  - o Khi load form, xuất hiện OpenFileDialog cho chọn tập tin GhiChep.txt và hiển thị nội dung tập tin lên form.
  - o Nhấn chọn Button “Dừng Xem” thì đóng form Dãy Fibonaci để trở về form hình 7.26.
- Nhấn chọn Button “Tiếp tục” xóa sạch nội dung các ô nhập, ô xuất kết quả và nội dung tập tin GhiChep.txt và chờ nhập số n mới.

## CHƯƠNG 8: LÀM VIỆC VỚI ĐIỀU KHIỂN IN ÂN

### 8.1. Điều khiển PageSetupDialog

Điều khiển *PageSetupDialog* được hiển thị dạng hộp thoại khi chương trình thực thi. Việc hiển thị dạng hộp thoại cho phép người dùng dễ dàng thay đổi các thiết lập về trang như: canh lề, định hướng, kích thước trang, . . . . Hộp thoại *PageSetupDialog* có giao diện như hình 8.1.



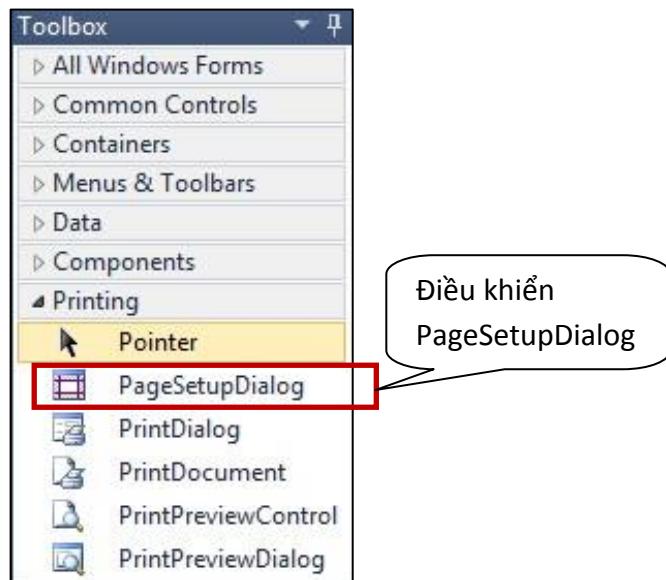
Hình 8.1: Giao diện hộp thoại *PageSetupDialog*

Những thay đổi của người dùng trên hộp thoại *PageSetupDialog* cũng sẽ làm thay đổi các giá trị tương ứng trên hộp thoại *PrintDocument*. Do đó việc thiết lập các giá trị trong hộp thoại *PageSetupDialog* cũng ảnh hưởng đến hình dạng của trang khi in ấn.

Hiển thị hộp thoại *PageSetupDialog* sử dụng phương thức *ShowDialog()*. Tuy nhiên, điểm khác biệt là để có thể gọi được phương thức *ShowDialog()* thì cần phải thực hiện một thao tác trước là gán một đối tượng thuộc lớp *PrintDocument* cho thuộc tính *Document* của điều khiển *PageSetupDialog*. Bởi vì như đã nói ở trên là *PageSetupDialog* và *PrintDocument* có liên hệ với nhau. Những thay đổi tùy chọn trong *PageSetupDialog* đều ảnh hưởng đến hình dạng của trang khi in, mặt khác

*PrintDocument* là đối tượng chính để thực hiện tiến trình in ấn. Do đó cần phải thiết lập một đối tượng thuộc lớp *PrintDocument* cho thuộc tính *Document* trước.

Điều khiển *PageSetupDialog* nằm trong nhóm Printing của cửa sổ Toolbox như hình 8.2.



Hình 8.2: Điều khiển *PageSetupDialog* trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của *PageSetupDialog*:

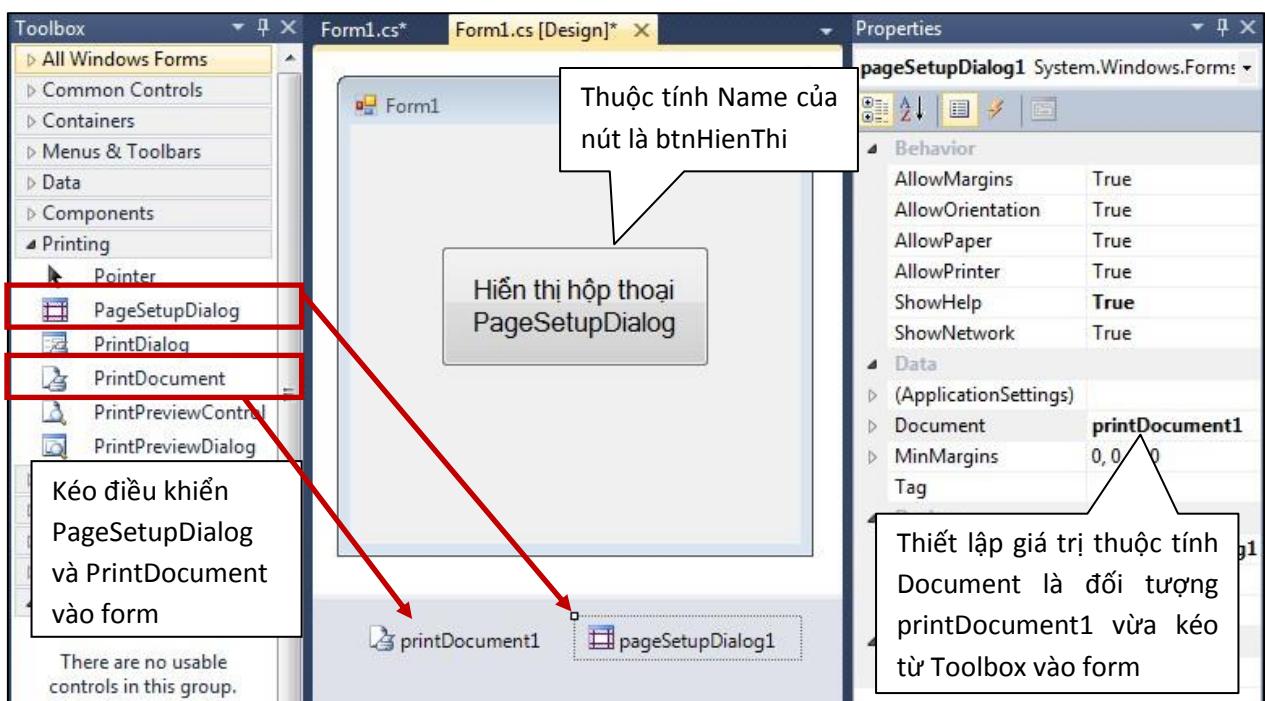
Bảng 8.1: Bảng mô tả các thuộc tính của *PageSetupDialog*

Thuộc tính	Mô tả
<i>AllowMargins</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép thiết lập độ rộng lề trên, dưới, phải và trái của trang</li> <li>- Nếu là False: Không cho phép thiết lập lề của trang</li> </ul>
<i>AllowOrientation</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép thiết lập trang hiển thị theo chiều ngang hoặc theo chiều dọc.</li> <li>- Nếu là False: Không cho phép thiết lập chiều hiển thị của trang.</li> </ul>
<i>AllowPaper</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép thiết lập kích thước của trang. Có thể thiết lập trang kích thước A4, A3, ...</li> <li>- Nếu là False: Không cho phép thiết lập kích thước của trang</li> </ul>
<i>AllowPrinter</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép người dùng sử dụng chọn nút máy in. Khi người dùng nhấp vào nút Printer sẽ</li> </ul>

	xuất hiện hộp thoại cho phép chọn máy in. - Nếu là False: Nút printer không có hiệu lực
<i>Document</i>	Mang giá trị là một đối tượng thuộc lớp PrintDocument.  Lưu ý: Thuộc tính <i>Document</i> phải được gán giá trị trước khi hộp thoại <i>PageSetupDialog</i> được hiển thị bằng phương thức <i>ShowDialog()</i> .
<i>MinMargins</i>	Kích thước nhỏ nhất mà người dùng có thể thiết lập để canh lề trên, dưới, trái hay phải của trang.
<i>ShowHelp</i>	Mang hai giá trị True hoặc False. - Nếu là True: Trên hộp thoại xuất hiện thêm nút Help - Nếu là False: Không xuất hiện nút Help trên hộp thoại
<i>ShowNetwork</i>	Mang hai giá trị True hoặc False. - Nếu là True: Trên hộp thoại xuất hiện nút Network. Nút này cho phép người dùng có thể chọn máy in mạng. - Nếu là False: Không hiển thị nút Network

➤ Hiển thị hộp thoại *PageSetupDialog*:

Lập trình viên có thể sử dụng điều khiển *PageSetupDialog* bằng cách thêm điều khiển *PageSetupDialog* từ cửa sổ Toolbox vào form. Sau đó, trước khi hiển thị hộp thoại *PageSetupDialog* bằng phương thức *ShowDialog()*, lập trình viên cần phải thiết lập thuộc tính *Document* bằng 1 đối tượng thuộc *PrintDocument* như hình 8.3.



Hình 8.3: Giao diện hiển thị hộp thoại PageSetupDialog

Sự kiện Click của nút btnHienThi:

```
private void btnHienThi_Click(object sender, EventArgs e)
{
    pageSetupDialog1.ShowDialog();
}
```

Ngoài ra, nếu không sử dụng điều khiển *PageSetupDialog* và *PrintDocument* bằng cách kéo thả vào form từ cửa sổ Toolbox, lập trình viên cũng có thể tạo đối tượng *PageSetupDialog* và hiển thị hộp thoại này bằng mã lệnh như sau:

```
private void btnHienThi_Click(object sender, EventArgs e)
{
    PageSetupDialog PageSD = new PageSetupDialog();
    System.Drawing.Printing.PrintDocument printDoc = new
        System.Drawing.Printing.PrintDocument();
    PageSD.Document = printDoc;
    PageSD.ShowDialog();
}
```

## 8.2. Điều khiển PrintPreviewDialog

Điều khiển *PrintPreviewDialog* giúp cho người dùng có thể xem trước trang in sẽ trông như thế nào trước khi trang đó được in ra trên giấy. Lý do điều khiển *PrintPreviewDialog* có thể xem trước được trang in như vậy là vì đã gọi phương thức *Print()* của đối tượng lớp *PrintDocument* được thiết lập trong thuộc tính *Document* của *PrintPreviewDialog*. Khi đó, thay vì phương thức *Print()* thực hiện chức năng in trực

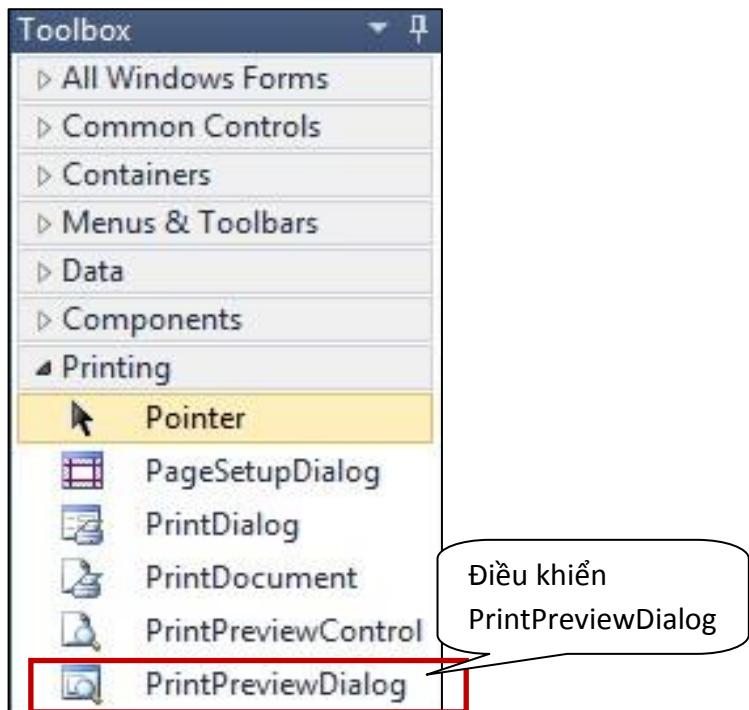
tiếp trên giấy thì trang in sẽ xuất ra trên một giao diện đồ họa được biểu diễn như một trang giấy của hộp thoại Print preview hình 8.4.



Hình 8.4: Giao diện PrintPreviewDialog khi được hiển thị

Trên giao diện hình 8.4 của điều khiển *PrintPreviewDialog*, người dùng có thể lựa chọn trang sẽ xem, phóng to hoặc thu nhỏ trang, lựa chọn chế độ hiển thị trang (1 trang, 2 trang, 3 trang, ...). Sau khi đã xem xong, người dùng có thể in trang hiển thị trên giấy bằng cách nhấn vào biểu tượng máy in trên hộp thoại Print preview.

Điều khiển *PrintPreviewDialog* nằm trong nhóm Printing của cửa sổ Toolbox như hình 8.5.



Hình 8.5: Điều khiển PrintPreviewDialog trong cửa sổ Toolbox

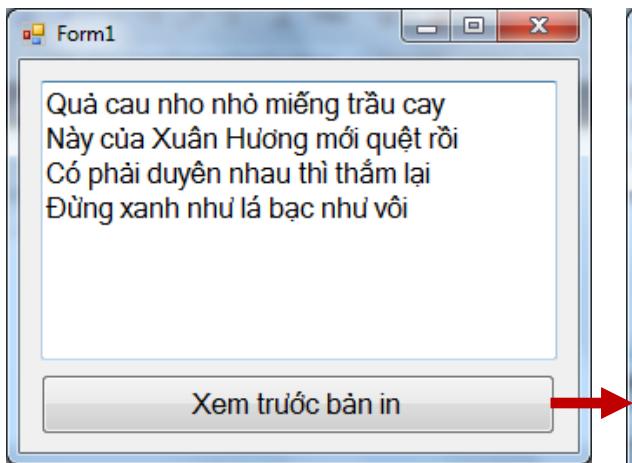
➤ Hiển thị hộp thoại PrintPreviewDialog:

Hộp thoại PrintPreviewDialog hiển thị bằng phương thức *ShowDialog()* hoặc phương thức *Show()*. Tuy nhiên trước khi hộp thoại được hiển thị lập trình viên cần thiết lập thuộc tính *Document* của PrintPreviewDialog có giá trị là một đối tượng *PrintDocument*. Lập trình viên có thể tạo đối tượng và hiển thị hộp thoại PrintPreviewDialog bằng mã lệnh như sau:

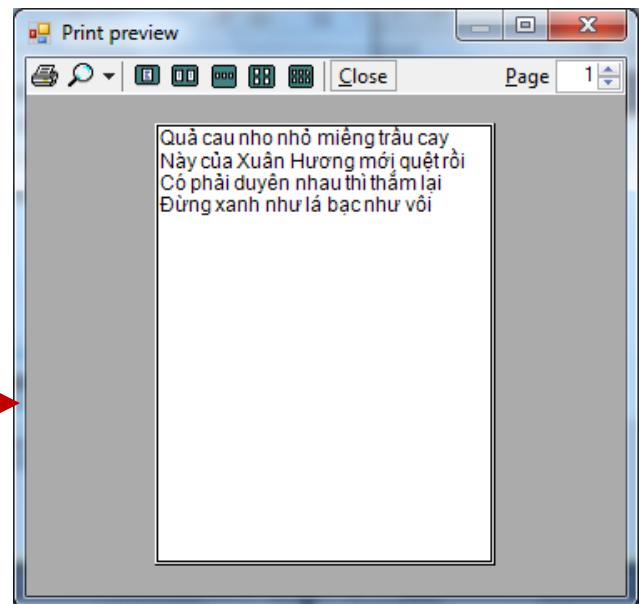
```
private void btnHienThi_Click(object sender, EventArgs e)
{
    PrintPreviewDialog preview = new PrintPreviewDialog ();
    System.Drawing.Printing.PrintDocument printDoc = new
        System.Drawing.Printing.PrintDocument();
    preview.Document = printDoc;
    preview.ShowDialog();
}
```

Ví dụ 8.1: Thiết kế chương trình có giao diện gồm *Button* và *Textbox* như hình 8.6.

Yêu cầu: Người dùng có thể nhập văn bản vào *Textbox* và khi ấn vào *Button* cho phép xem trang văn bản trước khi in như hình 8.7.



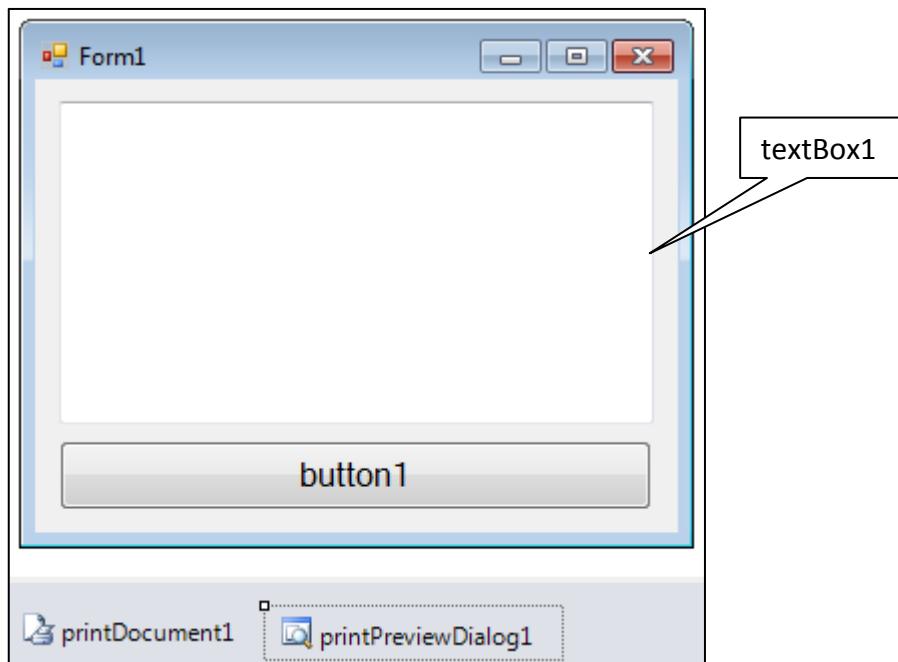
Hình 8.6: Giao diện chương trình nhập văn bản



Hình 8.7: Giao diện hộp thoại xem trang văn bản trước khi in

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển *Button*, *Textbox*, *PrintDocument* và *PrintPreviewDialog* từ cửa sổ Toolbox vào form như hình 8.8.



Hình 8.8: Giao diện ban đầu của form sau khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties.

- *textBox1*:

Thuộc tính *Name*: txtNoiDung

Thuộc tính *Multiline*: True

Thuộc tính *Size*: 297, 161

- Button1:  
 Thuộc tính *Name*: btnXemBanIn  
 Thuộc tính *Text*: “Xem trước bản in”  
 Thuộc tính *Size*: 296, 35
  - printPreviewDialog1:  
 Thuộc tính *Document*: printDocument1
- Bước 3: Viết mã lệnh cho các điều khiển
- Xây dựng hàm DrawText:

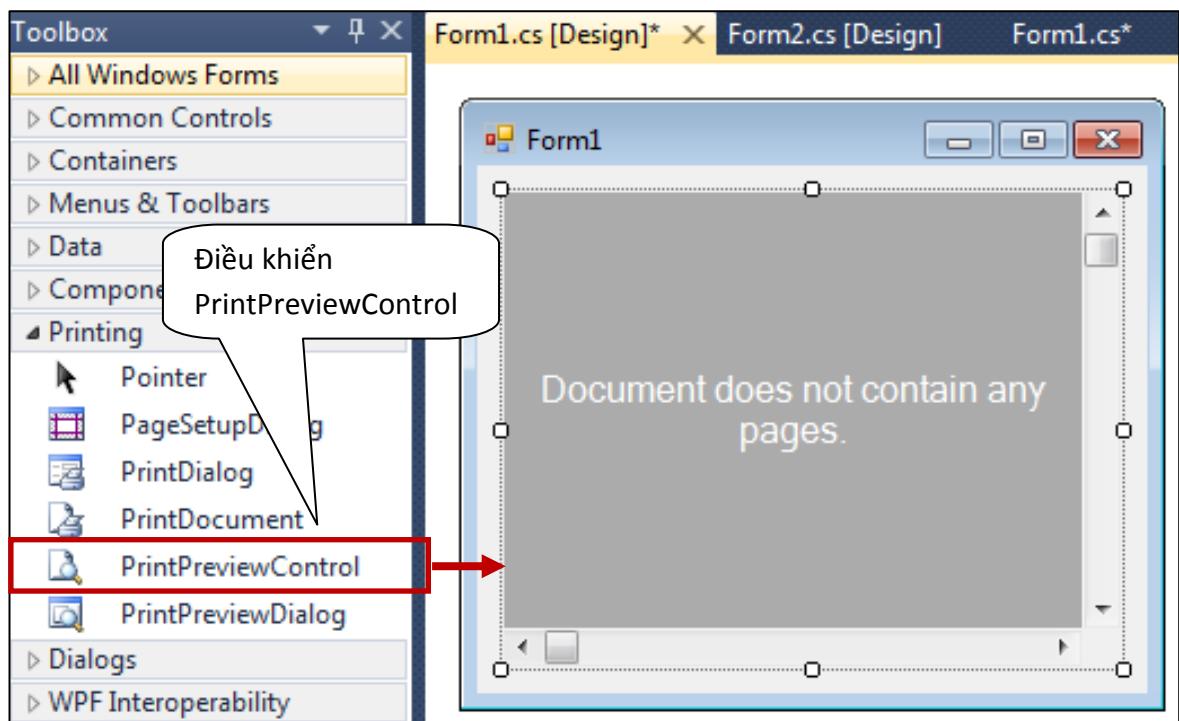
```
private void DrawText(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    string text = txtNoiDung.Text;
    System.Drawing.Font printFont = new
        System.Drawing.Font("Arial", 35,
    System.Drawing.FontStyle.Regular);
    e.Graphics.DrawString(text, printFont,
        System.Drawing.Brushes.Black, 0, 0);
}
```

- Viết mã lệnh cho sự kiện *Click* nút btnXemBanIn:

```
private void button1_Click(object sender, EventArgs e)
{
    printDocument1.PrintPage += DrawText;
    printPreviewDialog1.Document = printDocument1;
    printPreviewDialog1.ShowDialog();
}
```

### 8.3. Điều khiển PrintPreviewControl

Cũng như điều khiển *PrintPreviewDialog*, điều khiển *PrintPreviewControl* sử dụng để xem trước trang in trước khi in văn bản trên giấy bằng máy in. Tuy nhiên, điểm khác biệt là *PrintPreviewDialog* hiển thị ở dạng hộp thoại độc lập với form, còn *PrintPreviewControl* thì hiển thị trên form, do đó việc hiển thị *PrintPreviewControl* không cần phải sử dụng phương thức *Show()* hay phương thức *ShowDialog()*. Lập trình viên có thể kéo *PrintPreviewControl* từ cửa sổ Toolbox vào form và bố trí cạnh các điều khiển khác như *TextBox*, *Button*, *Label*, ... Điều khiển *PrintPreviewControl* nằm trong nhóm *Printing* của cửa sổ Toolbox như hình 8.9.



Hình 8.9: Điều khiển PrintPreviewControl trong cửa sổ Toolbox

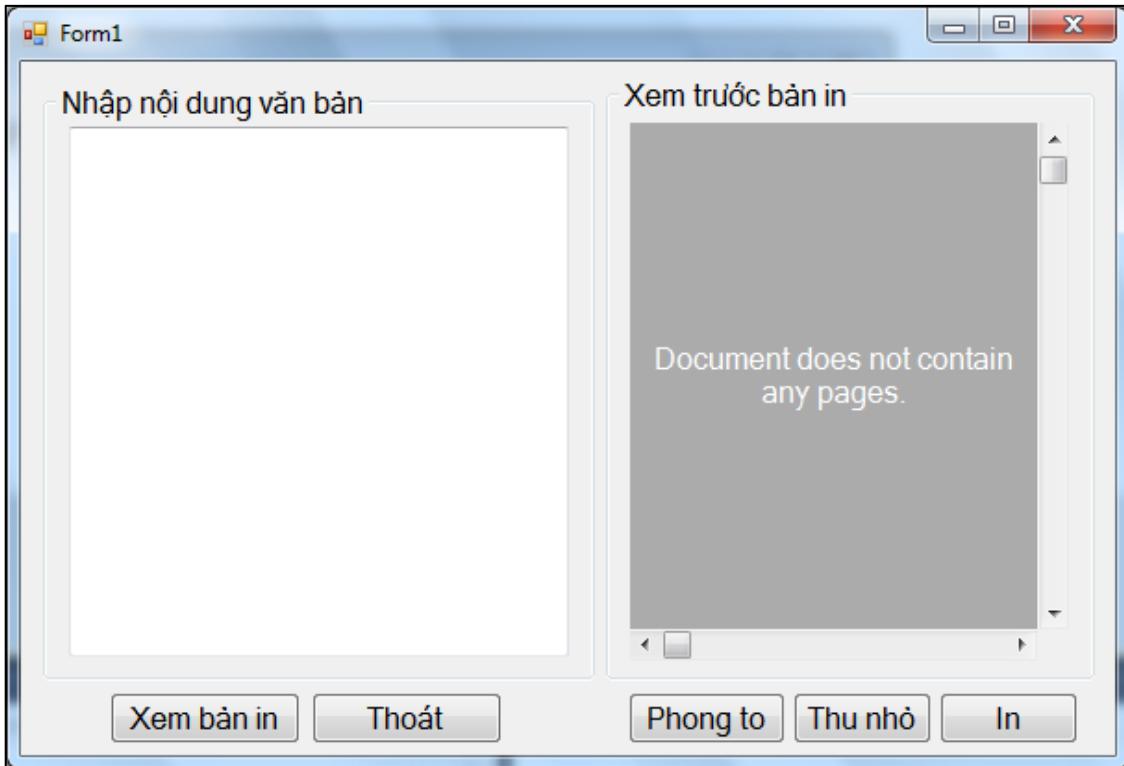
- Một số thuộc tính thường dùng của PrintPreviewControl:

Bảng 8.2: Bảng mô tả các thuộc tính của PrintPreviewControl

Thuộc tính	Mô tả
<i>AutoZoom</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép trang hiển thị tự động phóng to hay thu nhỏ vừa với điều khiển khi kích thước bị thay đổi.</li> <li>- Nếu là False: Kích thước trang sẽ cố định. Nếu kích thước điều khiển nhỏ hơn kích thước trang sẽ hiển thị thanh trượt.</li> </ul>
<i>Columns</i>	Thiết lập số trang hiển thị theo chiều ngang trên màn hình
<i>Document</i>	Thiết lập tài liệu sẽ được xem trước. Lưu ý: Giá trị thiết lập là một đối tượng thuộc lớp PrintDocument
<i>Rows</i>	Thiết lập số trang hiển thị theo chiều dọc trên màn hình.
<i>StartPage</i>	Thiết lập trang sẽ hiển thị đầu tiên khi xem trước trang in.
<i>UseAntiAlias</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép người dùng loại bỏ răng</li> </ul>

	cua trên trang hiển thị. Giúp trang hiển thị trông mượt hơn. Tuy nhiên, sử dụng chế độ này sẽ làm cho chương trình bị chậm. - Nếu là False: Không sử dụng chế độ loại bỏ răng cưa.
Zoom	Thiết lập kích thước hiển thị của trang.

Ví dụ 8.2: Viết chương trình có giao diện như hình 8.10. Chương trình bao gồm các điều khiển TextBox, GroupBox, Button, PrintDocument, PrintPreviewControl



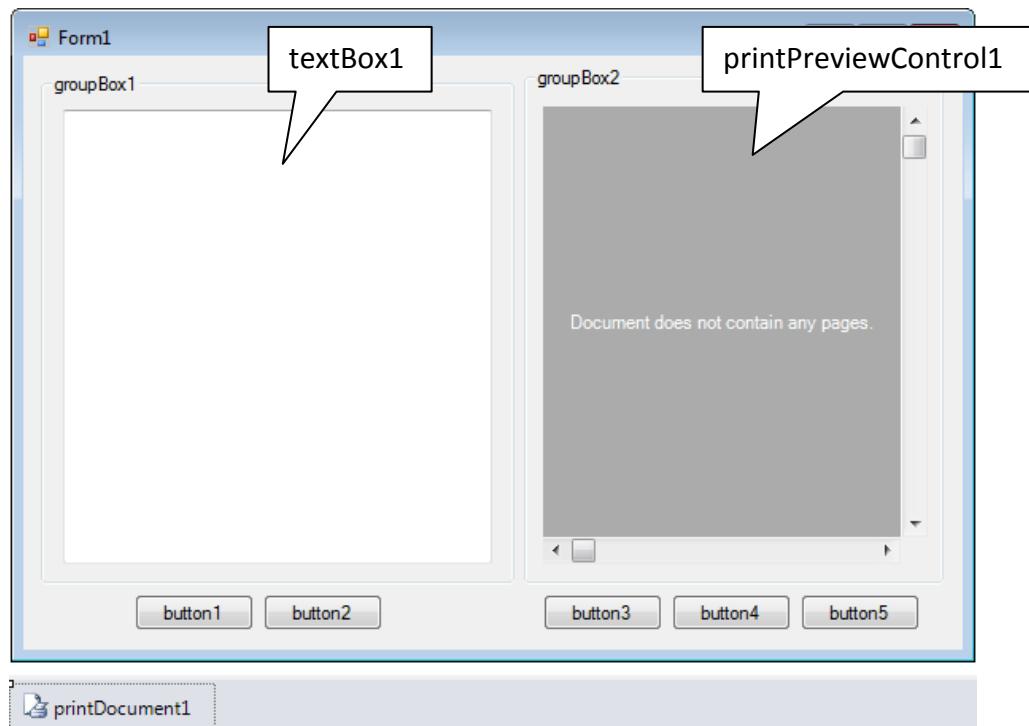
Hình 8.10: Giao diện chương trình xem trước trang in bằng PrintPreviewControl

Yêu cầu:

Người dùng nhập đoạn văn bản vào TextBox. Khi nhấn nút “Xem bản in” thì sẽ hiển thị trước trang in trên PrintPreviewControl. Ngoài ra người dùng có thể sử dụng nút “Phóng to” và nút “Thu nhỏ” để thay đổi kích thước trang hiển thị. Sử dụng nút “In” để in trang hiển thị ra giấy trên máy in.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển TextBox, GroupBox, Button, PrintDocument, PrintPreviewControl từ Toolbox vào form như hình 8.11.



Hình 8.11: Giao diện thiết kế form khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho các điều khiển trong cửa sổ Properties

- Form1:
  - Thuộc tính Size: 607, 412
- groupBox1:
  - Thuộc tính Text: “Nhập nội dung văn bản”
- groupBox2:
  - Thuộc tính Text: “Xem trước bản in”
- textBox1:
  - Thuộc tính Name: txtVanBan
  - Thuộc tính Multiline: True
  - Thuộc tính Size: 270, 286
- printPreviewControl1:
  - Thuộc tính AutoZoom: True
  - Thuộc tính Document: printDocument1
- button1:
  - Thuộc tính Name: btnXemBanIn
  - Thuộc tính Text: “Xem bản in”
- button2:
  - Thuộc tính Name: btnThoat
  - Thuộc tính Text: “Thoát”
- button3:
  - Thuộc tính Name: btnPhongTo

Thuộc tính Text: “Phóng to”

- button4:

Thuộc tính Name: btnThuNho

Thuộc tính Text: “Thu nhỏ”

- button5:

Thuộc tính Name: btnIn

Thuộc tính Text: “In”

Bước 3: Viết mã lệnh cho các điều khiển

- Xây dựng hàm DrawText:

```
private void DrawText(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    string text = txtVanBan.Text;
    System.Drawing.Font printFont = new
        System.Drawing.Font("Arial", 35,
    System.Drawing.FontStyle.Regular);
    e.Graphics.DrawString(text, printFont,
        System.Drawing.Brushes.Black, 0, 0);
}
```

- Sự kiện Click nút btnXemBanIn:

```
private void btnXemBanIn_Click(object sender, EventArgs e)
{
    PreView();
}
```

- Xây dựng hàm PreView:

```
private void PreView()
{
    printDocument1.PrintPage += new
        System.Drawing.Printing.PrintPageEventHandler(DrawText);
    printPreviewControl1.Document = printDocument1;
}
```

- Sự kiện Click nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện Click nút btnIn:

```
private void btnIn_Click(object sender, EventArgs e)
{
    printDocument1.Print();
}
```

- Sự kiện Click nút btnPhongTo:

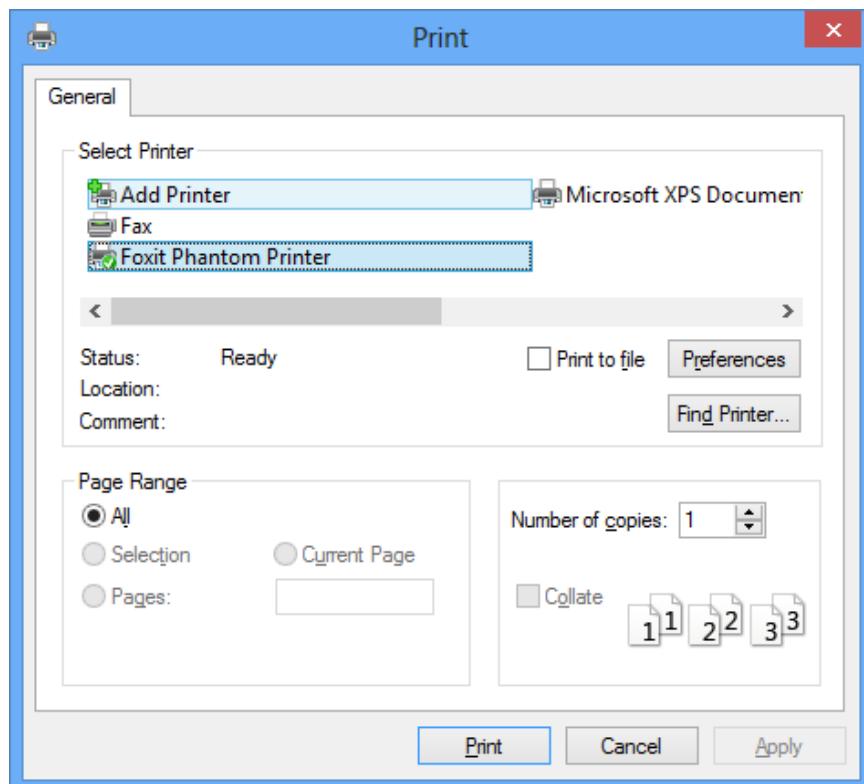
```
private void btnPhongTo_Click(object sender, EventArgs e)
{
    printPreviewControl1.Zoom += 0.01;
    PreView();
}
```

- Sự kiện Click nút btnThuNho:

```
private void btnThuNho_Click(object sender, EventArgs e)
{
    printPreviewControl1.Zoom -= 0.01;
    PreView();
}
```

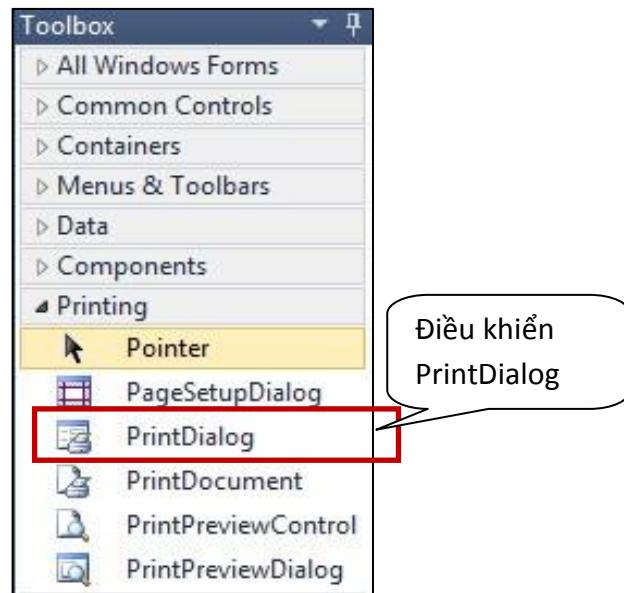
## 8.4. Điều khiển PrintDialog

Điều khiển PrintDialog cho phép hiển thị dạng hộp thoại như hình 8.12 để người dùng chọn máy in và in văn bản. Cũng như các điều khiển thực hiện công việc in ấn khác, hộp thoại PrintDialog được hiển thị bằng phương thức ShowDialog() nhưng trước khi gọi phương thức ShowDialog() lập trình viên cần thiết lập giá trị cho thuộc tính Document.



Hình 8.12: Giao diện hộp thoại PrintDialog

Điều khiển *PrintDialog* nằm trong nhóm Printing của cửa sổ Toolbox như hình 8.13.

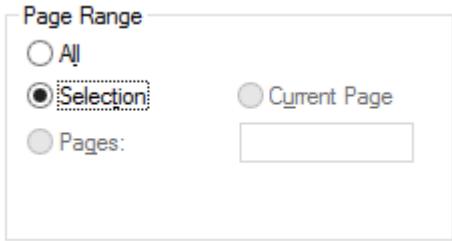
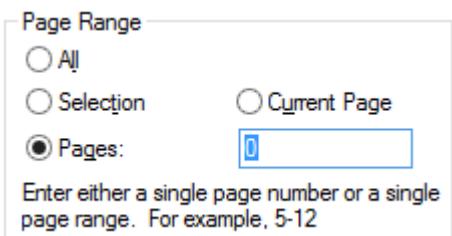


Hình 8.13: Điều khiển *PrintDialog* trong cửa sổ Toolbox

- Một số thuộc tính thường dùng của *PrintDialog*:

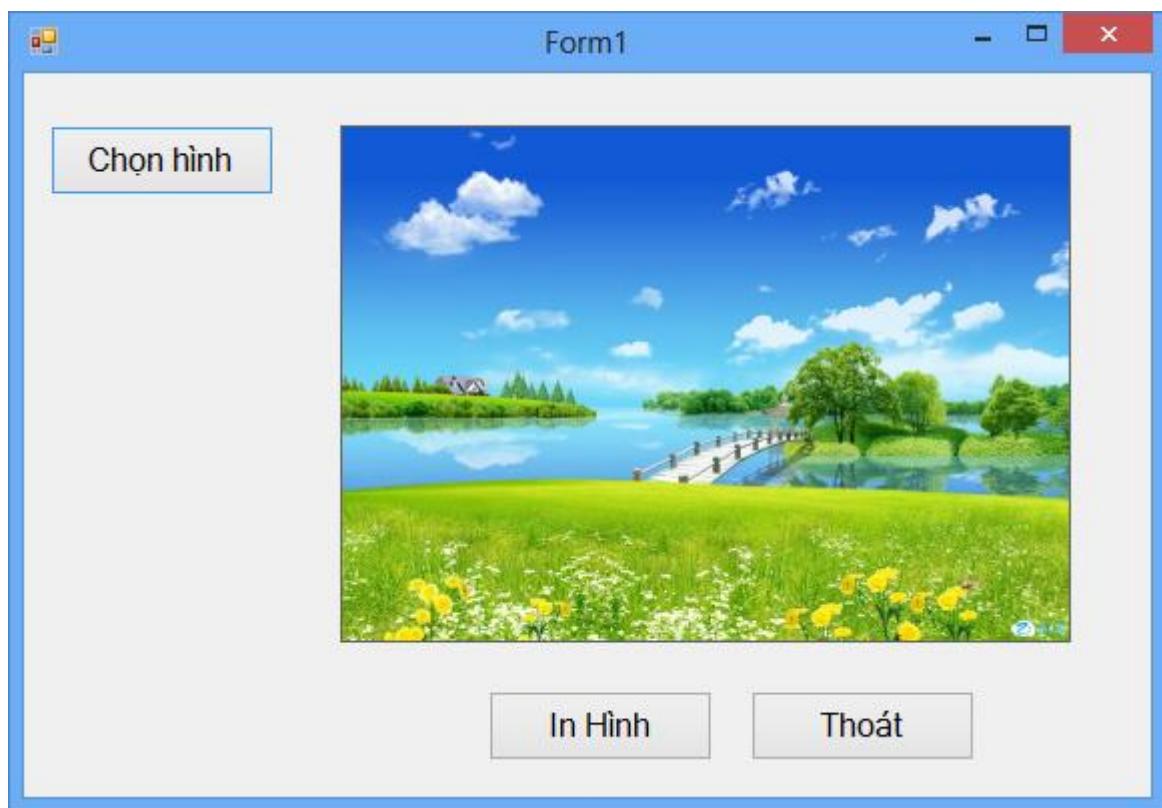
Bảng 8.2: Bảng mô tả các thuộc tính của *PrintDialog*

Thuộc tính	Mô tả
<i>AllowCurrentPage</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép chọn trang hiện hành để in.</li> <li>- Nếu là False: Không cho phép in chọn in trang hiện hành.</li> </ul>
<i>AllowPrintToFile</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép hiển thị Checkbox <input type="checkbox"/> <b>Print to file</b>, người dùng có thể in ra thành file thay vì in ra giấy trên máy in</li> <li>- Nếu là False: Không hiển thị Checkbox <input type="checkbox"/> <b>Print to file</b></li> </ul>
<i>AllowSelection</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Cho phép hiển thị RadioButton Selection để người dùng chọn trang sẽ in.</li> </ul>

	 <ul style="list-style-type: none"> <li>Nếu là False: Không hiển thị RadioButton Selection</li> </ul>
<i>AllowSomePages</i>	<p>Mang hai giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>Nếu là True: Cho phép hiển thị RadioButton Pages để người dùng chọn in trang thứ i</li> </ul>  <ul style="list-style-type: none"> <li>Nếu là False: Không hiển thị RadioButton Pages</li> </ul>
<i>Document</i>	<p>Thiết lập tài liệu sẽ được in.</p> <p>Lưu ý: Giá trị thiết lập là một đối tượng thuộc lớp PrintDocument</p>
<i>PrintToFile</i>	<p>Mang hai giá trị True hoặc False.</p> <ul style="list-style-type: none"> <li>Nếu là True: Checkbox PrintToFile trên hộp thoại được chọn.</li> <li>Nếu là False: Checkbox PrintToFile trên hộp thoại không được chọn.</li> </ul>
<i>PrintSettings</i>	<p>Thuộc tính PrintSettings có nhiều tùy chọn bên trong giúp thiết lập các công việc như:</p> <ul style="list-style-type: none"> <li>PrintSettings.PrinterName = &lt;Tên máy in&gt;: Chọn máy in</li> <li>PrintSettings.PrintToFile = &lt;True/ False&gt;: Cho phép hoặc không cho phép in ra tập tin.</li> <li>PrintSettings.FromPage = &lt;Số nguyên&gt;: Thiết lập bắt đầu in từ trang thứ mấy.</li> <li>PrintSettings.ToPage = &lt;Số nguyên&gt;: Thiết lập trang cuối cùng sẽ in</li> <li>PrintSettings.Size: Lấy kích cỡ giấy mà máy in hỗ</li> </ul>

	<p>trợ.</p> <ul style="list-style-type: none"> <li>- PrintSettings.Resolutions: Lấy tất cả độ phân giải mà máy in hỗ trợ.</li> <li>- PrintSettings.Duplex = &lt;True/ False&gt;: Thiết lập chế độ in một mặt hay hai mặt của trang giấy.</li> <li>- PrintSettings.CanDuplex: Trả về giá trị True hoặc False, cho biết máy in có hỗ trợ in hai mặt của trang giấy hay không.</li> </ul>
--	--

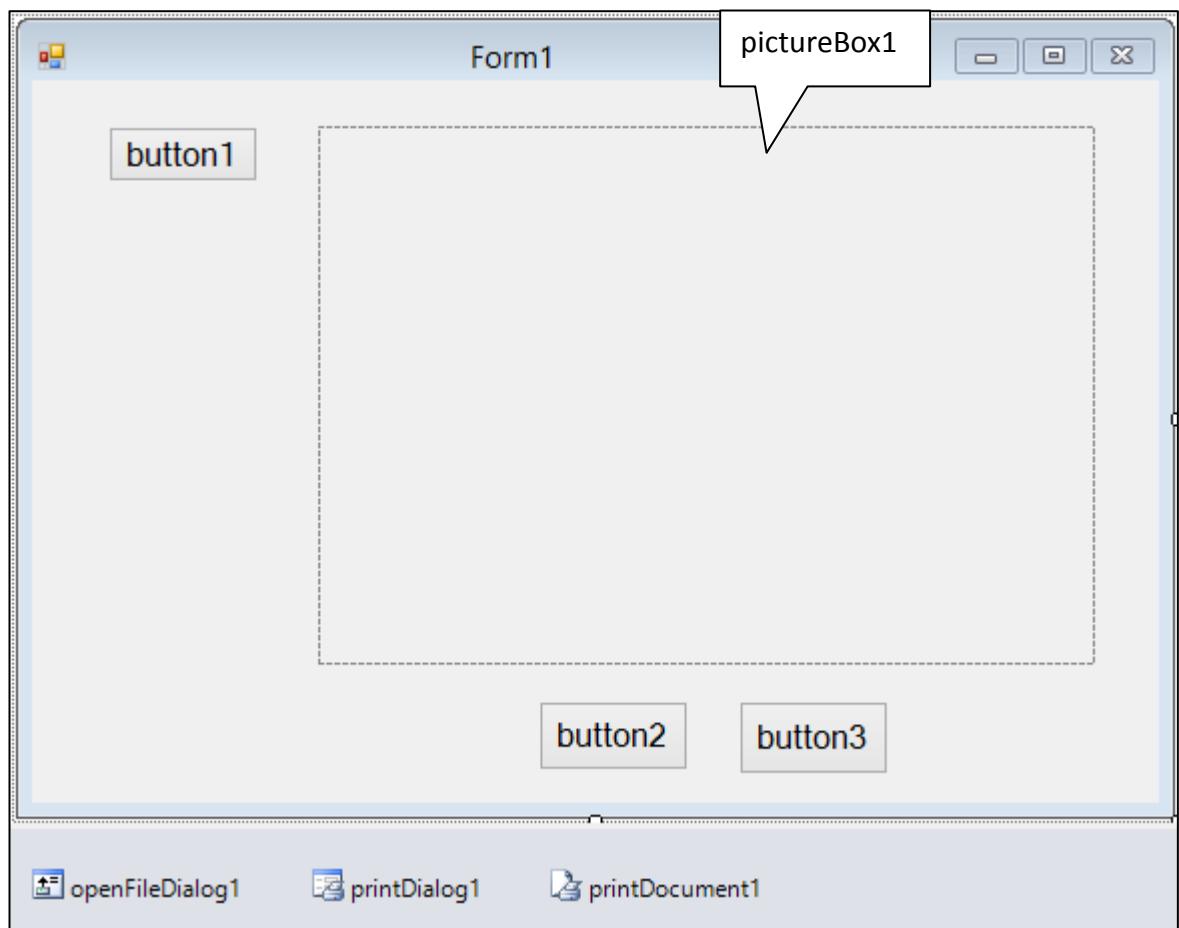
Ví dụ 8.3: Viết chương trình in hình ảnh. Chương trình có giao diện như hình 8.14 cho phép người dùng chọn 1 hình ảnh trên máy tính và hiển thị trên PictureBox. Người dùng có thể sử dụng Button “In Hình” để gọi hộp thoại PrintDialog và in hình ra giấy bằng máy in.



Hình 8.14: Giao diện chương trình in hình ảnh

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển Button, PictureBox, PrintDialog, PrintDocument và OpenFileDialog từ cửa sổ Toolbox và form như hình 8.15.



Hình 8.15: Giao diện ban đầu form in hình ảnh

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties.

- button1:
  - Thuộc tính *Name*: btnChonHinh
  - Thuộc tính *Text*: “Chọn hình”
- button2:
  - Thuộc tính *Name*: btnIn
  - Thuộc tính *Text*: “In hình”
- button3:
  - Thuộc tính *Name*: btnThoat
  - Thuộc tính *Text*: “Thoát”
- pictureBox1:
  - Thuộc tính *BorderStyle*: FixedSingle
  - Thuộc tính *SizeMode*: StretchImage
- openFileDialog1:
  - Thuộc tính *Filter*: “jpg images|\*.jpg”
- PrintDialog:
  - Thuộc tính *AllowCurrentPage*: True
  - Thuộc tính *AllowPrintToFile*: True

Thuộc tính *AllowSelection*: True  
Thuộc tính *AllowSomePages*: True  
Thuộc tính *Document*: printDocument1  
Thuộc tính *PrintToFile*: True  
Thuộc tính *ShowHelp*: True  
Thuộc tính *ShowNetwork*: True

Bước 3: Viết mã lệnh cho các điều khiển

- Viết hàm printPicture\_PrintPage:

```
void printPicture_PrintPage(object sender, PrintPageEventArgs e)
{
    try
    {
        if (pictureBox1.Image != null)
        {
            Bitmap bitmap = new Bitmap(pictureBox1.Image);
            if (bitmap != null)
            {
                e.Graphics.DrawImage(bitmap, 10, 10);
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}
```

- Sự kiện *Click* nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện *Click* nút btnChonHinh:

```
private void btnChonHinh_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.ImageLocation = openFileDialog1.FileName;
    }
}
```

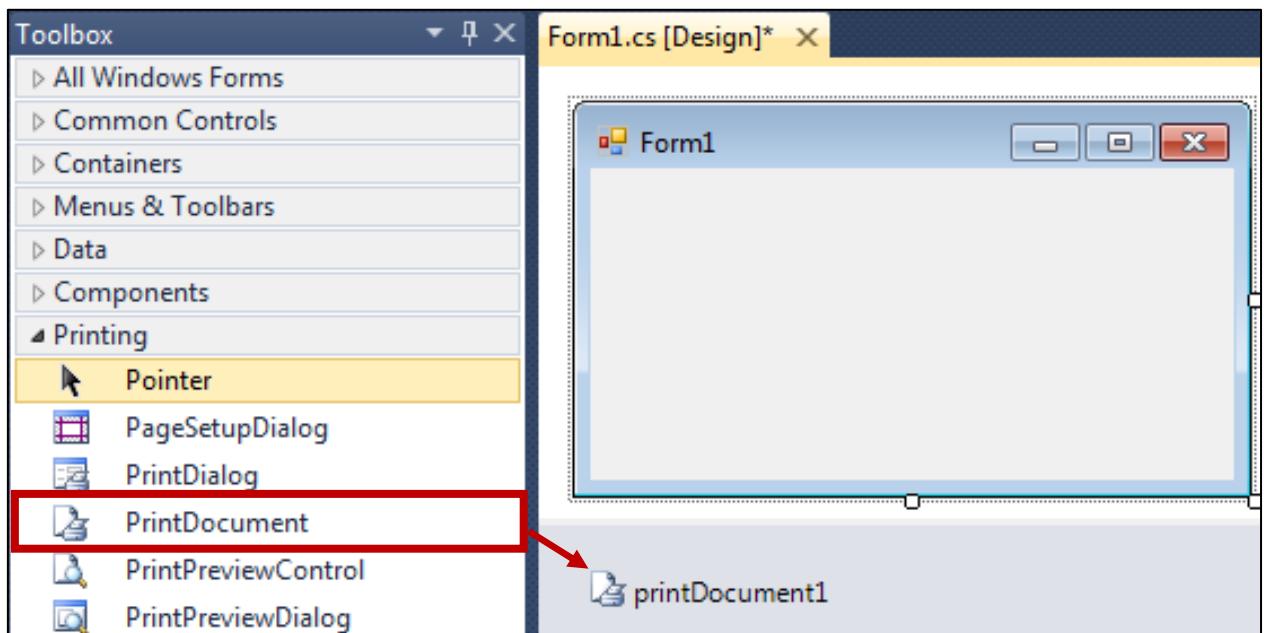
- Sự kiện Click nút btnIn:

```
private void btnIn_Click(object sender, EventArgs e)
{
    printDocument1.PrintPage += new
        PrintPageEventHandler(printPicture_PrintPage);
    printDialog1.Document = printDocument1;
    if (printDialog1.ShowDialog() == DialogResult.OK)
    {
        printDocument1.PrinterSettings =
            printDialog1.PrinterSettings;
        printDocument1.Print();
    }
}
```

## 8.5. Điều khiển PrintDocument

Lớp PrintDocument có không gian tên là System.Drawing.Printing. PrintDocument là lớp thường xuyên llop nắm giữ nội dung cần in ra tập tin hay máy in, do đó PrintDocument được sử dụng nhiều trong việc in ấn.

Đối tượng tạo ra từ PrintDocument được sử dụng cùng với các điều khiển PrintDialog, PrintPreviewDialog, PrintPreviewControl, ... Lập trình viên có thể tạo ra đối tượng PrintDocument bằng cách kéo điều khiển PrintDocument từ cửa sổ Toolbox vào form như hình 8.16.



Hình 8.16: Điều khiển PrintDocument trong cửa sổ Toolbox

Hoặc có thể tạo đối tượng PrintDocument bằng mã lệnh như sau:

```
PrintDocument prtDC= new PrintDocument();
```

➤ Sử dụng điều khiển PrintDocument:

Trong công việc in ấn, để in nội dung mà PrintDocument nắm giữ ra máy in hoặc ra tập tin, lập trình viên sử dụng phương thức Print() của lớp PrintDocument:

```
prtDC.Print();
```

Khi gọi phương thức Print thì sự kiện PrintPage được phát sinh để tiến hành vẽ các nội dung PrintDocument nắm giữ trên các trang giấy. Như vậy, lập trình viên cần phải định nghĩa các công việc cần phải làm khi sự kiện PrintPage phát sinh, thông thường công việc này là vẽ các nội dung PrintDocument nắm giữ lên trang giấy. Để đơn giản lập trình viên sẽ xây dựng một hàm độc lập thực hiện công việc vẽ này và gọi hàm thực hiện trong sự kiện PrintPage.

- Hàm vẽ văn bản:

```
private void DrawText(object sender,  
System.Drawing.Printing.PrintPageEventArgs e)  
{  
    string text = "Khoa Công nghệ thông tin";  
    System.Drawing.Font printFont = new  
        System.Drawing.Font("Arial", 35,  
        System.Drawing.FontStyle.Regular);  
    e.Graphics.DrawString(text, printFont,  
        System.Drawing.Brushes.Black, 0, 0);  
}
```

- Hàm vẽ hình chứa trong pictureBox1 ra trang giấy:

```
private void DrawPicture(object sender,  
System.Drawing.Printing.PrintPageEventArgs e)  
{  
    try  
    {  
        if (pictureBox1.Image != null)  
        {  
            Bitmap bitmap = new Bitmap(pictureBox1.Image);  
            if (bitmap != null)  
            {  
                e.Graphics.DrawImage(bitmap, 10, 10);  
            }  
        }  
    }  
    catch (Exception ex)  
    {  
        MessageBox.Show(ex.Message);  
    }  
}
```

- Thực hiện gọi hàm vẽ văn bản hoặc vẽ hình ảnh lên trang giấy in hoặc lên tập tin trong sự kiện PrintPage:

```

private void btnIn_Click(object sender, EventArgs e)
{
    //khai báo đối tượng lớp PrintDocument
    PrintDocument printDC = new PrintDocument();
    //Sử dụng hàm vẽ văn bản trong sự kiện PrintPage. Nếu muốn vẽ
    //hình thì thay DrawText bằng DrawPicture
    printDC.PrintPage += new
        PrintPageEventHandler(DrawText);
    //Khai báo đối tượng lớp PrintPreviewDialog
    PrintPreviewDialog printPrev = new PrintPreviewDialog();
    printPrev.Document = printDC;
    //Hiển thị hộp thoại PrintPreviewDialog để xem trước trang in
    if (printPrev.ShowDialog() == DialogResult.OK)
    {
        //Nếu người dùng nhấn nút in thì phương thức Print sẽ
        //thực hiện và khi đó sẽ phát sinh sự kiện PrintPage thực
        //hiện công việc vẽ văn bản hoặc vẽ hình ảnh
        printDC.Print();
    }
}

```

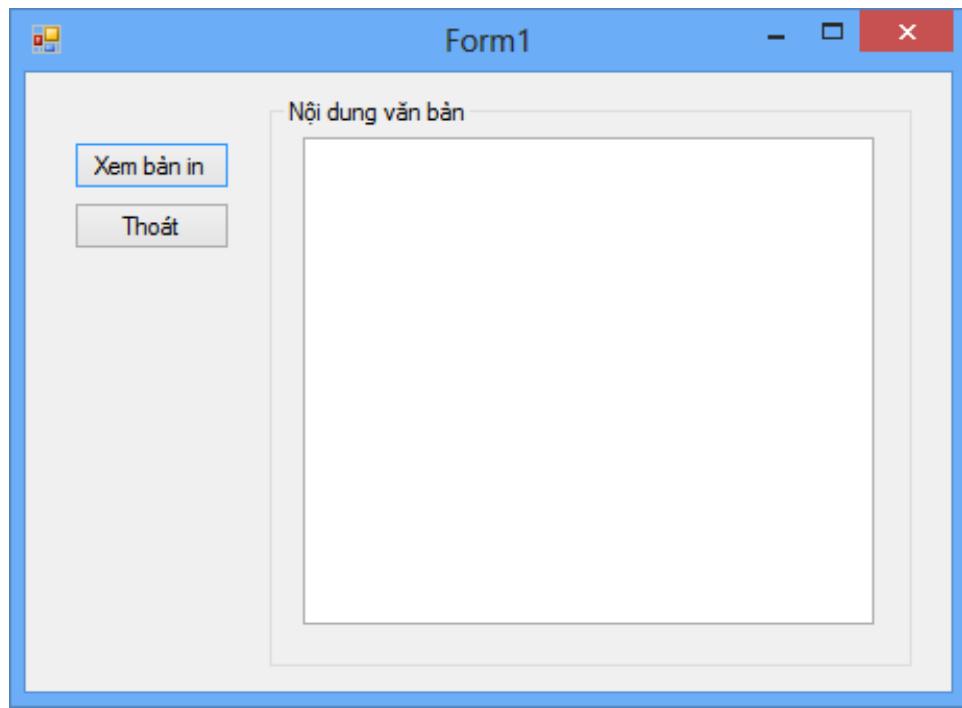
➤ In nội dung trên nhiều trang:

Có vấn đề phát sinh với việc in ấn khi nội dung cần in quá nhiều, nhiều hơn kích thước của một trang giấy và nếu xử lý bằng những câu lệnh trên thì người dùng sẽ chỉ in được một trang. Những nội dung vượt quá kích thước trang sẽ bị mất đi. Vấn đề này xảy ra bởi vì PrintDocument không có ký tự ngắt dòng và cũng không thể tự động ngắt trang. Do đó, cần phải viết mã lệnh thực hiện các công việc ngắt dòng và ngắt trang này.

Để có thể ngắt dòng in, lập trình viên cần biết được kích thước dòng và tính ra số ký tự tối đa có thể in trên một dòng của trang. Tương tự, muốn ngắt trang thì lập trình viên cũng phải tính toán được kích thước trang từ đó lấy ra thông tin số dòng có thể in trên một trang giấy.

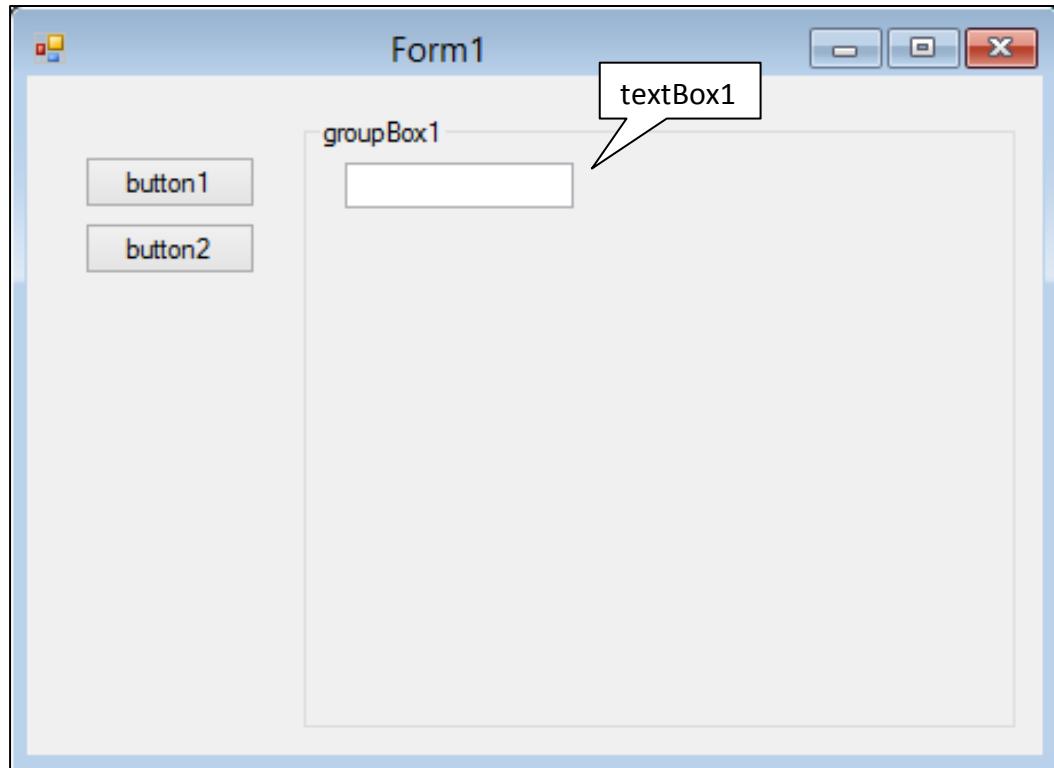
Ví dụ 8.4: Viết chương trình in văn bản có giao diện như hình 8.17. Chương trình gồm các điều khiển Button, TextBox, GroupBox.

Yêu cầu: Người dùng nhập văn bản vào TextBox. Khi nhấn nút “Xem bản in” sẽ hiển thị trước trang in. Trang hiển thị sẽ tự động ngắt dòng xuống hàng và sang trang mới nếu số lượng ký tự của văn bản vượt kích thước trang in.



Hình 8.17: Giao diện chương trình in văn bản

Bước 1: Thiết kế giao diện ban đầu cho điều khiển: Thêm các điều khiển Button, TextBox, GroupBox trong cửa sổ Toolbox vào form như hình 8.18.



Hình 8.18: Giao diện form sau khi thêm điều khiển

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties.

- button1:

Thuộc tính Name: btnIn

Thuộc tính Text: “Xem bản in”

- button2:  
Thuộc tính Name: btnThoat  
Thuộc tính Text: “Thoát”
- textBox1:  
Thuộc tính Name: txtVanBan  
Thuộc tính Multiline: True  
Thuộc tính Size: 274, 234
- groupBox1:  
Thuộc tính Text: “Nội dung văn bản”  
Thuộc tính Size: 308, 273

Bước 3: Viết mã lệnh cho các điều khiển

- Khai báo biến:

```
string stringtoPrint = "";
```

- Sự kiện Click nút btnIn:

```
private void btnIn_Click(object sender, EventArgs e)
{
    stringtoPrint = txtVanBan.Text;
    PrintDocument printDC = new PrintDocument();
    printDC.PrintPage += new PrintPageEventHandler(DrawText);
    PrintPreviewDialog printPrev = new PrintPreviewDialog();
    printPrev.Document = printDC;
    if (printPrev.ShowDialog() == DialogResult.OK)
    {
        printDC .Print();
    }
}
```

- Sự kiện Click nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Mã lệnh hàm DrawText:

```
private void DrawText(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
{
    int sokytu, sodong;
    string chuoiin;
    //khai báo font chữ
    System.Drawing.Font printFont = new
        System.Drawing.Font("Arial", 35,
        System.Drawing.FontStyle.Regular);
    //khai báo chuỗi định dạng
    StringFormat chuoidinhdang=new StringFormat();
    //khai báo vùng sẽ in trên trang
    System.Drawing.RectangleF vungin = new
        RectangleF(e.MarginBounds.Left,
        e.MarginBounds.Top, e.MarginBounds.Width,
        e.MarginBounds.Height);
    //khai báo kích thước
    SizeF kichthuoc = new SizeF(e.MarginBounds.Width,
        e.MarginBounds.Height - printFont.GetHeight(e.Graphics));
    chuoidinhdang.Trimming = StringTrimming.Word;
    //Sử dụng phương thức MeasureString để lấy số ký tự trên một
    //đòng và số lượng dòng văn bản
    e.Graphics.MeasureString(stringtoPrint, printFont, kichthuoc,
        chuoidinhdang,out sokytu,out sodong);
    //Trích các ký tự sẽ hiển thị trên một dòng
    chuoiin = stringtoPrint.Substring(0, sokytu);
    e.Graphics.DrawString(chuoiin, printFont,
        Brushes.Black, vungin, chuoidinhdang);
    if (sokytu < stringtoPrint.Length)
    {
        stringtoPrint = stringtoPrint.Substring(sokytu);
        e.HasMorePages = true;
    }
    else
    {
        e.HasMorePages = false;
        stringtoPrint = txtVanBan.Text;
    }
}
```

# CHƯƠNG 9: ĐIỀU KHIỂN NGƯỜI DÙNG TỰ TẠO

## 9.1. User Control

User Control là điều khiển do lập trình viên tạo ra theo nhu cầu sử dụng riêng. Trong nhiều trường hợp, lập trình viên đã thiết kế xong một điều khiển hay một giao diện chương trình gồm nhiều điều khiển, và nếu có nhu cầu sử dụng nhiều lần hoặc mang sang một dự án khác sử dụng thì lập trình viên sẽ chuyển những điều khiển hay nhóm điều khiển đó về dạng User Control.

Một cách cụ thể hơn User Control là điều khiển được tạo ra trên cơ sở các điều khiển mà Windows Form hỗ trợ sẵn như: TextBox, Label, Button, ...

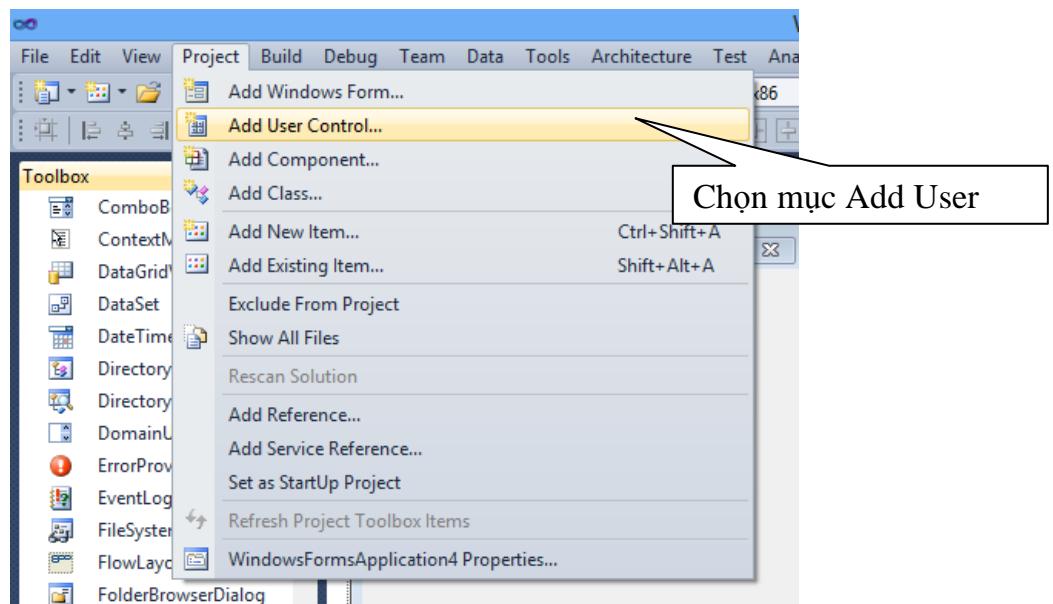
Việc sử dụng User Control như vậy sẽ giúp tiết kiệm chi phí thời gian khi thiết kế giao diện chương trình. Lập trình viên có thể tạo ra User Control bằng các cách như sau:

- Thêm mới User Control ngay trong dự án.
- Tạo lớp và khai báo thừa kế từ lớp Control.
- Tạo dự án mới bằng cách chọn loại dự án là Windows Control Library.

## 9.2. Xây dựng User Control

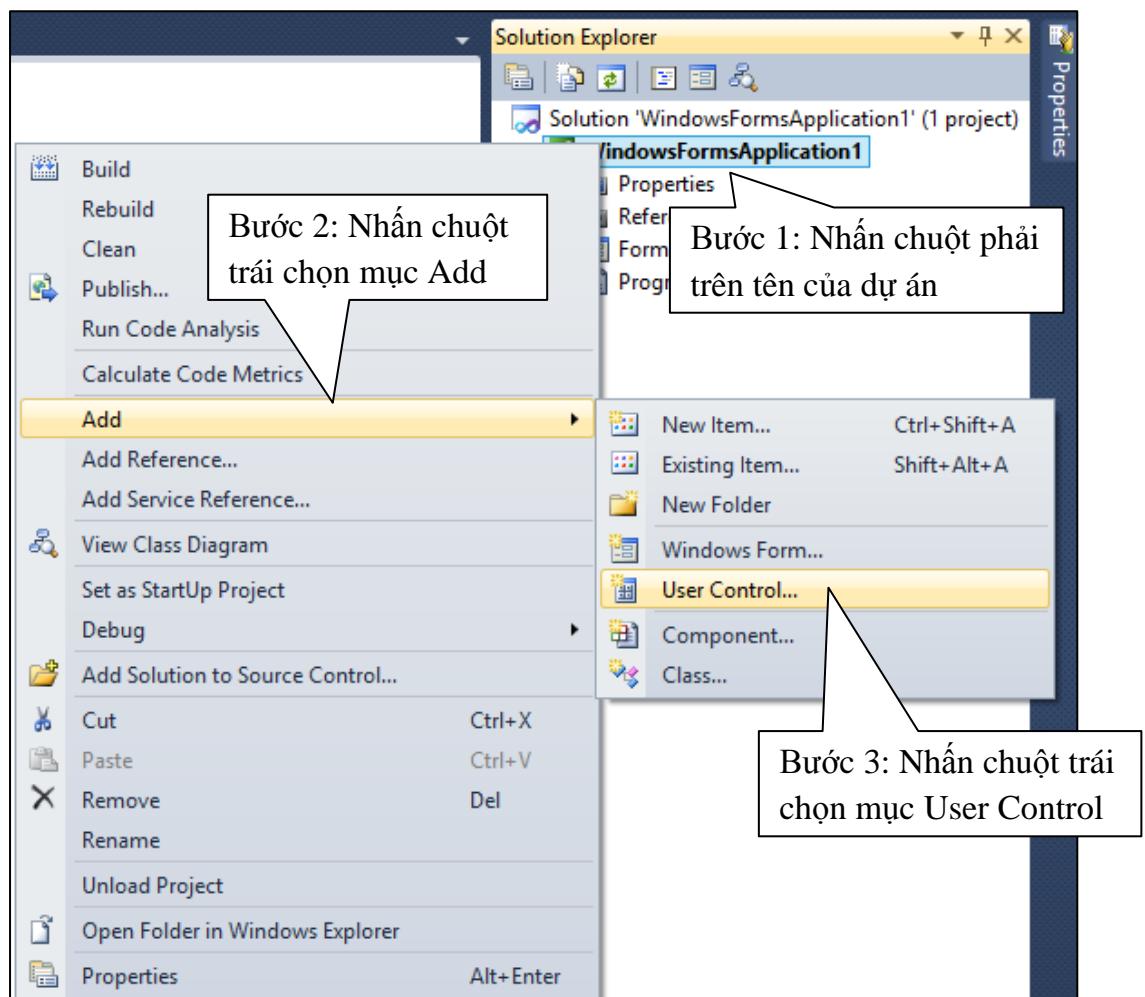
### 9.2.1. Thêm mới User Control ngay trong dự án

Lập trình viên thêm mới User Control trên thanh thực đơn như hình 9.1.



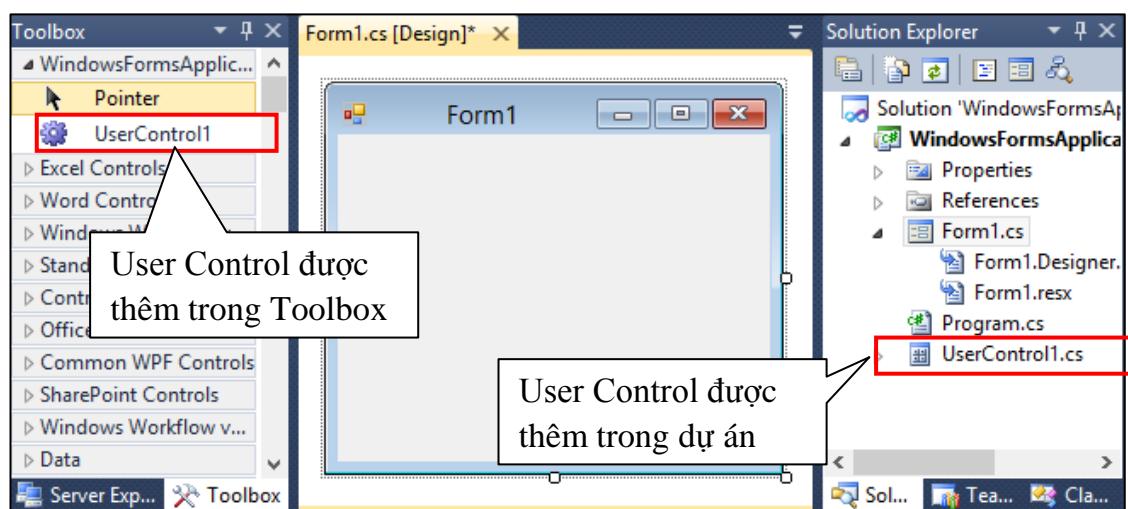
Hình 9.1: Thêm User Control từ thực đơn Project

Hoặc thêm mới User Control từ cửa sổ Solution Explorer như hình 9.2.



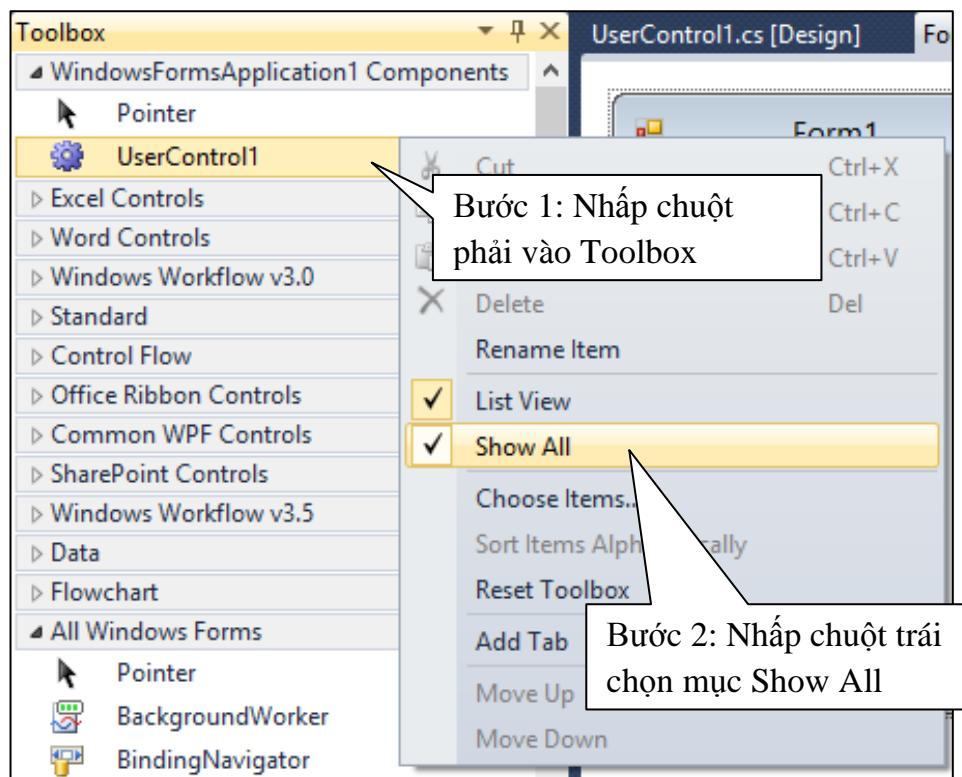
Hình 9.2: Thêm mới User Control từ cửa sổ Solution Explorer

Sau khi lập trình viên thêm mới một User Control vào dự án thì bên cửa sổ Toolbox sẽ hiển thị thêm một điều khiển là User Control vừa thêm trong nhóm UserControlInProject Components như hình 9.3.



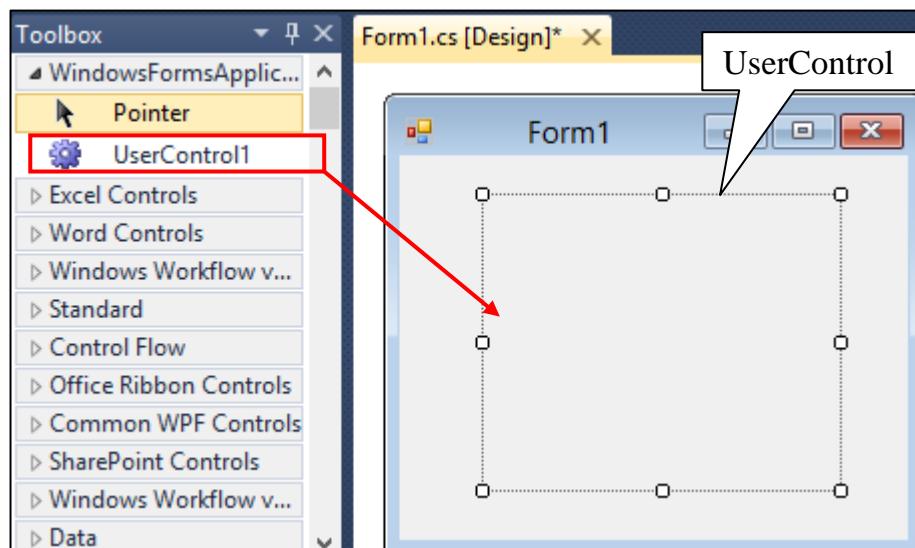
Hình 9.3: User Control mới được hiển thị trên cửa sổ Toolbox

Nếu sau khi thêm mới User Control mà không thấy hiển thị trên cửa sổ Toolbox có thể xử lý bằng cách nhấp phải vào Toolbox và chọn mục Show All như hình 9.4.



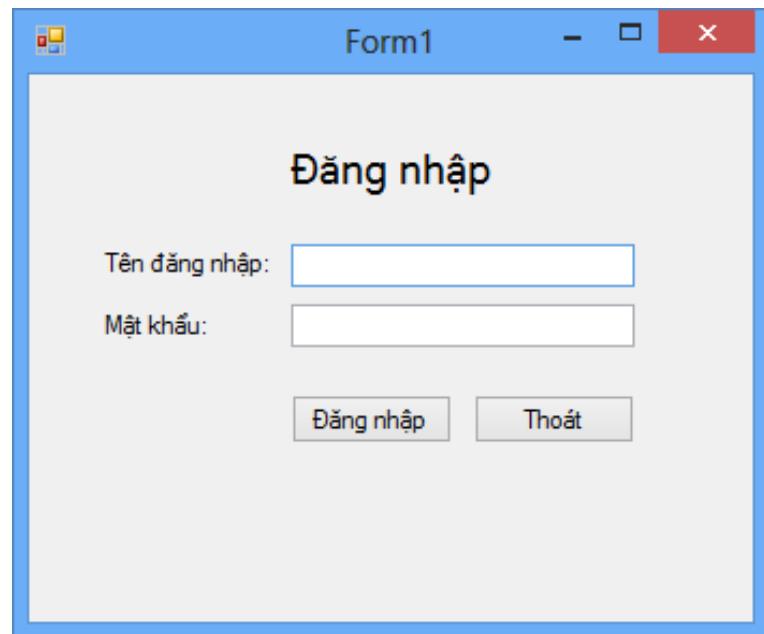
Hình 9.4: Chọn chức năng hiển thị tất cả điều khiển trên Toolbox

Sau khi đã thêm User Control, lập trình viên có thể sử dụng User Control bằng cách kéo User Control trên Toolbox vào form như hình 9.5.



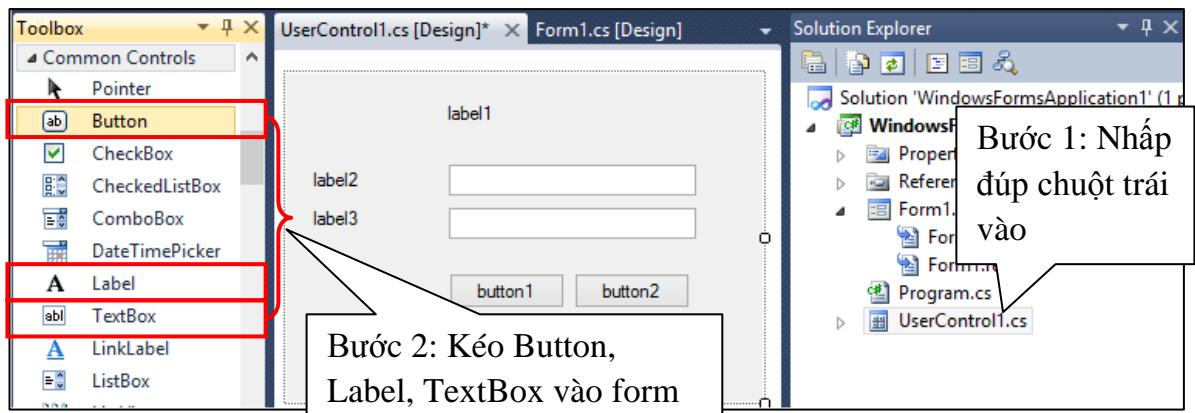
Hình 9.5: Giao diện form sau khi thả User Control vào

Ví dụ: Tạo UserControl Đăng nhập như hình 9.6. Sau đó kéo thả UserControl vào form.



Hình 9.6: Form đăng nhập

Bước 1: Thêm User Control có tên UserControl1 vào form. Sau đó thêm các điều khiển Label, TextBox và Button vào UserControl1 như hình 9.7.



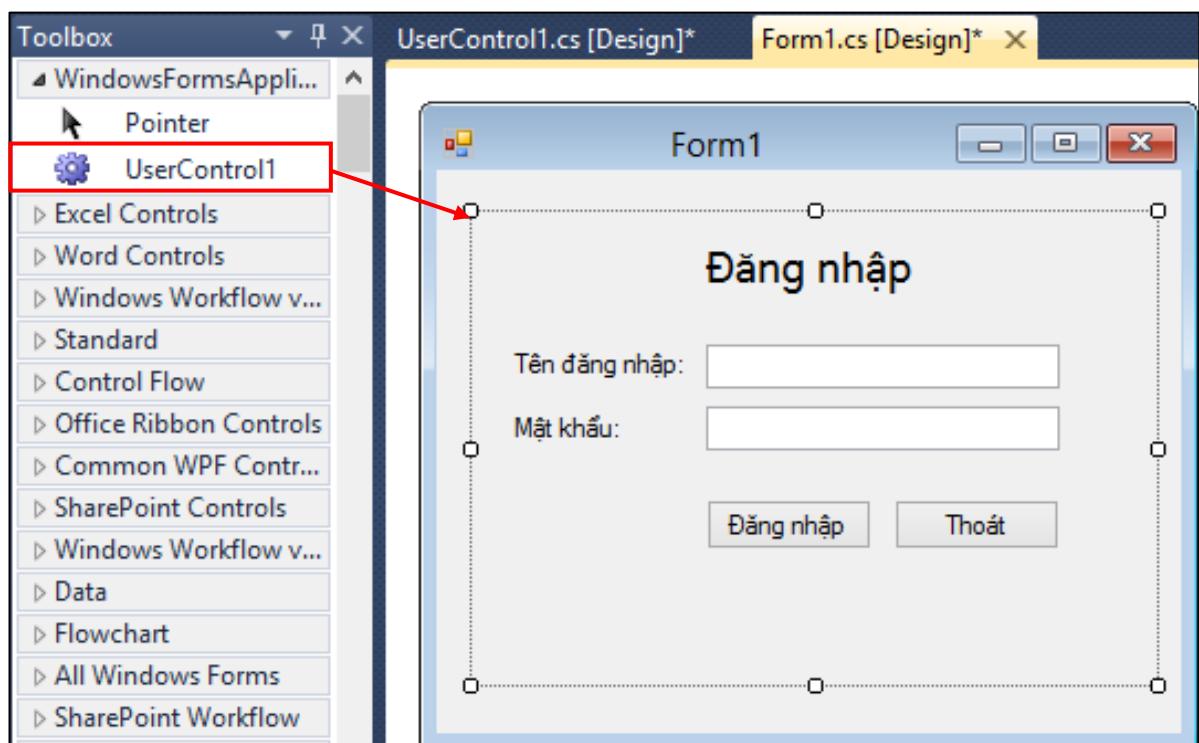
Hình 9.7: Giao diện UserControl1 sau khi thêm điều khiển.

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trên UserControl1 trong cửa sổ Properties.

- label1:  
Thuộc tính *Text*: “Đăng nhập”  
Thuộc tính *Font Size*: 14
- label2:  
Thuộc tính *Text*: “Tên đăng nhập.”
- label3:  
Thuộc tính *Text*: “Mật khẩu”
- textBox1:  
Thuộc tính *Name*: txtTen
- textBox2:

- Thuộc tính *Name*: txtMatKhau
- button1:
  - Thuộc tính *Name*: btnDangNhap
  - Thuộc tính *Text*: “Đăng nhập”
- button2:
  - Thuộc tính *Name*: btnThoat
  - Thuộc tính *Text*: “Thoát”

Bước 3: Khi đã thiết kế xong UserControl1, tiến hành biên dịch chương trình để UserControl1 cập nhật các thay đổi. Sau đó lập trình viên kéo UserControl1 từ cửa sổ Toolbox vào form sẽ được giao diện đăng nhập như hình 9.8



Hình 9.8: Giao diện form sau khi kéo điều khiển UserControl1 thả vào form

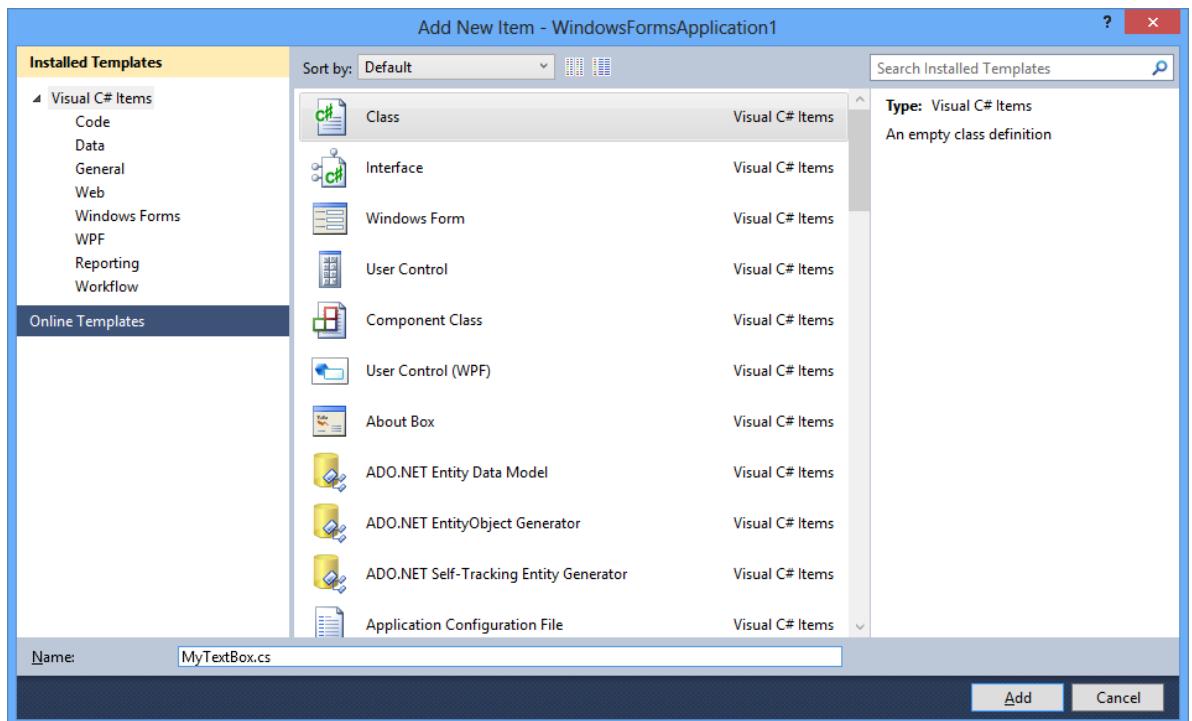
### 9.2.2. Tạo lớp và khai báo thừa kế từ lớp Control

Lập trình viên có thể tạo ra Controls riêng bằng cách tạo ra một lớp mới và chỉ định thừa kế đến một lớp cụ thể. Lớp cụ thể này có thể là lớp: TextBox, Label, ComboBox, ...

Ví dụ 9.2: Tạo ra điều khiển MyTextBox. Điều khiển MyTextBox thừa kế từ lớp TextBox và có thêm thuộc tính mới là NewValue.

Hướng dẫn:

Bước 1: Tạo lớp mới bằng cách chọn Project > Add Class để hiển thị cửa sổ Add New Item và đặt tên lớp là MyTextBox như hình 9.9.



Hình 9.9: Giao diện cửa sổ Add New Item

### Bước 2: Viết mã lệnh cho lớp MyTextBox

- Khai báo thừa kế đến lớp TextBox và thêm thuộc tính NewValue:

```

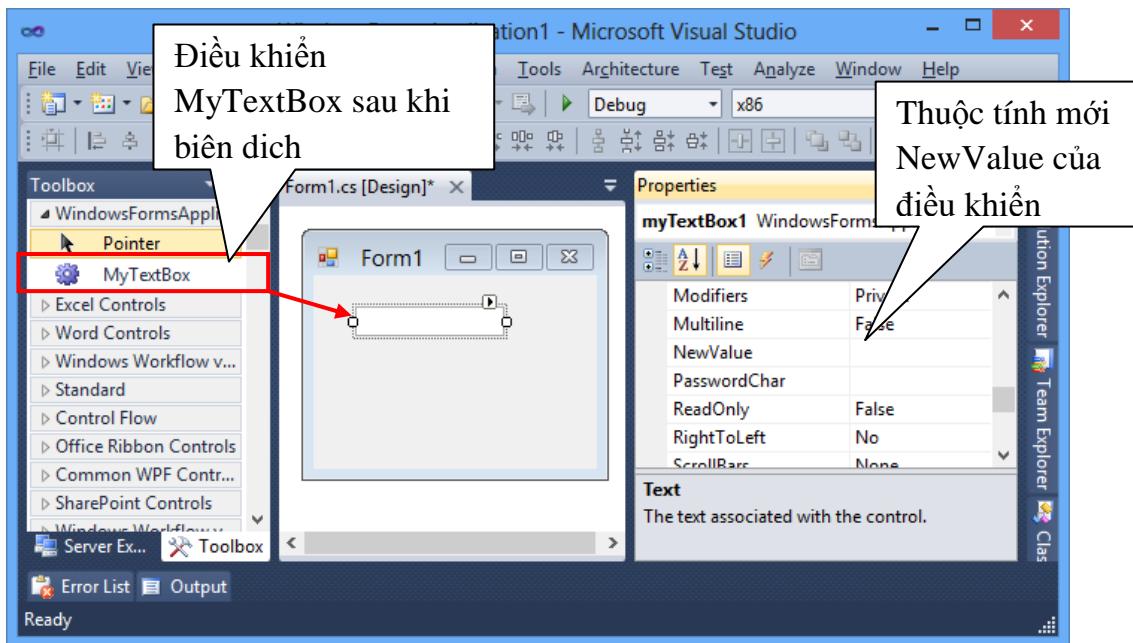
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    class MyTextBox:TextBox
    {
        string strNewValue;
        public string NewValue
        {
            get
            {
                return strNewValue;
            }
            set
            {
                strNewValue = value;
            }
        }
    }
}

```

Khai báo thêm  
không gian tên  
Forms

Thừa kế đến lớp  
TextBox

số



Hình 9.10: Giao diện form sau khi thêm điều khiển MyTextBox

Lập trình viên cũng có thể khai báo chuỗi diễn giải để mô tả chức năng của thuộc tính vừa thêm. Để làm được việc này cần khai báo không gian tên System.ComponentModel và thiết lập thuộc tính Description.

Ví dụ: Khai báo diễn giải cho thuộc tính NewValue cho điều khiển MyTextBox ở ví dụ 9.2.

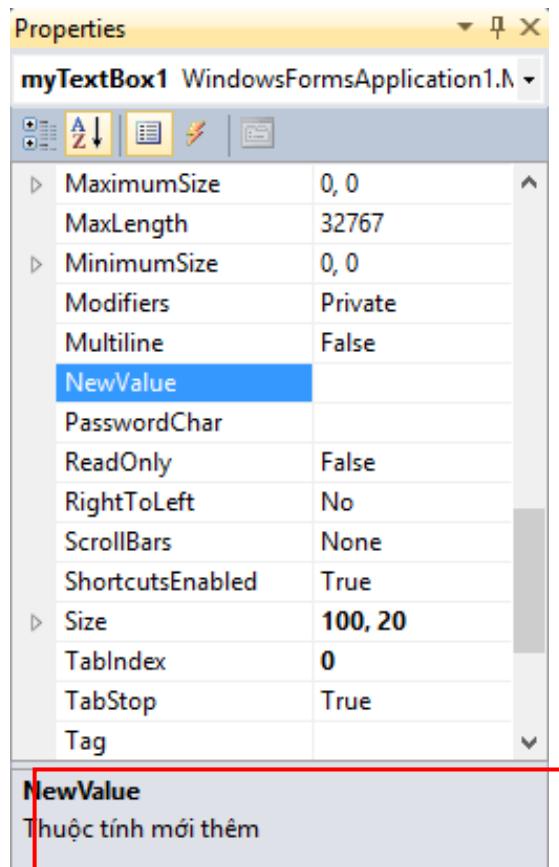
- Viết mã lệnh trong lớp MyTextBox như sau:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.ComponentModel;
namespace WindowsFormsApplication1
{
    class MyTextBox: TextBox
    {
        string strnewValue;
        [Description("Thuộc tính mới thêm")]
        public string NewValue
        {
            get{ return strnewValue; }
            set{ strnewValue = value; }
        }
    }
}

```

- Giao diện chương trình khi thêm mô tả thuộc tính như hình 9.11:

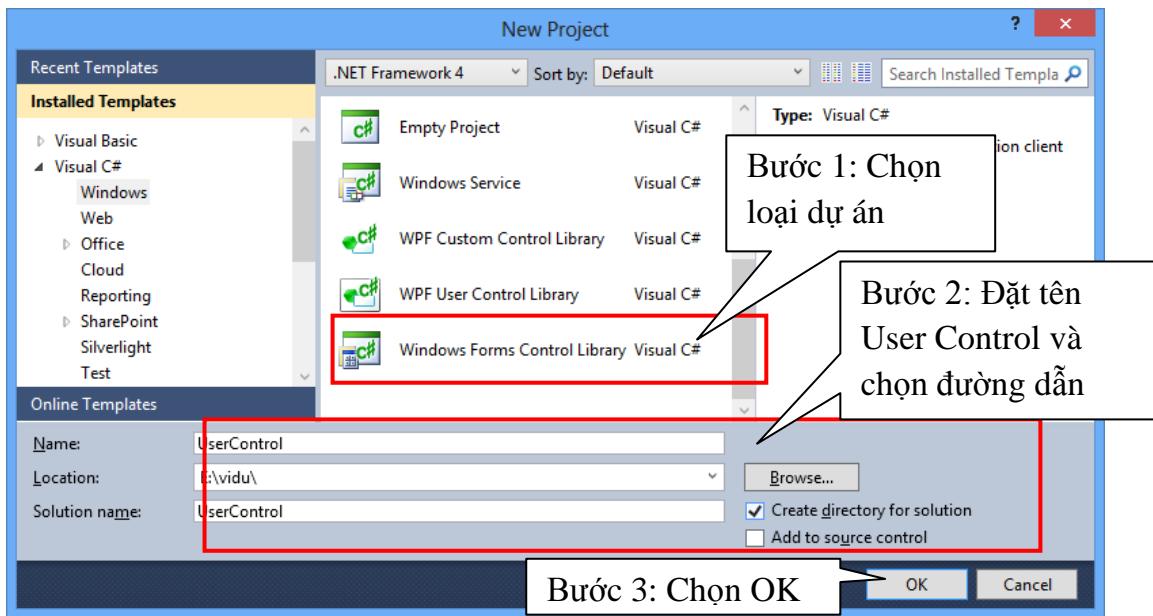


Hình 9.11: Mô tả thuộc tính *NewValue*

### 9.2.3. Tạo dự án mới bằng cách chọn loại dự án là Windows Control Library

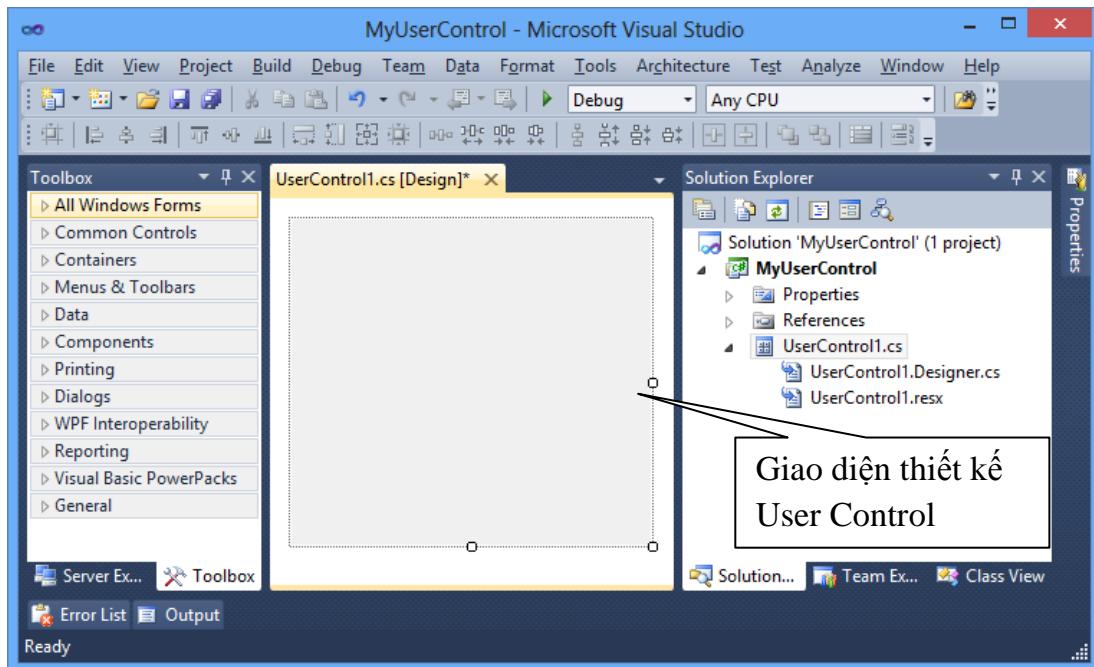
Trong trường hợp muốn tạo User Control để sử dụng cho các dự án khác nhau, lập trình viên có thể tạo User Control và đưa về dạng tập tin .DLL.

Để tạo được User Control ở dạng .DLL, cần tạo mới dự án ở dạng Windows Forms Control Library: Chọn File > New Project hiển thị cửa sổ New Project như hình 9.12.



Hình 9.12: Cửa sổ New Project tạo User Control dạng DLL.

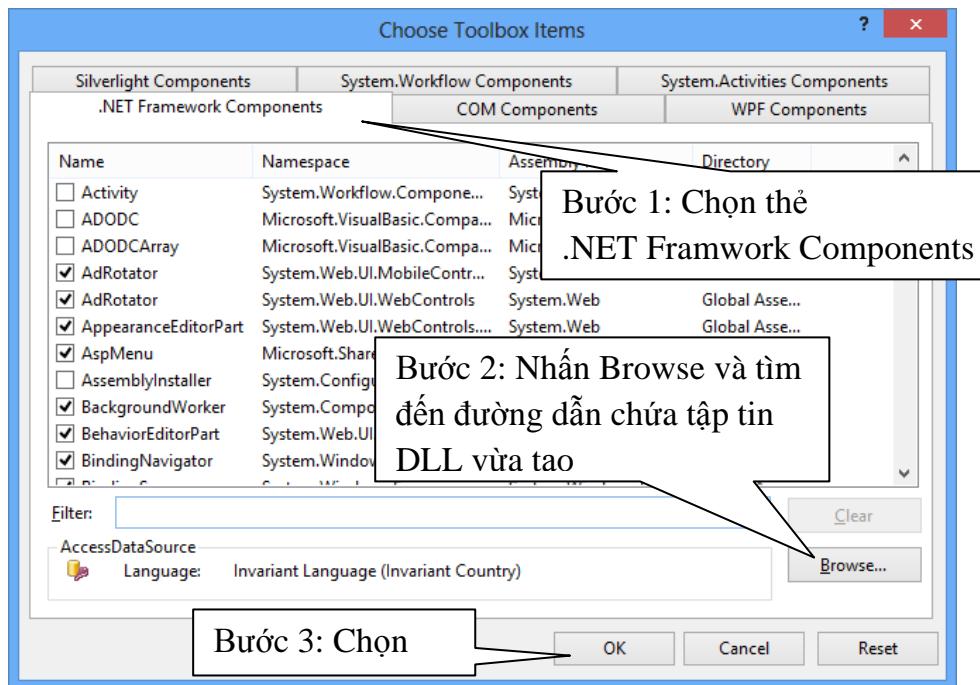
Sau khi chọn OK sẽ được giao diện User Control như hình 9.13:



Hình 9.13: Giao diện User Control khi thêm

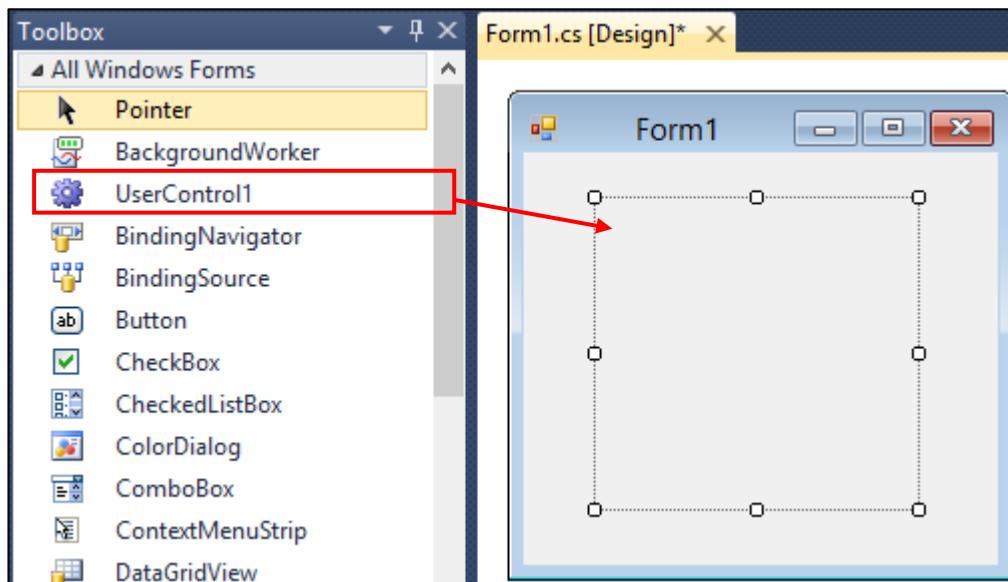
Khi thiết kế xong User Control, lập trình viên cần biên dịch dự án để tạo User Control ở dạng tập tin DLL. Tập tin DLL được tạo trong thư mục Debug. Khi muốn sử dụng User Control trong ứng dụng nào chỉ cần thêm DLL đó vào Toolbox như sau:

- Trên dự án muốn thêm, nhấp phải chuột trên Toolbox chọn Choose Items, cửa sổ Choose Toolbox Items được hiển thị như hình 9.14:



Hình 9.14: Cửa sổ Choose Toolbox Item

User Control được hiển thị trên cửa sổ Toolbox, lập trình viên sử dụng bằng cách kéo User Control vừa thêm vào form như hình 9.15.



Hình 9.15: Thêm User Control vào form từ cửa sổ Toolbox

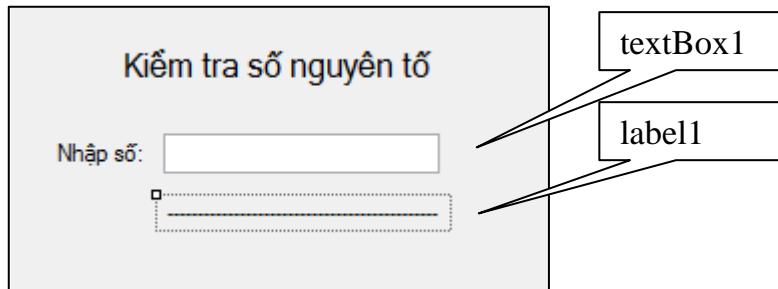
### 9.3. Sử dụng User Control

Sau khi thêm User Control từ cửa sổ Toolbox vào form, lập trình viên có thể truy xuất từng điểm khiển trên User Control thông qua thuộc tính Controls.

Ví dụ muốn gán giá trị “Xin chào các bạn” cho thuộc tính Text cho điều khiển Label có tên lblNhan của UserControl1 cần thực hiện như sau:

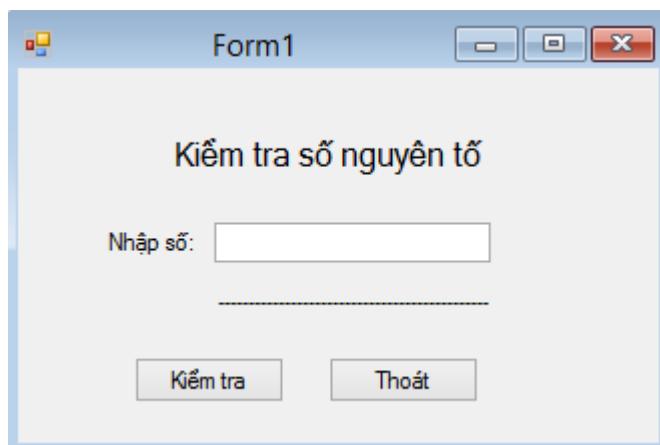
```
UserControl1.Controls["lblNhan"].Text = "Xin chào các bạn";
```

Ví dụ 9.3: Xây dựng kiểm tra số nguyên tố, giao diện xây dựng từ User Control gồm các điều khiển Label, TextBox để nhập số nguyên như hình 9.16.



Hình 9.16: Giao diện User Control kiểm tra số nguyên tố

Sau khi xây dựng xong User Control, tiến hành kéo User Control vào form. Trên form thêm hai Button “Kiểm tra” để kiểm tra số nhập vào có phải số nguyên tố hay không và một Button để thoát chương trình. Giao diện form như hình 9.17.



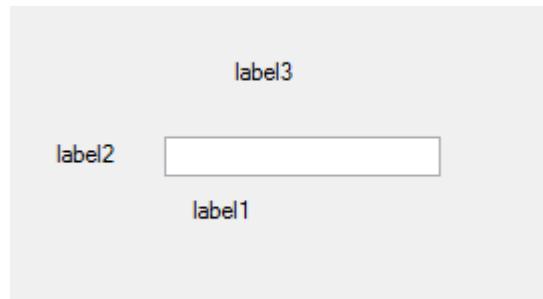
Hình 9.17: Giao diện form kiểm tra số nguyên tố

Yêu cầu: Khi nhấn Button “Kiểm tra”, nếu giá trị trong textBox1 là số nguyên tố thì hiển thị trên label1 chuỗi “Là số nguyên tố”, nếu không phải số nguyên tố thì hiển thị trên label1 chuỗi “Không phải số nguyên tố”.

Hướng dẫn:

Bước 1: Xây dựng giao diện ban đầu:

- Thêm mới User Control: chọn Project > Add User Control. User Control có tên UserControl1 được thêm vào. Trên UserControl1 kéo các điều khiển Label và TextBox từ cửa sổ Toolbox vào như hình 9.18.

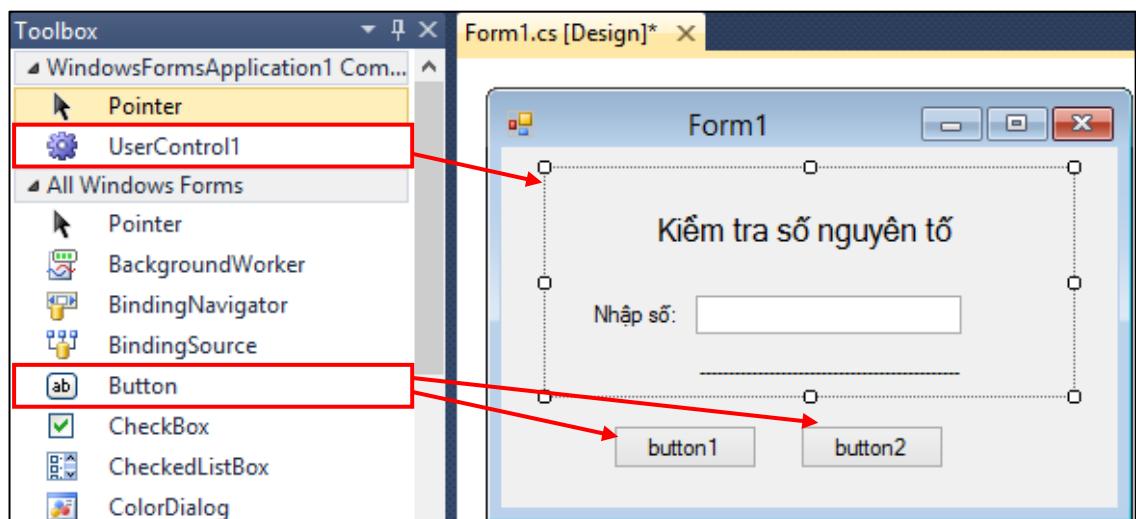


Hình 9.18: Giao diện ban đầu UserControl1

- Thiết lập thuộc tính cho label3:  
Thuộc tính *Text*: “Kiểm tra số nguyên tố”  
Thuộc tính *Font Size*: 14
- Thiết lập thuộc tính cho label2:  
Thuộc tính *Text*: “Nhập số:”
- Thiết lập thuộc tính cho label1:  
Thuộc tính *Text*: “-----”  
Thuộc tính *Name*: lblKetQua
- Thiết lập thuộc tính textBox1:  
Thuộc tính *Name*: txtNhapSo

Khi thiết lập xong thuộc tính sẽ được giao diện UserControl1 như hình 9.16

- Trên form, kéo User Control vừa tạo vào form và thêm hai Button như hình 9.19.



Hình 9.19: Giao diện form khi thêm UserControl1 và Button

- Thiết lập thuộc tính cho button1:  
Thuộc tính *Text*: “Kiểm tra”  
Thuộc tính *Name*: btnKiemTra
- Thiết lập thuộc tính cho button2:  
Thuộc tính *Text*: “Thoát”

Thuộc tính Name: btnThoat

Khi thiết lập xong thuộc tính sẽ được giao diện form như hình 9.17.

Bước 2: Viết mã lệnh cho điều khiển

- Viết hàm kiểm tra số nguyên tố:

```
bool kiemTraSoNguyenTo(int x)
{
    bool kt = true;
    for (int i = 2; i <= x / 2; i++)
        if (x % i == 0)
            kt = false;
    return kt;
}
```

- Sự kiện Click nút btnThoat:

```
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
```

- Sự kiện Click nút btnKiemTra:

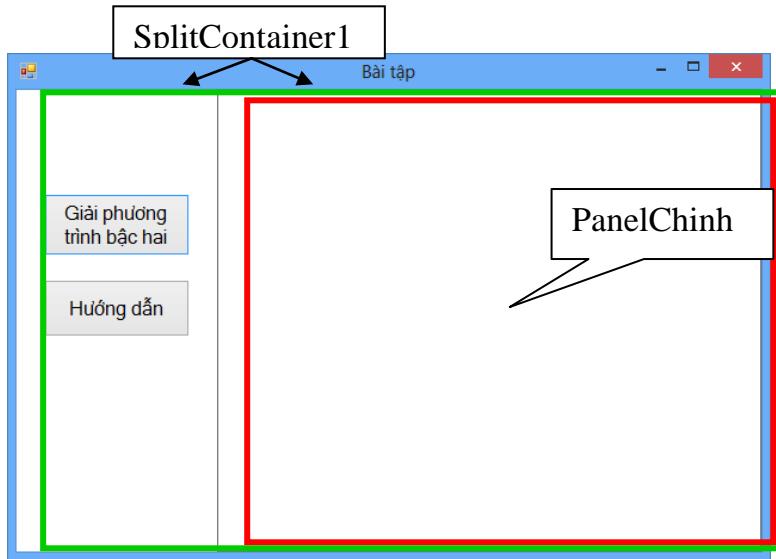
```
private void btnKiemtra_Click(object sender, EventArgs e)
{
    int gt =
        int.Parse(userControl11.Controls["txtNhapSo"].Text);
    if (kiemTraSoNguyenTo(gt) == true)
        userControl11.Controls["lblKetQua"].Text = " là số
                                                nguyên tố";
    else
        userControl11.Controls["lblKetQua"].Text =
            gt.ToString() + " không là số nguyên tố";
    userControl11.Controls["txtNhapSo"].Text = "";
}
```

## 9.4. Cách sử dụng những thư viện tạo sẵn để thiết kế giao diện

Trong thực tế, có những thư viện tạo sẵn hỗ trợ lập trình viên xây dựng giao diện với mục đích tiết kiệm thời gian như: DotNetBar, SandDock, .... Phần lớn các thư viện này đều xây dựng ở dạng tập tin .DLL, do đó lập trình viên có thể dễ dàng sử dụng bằng cách thêm điều khiển vào cửa sổ Toolbox, sau đó kéo thả điều khiển đó vào form. Xem cách sử dụng tại mục 9.2.3.

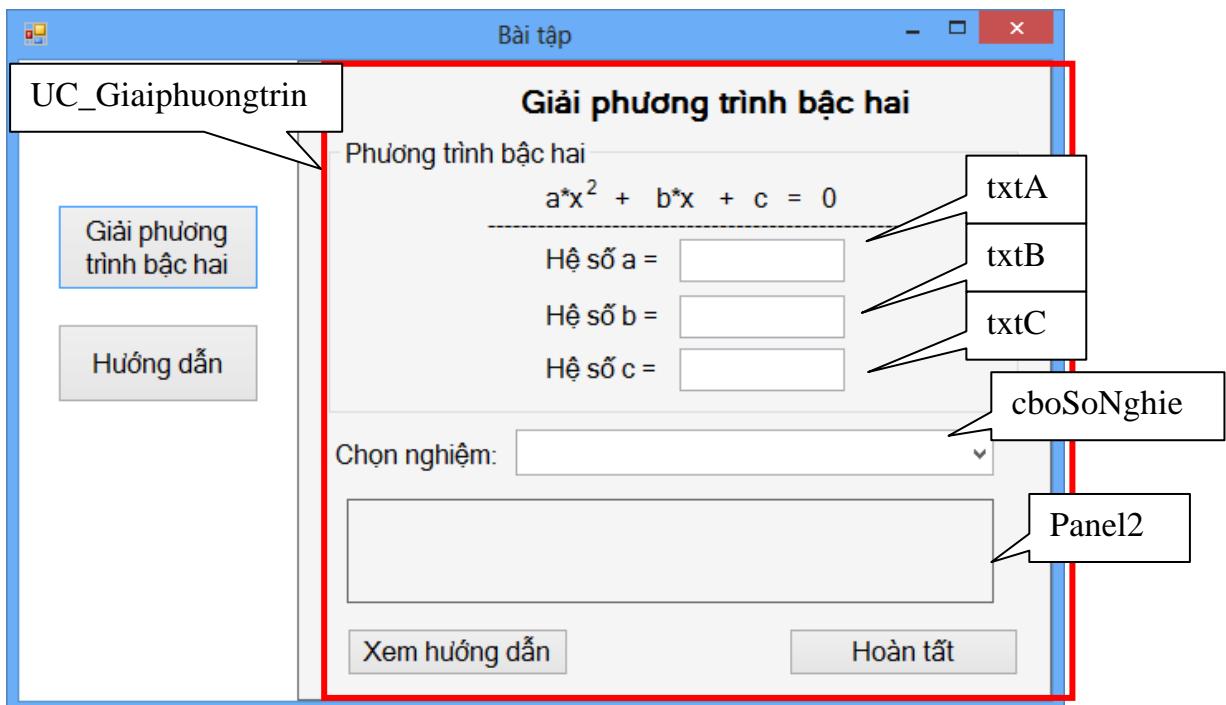
## 9.5. Bài tập cuối chương

Viết chương trình hỗ trợ người dùng học giải phương trình bậc hai có giao diện như hình 9.20.



Hình 9.20: Giao diện chính hỗ trợ người dùng học giải phương trình bậc hai

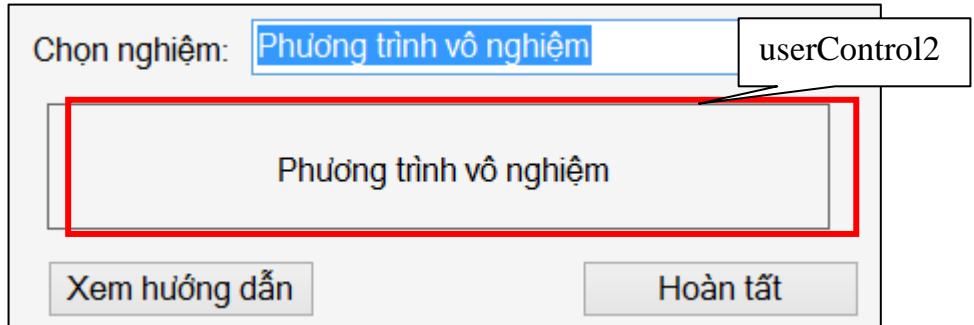
- Khi người dùng nhấp chuột chọn Button “Giải phương trình bậc hai” trên giao diện chính:
  - Chương trình sẽ hiển thị giao diện như hình 9.21. Giao diện “Giải phương trình bậc hai” hiển thị trong PanelChinh là UserControl “UC\_Giaiphuongtrinh”.



Hình 9.21: Giao diện hiển thị phương trình bậc hai

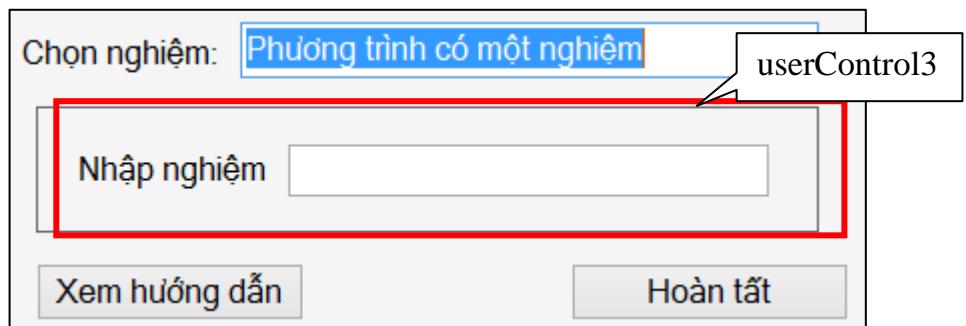
- Các TextBox: txtA, txtB, txtC chứa các hệ số a, b, c của phương trình và được phát sinh ngẫu nhiên.
- ComboBox “cboSoNghiem”: Chứa 3 lựa chọn cho người dùng lựa chọn bao gồm 3 lựa chọn: “Phương trình vô nghiệm”, “Phương trình có một nghiệm”, “Phương trình có hai nghiệm”.

- Với mỗi sự lựa chọn ở cboSoNghiem thì sẽ hiển thị các UserControl: “UC\_VoNghiem”, “UC\_MotNghiem”, “UC\_HaiNghiem” tương ứng như sau:
  - Nếu người dùng chọn mục “Phương trình vô nghiệm” thì Panel2 sẽ hiển thị UserControl “UC\_VoNghiem” bên trong như hình 9.22.



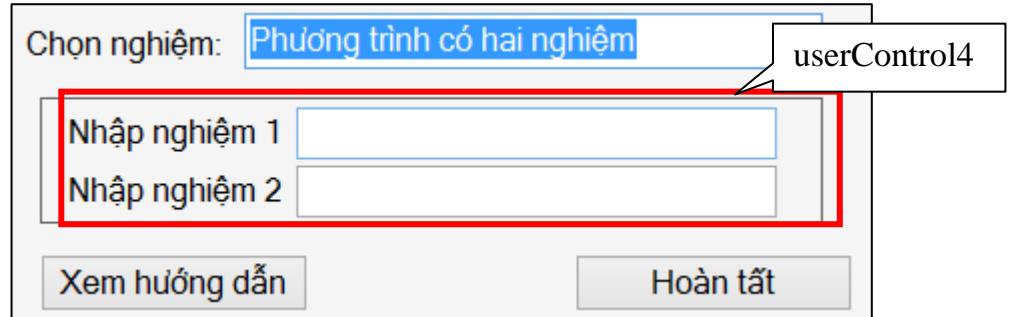
Hình 9.22: Giao diện hiển thị UC\_VoNghiem khi phương trình vô nghiệm

- Nếu người dùng chọn “Phương trình có một nghiệm” thì Panel2 sẽ hiển thị UserControl “UC\_MotNghiem” bên trong như hình 9.23. Trong UserControl “UC\_MotNghiem” có một điều khiển TextBox cho phép người dùng nhập nghiệm đúng của phương trình. Sau khi nhập xong người dùng nhấn Button “Hoàn tất” để kiểm tra kết quả. Nếu kết quả đúng sẽ hiển thị MessageBox với nội dung “Bạn đã giải đúng”, nếu nhập sai sẽ hiển thị MessageBox với nội dung “Bạn đã giải sai”.



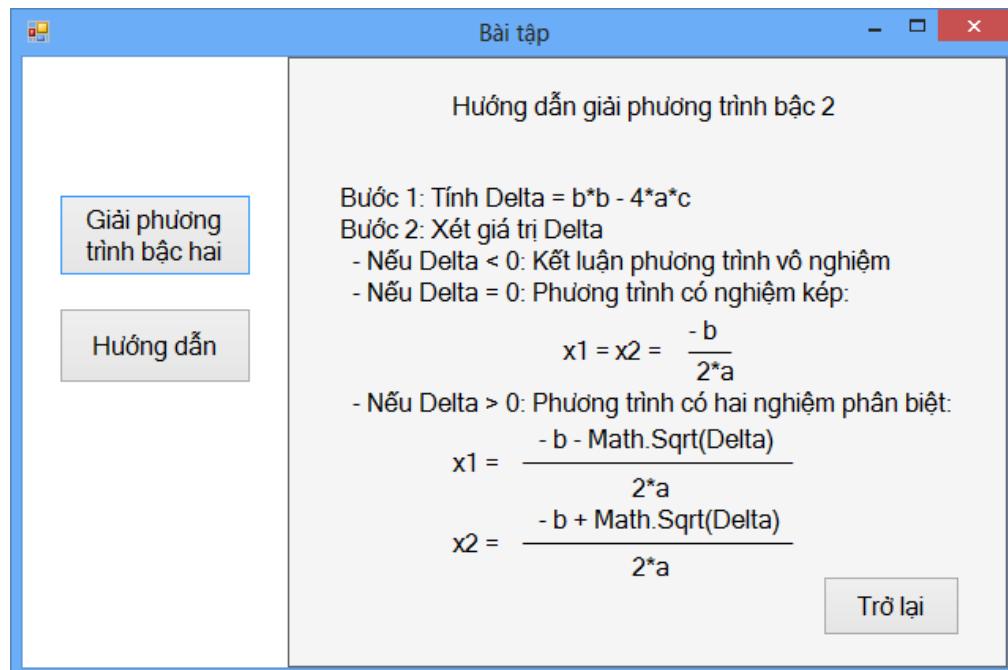
Hình 9.23: Giao diện hiển thị userControl3 khi phương trình có một nghiệm

- Nếu người dùng chọn “Phương trình có hai nghiệm” thì Panel2 sẽ hiển thị UserControl “UC\_HaiNghiem” bên trong như hình 9.24. Trong UserControl “UC\_HaiNghiem” có hai điều khiển TextBox cho phép người dùng nhập nghiệm đúng của phương trình. Sau khi nhập xong người dùng nhấn Button “Hoàn tất” để kiểm tra kết quả. Nếu kết quả đúng sẽ hiển thị MessageBox với nội dung “Bạn đã giải đúng”, nếu nhập sai sẽ hiển thị MessageBox với nội dung “Bạn đã giải sai”.



Hình 9.24: Giao diện hiển thị userControl3 khi phương trình có hai nghiệm

- Nhấn Button “Xem hướng dẫn” thì sẽ hiển thị UserControl “UC\_HuongDan” trên PanelChinh mô tả cách giải phương trình như hình 9.25.



Hình 9.25: Giao diện UC\_Huongdan trên PanelChinh

Nhấn Button “Trở lại” sẽ quay về màn hình “Giải phương trình bậc hai” như hình 9.21.

- Khi người dùng nhấn Button “Hướng dẫn” trên giao diện chính: thì sẽ hiển thị UserControl “UC\_Huongdan” trên PanelChinh như hình 9.25.

# CHƯƠNG 10: LÀM VIỆC VỚI MÀN HÌNH VÀ HỆ THỐNG

## 10.1. Lớp SystemInformation

Lớp *SystemInformation* là lớp cung cấp thông tin về môi trường hệ thống hiện tại, nằm trong không gian tên *System.Windows.Forms*. Các thuộc tính của lớp *SystemInformation* chứa thông tin về hệ thống như: tài khoản đăng nhập, tên máy tính, tình trạng kết nối mạng, tên miền, ...

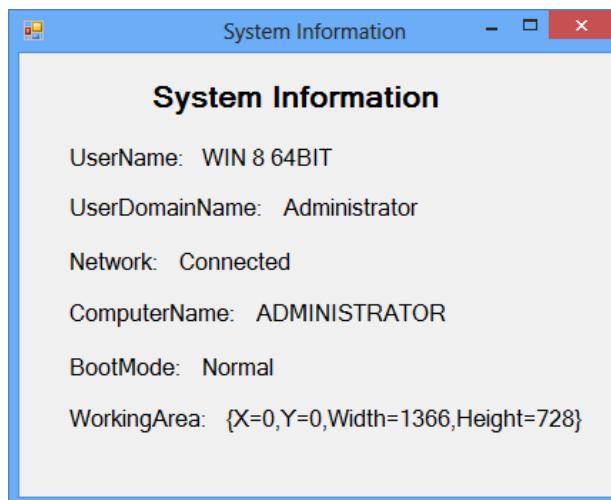
- Các thuộc tính thường dùng của lớp *SystemInformation*:

Bảng 10.1: Bảng mô tả các thuộc tính của lớp *SystemInformation*

Thuộc tính	Mô tả
<i>UserName</i>	Trả về tài khoản người dùng sử dụng trong tiến trình hiện tại.
<i>UserDomainName</i>	Trả về tên miền mà người dùng đã đăng nhập với tài khoản hợp lệ.
<i>Network</i>	Mang hai giá trị True hoặc False. <ul style="list-style-type: none"><li>- Nếu là True: Máy đang trong tình trạng kết nối mạng.</li><li>- Nếu là False: Máy trong tình trạng không kết nối mạng.</li></ul>
<i>ComputerName</i>	Trả về tên của máy tính đã đặt khi cài đặt hệ điều hành
<i>BootMode</i>	Trả về một trong ba giá trị: Normal, FailSafe (SafeMode), FailSafeWithNetwork (Safe Mode With Network). Các giá trị này là các chế độ khởi động của hệ điều hành Windows.
<i>PowerStatus</i>	Trả về giá trị cho biết tình trạng nguồn điện của máy tính trong tình trạng thế nào. <ul style="list-style-type: none"><li>- <i>PowerStatus.BatteryChargeStatus</i>: Cho biết tình trạng nguồn điện của máy tính thế nào. Nếu là máy tính xách tay sẽ được kết quả: Trong tình trạng sạc pin, pin còn nhiều hay ít điện.</li><li>- <i>PowerStatus.BatteryFullLifetime</i>: Trả về thời gian (tính bằng giây) của pin có thể sử dụng khi pin đầy. Nếu không xác định được sẽ trả về -1.</li><li>- <i>PowerStatus.BatteryLifePercent</i>: Hiển thị % nguồn điện còn lại trong pin.</li><li>- <i>PowerStatus.BatteryLifeRemaining</i>: Trả về thời gian sử dụng còn lại của pin (tính bằng giây). Trả về -1 nếu</li></ul>

	sử dụng nguồn điện trực tiếp. - PowerStatus.PowerLineStatus: Trả về giá trị Online nếu máy tính có cắm điện, ngược lại trả về Offline.
WorkingArea	Cho biết thông tin về kích cỡ (Width và Height) vùng làm việc trên màn hình.

Ví dụ 10.1: Viết chương trình hiển thị thông tin máy tính có giao diện như hình 10.1.



Hình 10.1: Giao diện chương trình hiển thị thông tin máy tính

Hướng dẫn:

Bước 1: Thiết kế giao diện form ban đầu như hình 10.2.



Hình 10.2: Giao diện ban đầu form hiển thị thông tin máy tính

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties.

- Form1:
  - Thuộc tính *Text*: “System Information”
- label1:
  - Thuộc tính *Text*: “System Information”
  - Thuộc tính *Font Size*: 16
  - Thuộc tính *Font Style*: Bold
- label2:

- thuộc tính *Text*: “UserName.”
- label3:  
Thuộc tính *Name*: username
- label4:  
Thuộc tính *Text*: “UserDomainName.”
- label5:  
Thuộc tính *Name*: userdomainname
- label6:  
Thuộc tính *Text*: “NetWork.”
- label7:  
Thuộc tính *Name*: network
- label8:  
Thuộc tính *Text*: “ComputerName.”
- label9:  
Thuộc tính *Name*: computename
- label10:  
Thuộc tính *Text*: “BootMode.”
- label11:  
Thuộc tính *Name*: bootmode
- label12:  
Thuộc tính *Text*: “WorkingArea.”
- label13:  
Thuộc tính *Name*: workingarea

Bước 3: Viết mã lệnh cho các điều khiển.

- Sự kiện *Load* của form:

```
private void Form1_Load(object sender, EventArgs e)
{
    username.Text = SystemInformation.UserName.ToString();
    userdomainname.Text =
        SystemInformation.UserDomainName.ToString();
    if (SystemInformation.Network == true)
        Network.Text = "Connected";
    else
        Network.Text = "Disconnected";
    computename.Text =
        SystemInformation.ComputerName.ToString();
    bootmode.Text = SystemInformation.BootMode.ToString();
    workingarea.Text =
        SystemInformation.WorkingArea.ToString();
}
```

## 10.2. Lớp Screen

Lớp *Screen* nằm trong không gian tên `System.Windows.Forms`, lớp *Screen* cung cấp các thông tin về độ phân giải hiện hành của màn hình, tên thiết bị và số lượng Bit trên một Pixel.

- Các thuộc tính thường dùng của lớp *Screen*:

Thuộc tính thường được sử dụng của lớp *Screen* là *PrimaryScreen*. Thông qua *PrimaryScreen*, lập trình viên có thể lấy được kích thước vùng hiển thị (*Bounds*), kích thước vùng làm việc, số bit trên một pixel, tên thiết bị.

*Bảng 10.2: Bảng mô tả thuộc tính của lớp Screen*

Thuộc tính	Mô tả
<i>PrimaryScreen.Bounds</i>	Lấy kích thước hiển thị trên màn hình
<i>PrimaryScreen.WorkingArea</i>	Lấy kích thước của màn hình làm việc
<i>PrimaryScreen.BitsPerPixel</i>	Lấy số lượng bit trên một Pixel

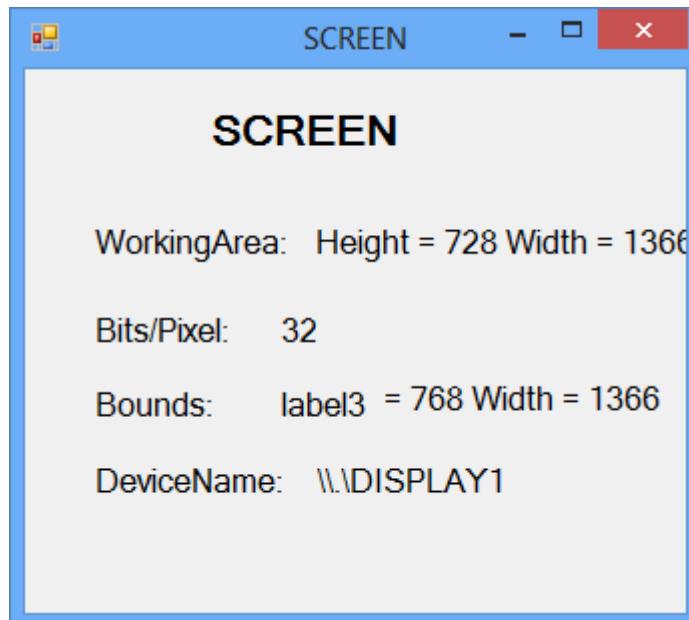
- Các phương thức thường dùng của lớp *Screen*:

Ngoài cách sử dụng thuộc tính *PrimaryScreen* để lấy kích thước màn hình làm việc, lập trình viên cũng có thể sử dụng phương thức *GetWorkingArea*.

*Bảng 10.3: Bảng mô tả phương thức của lớp Screen*

Thuộc tính	Mô tả
<i>GetWorkingArea</i>	Cung cấp các thông tin về kích thước của màn hình làm việc

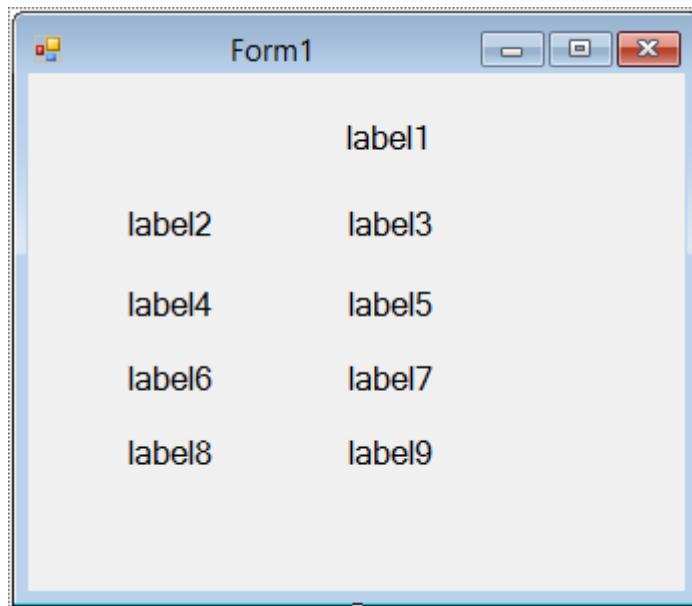
Ví dụ 10.2: Viết chương trình hiển thị các thông tin độ phân giải màn hình, tên thiết bị, số bit trên một pixel như hình 10.3.



*Hình 10.3: Giao diện hiển thị thông tin của màn hình*

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển Label từ cửa sổ Toolbox vào form như hình 10.4.



Hình 10.4: Giao diện ban đầu form hiển thị thông tin màn hình

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển trong cửa sổ Properties.

- Form1:  
Thuộc tính *Text*: “SCREEN”
- label1:  
Thuộc tính *Text*: “System SCREEN”  
Thuộc tính *Font Size*: 16  
Thuộc tính *Font Style*: Bold
- label2:  
Thuộc tính *Text*: “WorkingArea:”
- label3:  
Thuộc tính *Name*: workingarea
- label4:  
Thuộc tính *Text*: “Bits/Pixel:”
- label5:  
Thuộc tính *Name*: bits
- label6:  
Thuộc tính *Text*: “Bounds:”
- label7:  
Thuộc tính *Name*: bounds
- label8:  
Thuộc tính *Text*: “DeviceName:”
- label9:

Thuộc tính *Name*: devicename

Bước 3: Viết mã lệnh cho điều khiển

Sự kiện *Load* của form:

```
private void Form1_Load(object sender, EventArgs e)
{
    Rectangle rec = Screen.GetWorkingArea(this);
    workingarea.Text = "Height = " + rec.Height + " Width =
" + rec.Width;
    bits.Text =
    Screen.PrimaryScreen.BitsPerPixel.ToString();
    bounds.Text = "Height = "
    Screen.PrimaryScreen.Bounds.Height.ToString() + " Width
= " + Screen.PrimaryScreen.Bounds.Width.ToString();
    devicename.Text =
    Screen.PrimaryScreen.DeviceName.ToString();
}
```

### 10.3. Lớp SendKeys

Lớp *SendKeys* nằm trong không gian tên *System.Windows.Forms*, lớp *SendKeys* cung cấp các phương thức cho phép làm việc với các phím trên bàn phím. Ba phương thức thường được sử dụng trong lớp *SendKeys* là phương thức: *Send*, *SendWait* và *Flush*.

- Các phương thức thường dùng của lớp *SendKeys*:

*Bảng 10.4: Bảng mô tả các phương thức của lớp SendKeys*

Phương thức	Mô tả
<i>Send(&lt;Mã phím&gt;)</i>	<p>Phương thức <i>Send</i> cho phép gửi phím đến ứng dụng hiện hành. Để gửi một phím nào đó đến ứng dụng lập trình viên chỉ việc gởi ký tự tương ứng trong phương thức <i>Send</i>.</p> <p>Ví dụ: Muốn gửi phím A đến ứng dụng, lập trình viên phải viết như sau:</p> <p style="text-align: center;">SendKeys.Send("A");</p> <p>Trường hợp muốn gửi tổ hợp phím đến ứng dụng, lập trình viên sẽ gửi một chuỗi ký tự tương ứng trong phương thức <i>Send</i>.</p> <p>Ví dụ: Muốn gửi tổ hợp phím A và B đến ứng dụng, lập trình viên phải viết như sau:</p> <p style="text-align: center;">SendKeys.Send("AB");</p> <p>Một số ký tự đặc biệt như +, %, ^, ~, (, ), [ , ], { , } nếu</p>

	<p>muốn gửi đến ứng dụng phải đặt trong cặp dấu { }. Ví dụ: Muốn gửi ký tự + đến ứng dụng, lập trình viên phải viết như sau:</p> <pre>SendKeys.Send("{{+}}");</pre> <p>Các phím đặc biệt không thuộc nhóm ký tự như: BackSpace, Enter, Caps Lock, Delete, ... cũng có thể gửi đến ứng dụng nhưng phải gửi bằng mã tương ứng. Bảng mã được mô tả cụ thể trong bảng 10.4.</p> <p>Gửi các phím chức năng: F1, F2, F3, .. thì gửi bằng mã của phím tương ứng như mô tả trong bảng 10.5.</p>
<i>SendWait(&lt;Mã phím&gt;)</i>	Phương thức SendWait cũng thực hiện công việc gửi phím hoặc tổ hợp phím đến ứng dụng hiện hành như phương thức Send. Tuy nhiên, điểm khác biệt là phương thức SendWait sẽ chờ cho công việc được gọi khi gửi phím bằng SendWait đến ứng dụng thực hiện xong mới thực hiện các công việc sau đó.
<i>Flush()</i>	Phương thức Flush có chức năng bỏ qua những thực thi đang chờ để thực thi một công việc khác. Cụ thể nếu có một tiến trình đang chạy và muốn ứng dụng ngay lập tức thực thi một công việc khác mà không phải chờ tiến trình làm việc xong thì cần gọi phương thức Flush().

Bảng 10.5: Bảng mã các phím đặc biệt

Phím	Bảng mã
<i>BACKSPACE</i>	{BACKSPACE}, {BS}, {BKSP}
<i>BREAK</i>	{BREAK}
<i>CAPS LOCK</i>	{CAPSLOCK}
<i>DELETE</i>	{DELETE}, {DEL}
<i>END</i>	{END}
<i>ENTER</i>	{ENTER}
<i>ESC</i>	{ESC}
<i>HELP</i>	{HELP}
<i>HOME</i>	{HOME}
<i>INSERT</i>	{INSERT}
<i>NUM LOCK</i>	{NUMLOCK}
<i>PAGE DOWN</i>	{PGDN}

<i>PAGE UP</i>	{PGUP}
<i>DOWN ARROW</i>	{DOWN}
<i>UP ARROW</i>	{UP}
<i>LEFT ARROW</i>	{LEFT}
<i>RIGHT ARROW</i>	{RIGHT}
<i>PRINT SCREEN</i>	{PRTSC}
<i>SCROLL LOCK</i>	{SCROLLLOCK}
<i>TAB</i>	{TAB}

Bảng 10.6: Bảng mã các phím chức năng

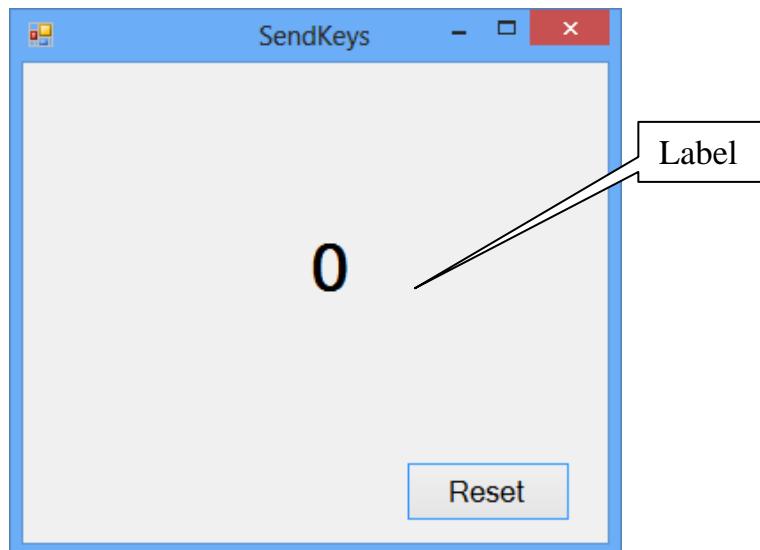
Phím	Bảng mã
<i>F1</i>	{F1}
<i>F2</i>	{F2}
<i>F3</i>	{F3}
<i>F4</i>	{F4}
<i>F5</i>	{F5}
<i>F6</i>	{F6}
<i>F7</i>	{F7}
<i>F8</i>	{F8}
<i>F9</i>	{F9}
<i>F10</i>	{F10}
<i>F11</i>	{F11}
<i>F12</i>	{F12}
+	{ADD}
-	{SUBTRACT}
*	{MULTIPLY}
/	{DIVIDE}

Trường hợp muốn gửi một tổ hợp phím mà trong đó có phím Shift, Ctrl hay Alt lập trình viên có thể sử dụng mã của các phím Shift, Ctrl và Alt tương ứng trong bảng 10.6.

Bảng 10.7: Bảng mã các phím chức năng

Phím	Bảng mã
<i>Ctrl</i>	^
<i>Shift</i>	+
<i>Alt</i>	%

Ví dụ 10.3: Viết chương trình tăng số có giao diện như hình 10.5.



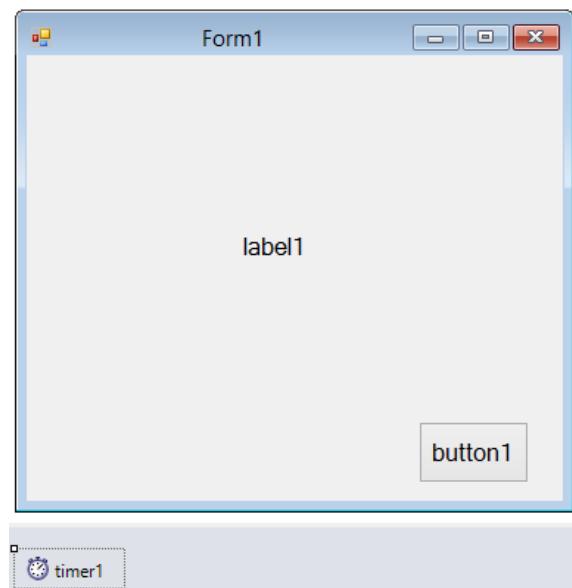
Hình 10.5: Giao diện chương trình tăng số

Yêu cầu:

- Khi nhấn F5, form như hình 10.5 sẽ hiển thị. Mỗi một giây số hiển thị trên Label sẽ tăng 1 đơn vị.
- Khi nhấn Button “Reset” thì số hiển thị trong Label sẽ trở về 0.
- Khi số hiển thị trong Label đạt giá trị 60 thì chương trình sẽ tự động thực hiện việc nhấn Button “Reset” để thiết lập lại giá trị trong Label trở về 0.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển Label, Timer, Button từ cửa sổ Toolbox vào form như hình 10.6.



Hình 10.6: Giao diện ban đầu chương trình tăng số

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển.

- Form1:

Thuộc tính Text: “Sendkeys”

- Thuộc tính AcceptButton: btnReset
- label1:
 

Thuộc tính Text: “---”

Thuộc tính Name: lblHienthi

Thuộc tính Font Size: 24

Thuộc tính Font Style: Bold
  - button1:
 

Thuộc tính Text: “Reset”

Thuộc tính Name: btnReset
  - timer1:
 

Thuộc tính Interval: 1000

Thuộc tính Enable: True

Bước 3: Viết mã lệnh cho các điều khiển

- Khai báo biến chương trình:

```
int time = 0;
```

- Sự kiện Click của btnReset:

```
private void btnReset_Click(object sender, EventArgs e)
{
    time = 0;
}
```

- Sự kiện Tick của Timer1:

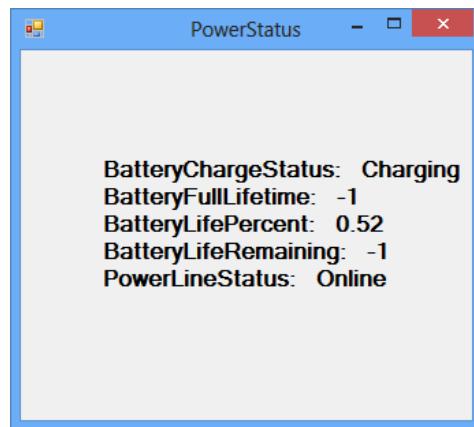
```
private void timer1_Tick(object sender, EventArgs e)
{
    lblHienThi.Text = time.ToString();
    time += 1;
    if (time == 60)
        SendKeys.Send("{Enter}");
}
```

## 10.4. Lớp PowerStatus

Lớp PowerStatus chứa các thuộc tính cung cấp thông tin về tình trạng nguồn điện đang sử dụng của máy tính. Điểm đặc biệt là lớp PowerStatus không có phương thức khởi tạo, do đó lập trình viên có thể tạo đối tượng lớp PowerStatus và sử dụng thông qua thuộc tính PowerStatus của lớp SystemInformation như sau:

```
PowerStatus pwts;
pwts = SystemInformation.PowerStatus;
string power = "";
power = pwts.BatteryChargeStatus.ToString();
power += " " + pwts.BatteryFullLifetime.ToString();
power += " " + pwts.BatteryLifePercent.ToString();
power += " " + pwts.PowerLineStatus.ToString();
```

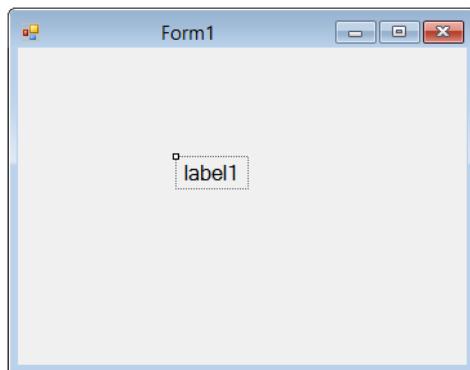
Ví dụ 10.4: Viết chương trình hiển thị tình trạng của pin có giao diện như hình 10.7.



Hình 10.7: Giao diện hiển thị thông tin tình trạng pin

Hướng dẫn:

Bước 1: Thiết kế form có giao diện ban đầu như hình 10.8: Thêm điều khiển Label từ Toolbox vào form.



Hình 10.8: Giao diện ban đầu form hiển thị thông tin tình trạng pin

Bước 2: Thiết lập giá trị thuộc tính trong cửa sổ Properties.

- Form1:

Thuộc tính *Text*: “PowerStatus”

label1:

Thuộc tính *Name*: lblHienthi

Bước 3: Viết mã lệnh cho chương trình

- Sự kiện *Load* của form:

```

private void Form1_Load(object sender, EventArgs e)
{
    PowerStatus pwts;
    pwts = SystemInformation.PowerStatus;
    string power = "BatteryChargeStatus: ";
    power += pwts.BatteryChargeStatus.ToString();
    power += "\nBatteryFullLifetime: " +
    pwts.BatteryFullLifetime.ToString();
    power += "\nBatteryLifePercent: " +
    pwts.BatteryLifePercent.ToString();
    power += "\nBatteryLifeRemaining: " +
    pwts.BatteryLifeRemaining.ToString();
    power += "\nPowerLineStatus: " +
    pwts.PowerLineStatus.ToString();
    lblHienThi.Text = power;
}

```

## 10.5. Lớp Application

Lớp *Application* chứa các thuộc tính và phương thức liên quan đến ứng dụng như: khởi động ứng dụng, đóng ứng dụng hoặc các thông tin về ứng dụng.

Lưu ý lớp *Application* là dạng lớp cô lập (Sealed) do đó không cho phép các lớp khác thừa kế.

- Các thuộc tính thường dùng của lớp *Application*:

*Bảng 10.8: Bảng mô tả các thuộc tính của lớp Application*

Thuộc tính	Mô tả
<i>CommonAppDataPath</i>	Trả về đường dẫn, đường dẫn này lưu trữ những tập tin để chia sẻ cho người dùng khác sử dụng.
<i>CompanyName</i>	Trả về tên của công ty được khai báo trong thuộc <i>AssemblyCompany</i> của tập tin AssemblyInfo.cs
<i>CurrentCulture</i>	Cho phép thiết lập hoặc lấy các thông tin định dạng thời gian, ngày tháng, ngôn ngữ, ... hiện đang sử dụng
<i>CurrentInputLanguage</i>	Cho phép thiết lập hoặc lấy thông tin về ngôn ngữ đang sử dụng
<i>ExecutablePath</i>	Trả về đường dẫn của tập tin thực thi (tập tin có đuôi .exe). Ví dụ: C:\Users\PC_PHUC\Desktop\DuAnDauTien\DuAnDauTien\bin\Debug\DuAnDauTien.exe
<i>ProductName</i>	Trả về tên của ứng dụng được khai báo trong thuộc

	tính <i>AssemblyProduct</i> của tập tin AssemblyInfo.cs
<i>ProductVersion</i>	Trả về phiên bản của ứng dụng được khai báo trong thuộc tính <i>AssemblyVersion</i> của tập tin AssemblyInfo.cs
<i>StartupPath</i>	Trả về đường dẫn chứa tập tin thực thi, đường dẫn này không chứa tên của tập tin thực thi. Ví dụ: C:\Users\PC_PHUC\Desktop\DuAnDauTien\DuAnDauTien\bin\Debug
<i>UserAppDataPath</i>	Tương tự thuộc tính <i>CommonAppDataPath</i> , <i>UserAppDataPath</i> Trả về đường dẫn, đường dẫn này lưu trữ những tập tin để chia sẻ cho người dùng khác sử dụng.

➤ Các phương thức thường dùng của lớp Application:

Bảng 10.9: Bảng mô tả các phương thức của lớp Application

Phương thức	Mô tả
<i>Exit()</i>	Giúp đóng ứng dụng.
<i>ExitThread()</i>	Giúp đóng tiến trình. Một ứng dụng có thể có nhiều tiến trình chạy song song, phương thức ExitThread() giúp đóng tiến trình hiện tại.
<i>DoEvents()</i>	Có chức năng như phương thức Flush của lớp SendKeys. có chức năng bỏ qua những thực thi đang chờ để thực thi một công việc khác. Cụ thể nếu có một tiến trình đang chạy và muốn ứng dụng ngay lập tức thực thi một công việc khác mà không phải chờ tiến trình làm việc xong thì cần gọi phương thức DoEvents().
<i>Restart()</i>	Phương thức Restart() thực hiện hai công việc là đóng ứng dụng lại sau đó thì sẽ khởi động ứng dụng
<i>Run()</i>	Phương thức Run() giúp khởi động một đối tượng trong ứng dụng.

## 10.6. Lớp Clipboard

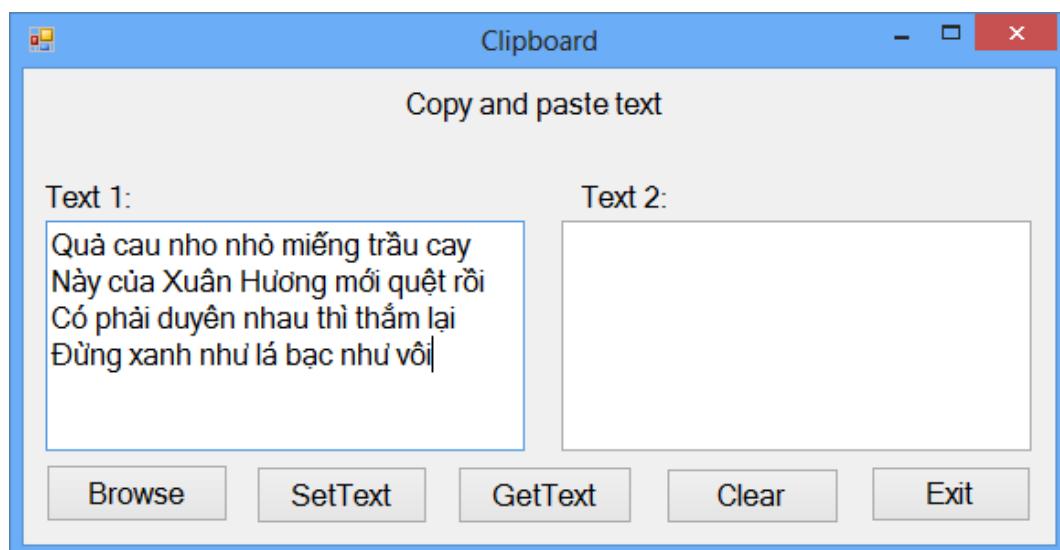
Lớp *Clipboard* cung cấp cho người một vùng nhớ tạm để lưu trữ dữ liệu, đồng thời cũng hỗ trợ các phương thức cho phép thực hiện công việc sao chép dữ liệu từ nơi này và dán dữ liệu đến một nơi khác thông qua vùng nhớ tạm mà *Clipboard* cung cấp.

- Các phương thức thường dùng của lớp *Clipboard*:

Bảng 10.10: Bảng mô tả các phương thức của lớp *Clipboard*

Phương thức	Mô tả
<i>Clear()</i>	Cho phép xóa dữ liệu đang chứa trong <i>Clipboard</i>
<i>SetText(&lt;Chuỗi&gt;)</i>	Lưu <chuỗi> vào trong <i>Clipboard</i>
<i>GetText()</i>	Trả về chuỗi lưu trữ trong <i>Clipboard</i>
<i>ContainsText()</i>	Trả về một trong hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Trong <i>Clipboard</i> có nội dung</li> <li>- Nếu là False: Trong <i>Clipboard</i> không chứa một chuỗi nào cả.</li> </ul>
<i>SetImage(&lt;Image&gt;)</i>	Lưu trữ hình ảnh vào <i>Clipboard</i>
<i>GetImage()</i>	Trả về hình đang lưu trữ trong <i>Clipboard</i>
<i>ContainsImage()</i>	Trả về một trong hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Trong <i>Clipboard</i> có chứa hình ảnh</li> <li>- Nếu là False: Trong <i>Clipboard</i> không chứa hình ảnh.</li> </ul>
<i>SetData(&lt;Object&gt;)</i>	Lưu trữ đối tượng <Object> vào <i>Clipboard</i>
<i>GetData()</i>	Trả về đối tượng đang lưu trữ trong <i>Clipboard</i>
<i>ContainsData()</i>	Trả về một trong hai giá trị True hoặc False. <ul style="list-style-type: none"> <li>- Nếu là True: Trong <i>Clipboard</i> có chứa đối tượng</li> <li>- Nếu là False: Trong <i>Clipboard</i> không chứa đối tượng</li> </ul>

Ví dụ 10.5: Viết chương trình thực hiện công việc sao chép văn bản từ hai TextBox. Chương trình gồm các điều khiển: TextBox, Label, Button, OpenFileDialog. Giao diện chương trình như hình 10.9.



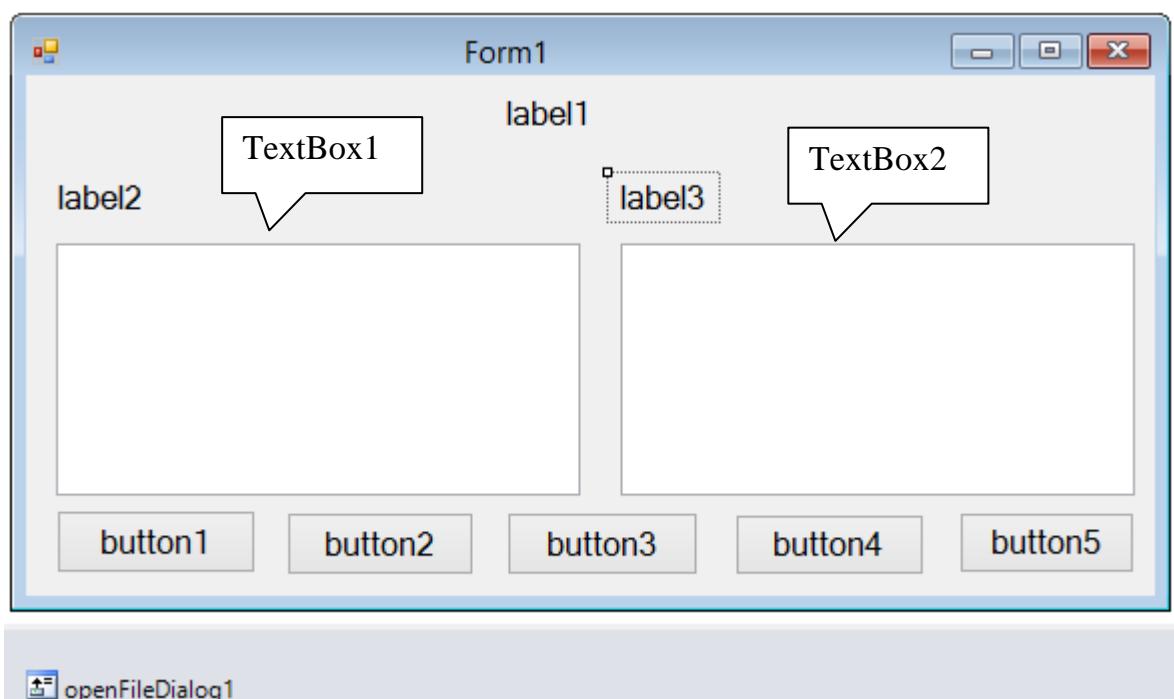
Hình 10.9: Giao diện form sao chép văn bản

Yêu cầu:

- Browse: Giúp chọn tập tin .txt trong hệ thống và hiển thị nội dung tập tin trên TextBox txt1.
- SetText: Sao chép nội dung văn bản trong TextBox txt1 vào Clipboard
- GetText: Chép nội dung văn bản trong Clipboard vào TextBox txt2
- Clear: Xóa dữ liệu lưu trong Clipboard
- Exit: Đóng chương trình.

Hướng dẫn:

Bước 1: Thiết kế giao diện ban đầu: Thêm các điều khiển Label, TextBox, Button, OpenFileDialog từ cửa sổ Toolbox vào form như hình 10.10.



Hình 10.10: Giao diện ban đầu form sao chép văn bản

Bước 2: Thiết lập giá trị thuộc tính cho điều khiển

- Form1:
  - Thuộc tính Text: “Clipboard”
  - Thuộc tính Size: 580, 299
- label1:
  - Thuộc tính Text: “Copy and paste Text”
- label2:
  - Thuộc tính Text: “Text 1:”
- label3:
  - Thuộc tính Text: “Text 2:”
- TextBox1:
  - Thuộc tính Multiline: True

- Thuộc tính Name: txt1
- Thuộc tính Size: 262, 126
- TextBox2:
  - Thuộc tính Multiline: True
  - Thuộc tính Name: txt2
  - Thuộc tính Size: 262, 126
- button1:
  - Thuộc tính Text: “Browse”
  - Thuộc tính Name: browse
- button2:
  - Thuộc tính Text: “SetText”
  - Thuộc tính Name: settext
- button3:
  - Thuộc tính Text: “GetText”
  - Thuộc tính Name: gettext
- button4:
  - Thuộc tính Text: “Clear”
  - Thuộc tính Name: clear
- button5:
  - Thuộc tính Text: “Exit”
  - Thuộc tính Name: exit
- openFileDialog1:
  - Thuộc tính Filter: “File Text|\*.txt”

### Bước 3: Viết mã lệnh cho các điều khiển

- Sự kiện *Click* của Button SetText:

```
private void settext_Click(object sender, EventArgs e)
{
    Clipboard.SetText(txt1.Text);
}
```

- Sự kiện *Click* của Button Browse:

```
private void browse_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string tentaptin = openFileDialog1.FileName;
        StreamReader rd = new StreamReader(tentaptin);
        txt1.Text = rd.ReadToEnd();
        rd.Close();
    }
}
```

- Sự kiện *Click* của Button GetText:

```
private void gettext_Click(object sender, EventArgs e)
{
    txt2.Text = Clipboard.GetText();
}
```

- Sự kiện *Click* của Button Clear:

```
private void clear_Click(object sender, EventArgs e)
{
    Clipboard.Clear();
}
```

- Sự kiện *Click* của Button Exit:

```
private void exit_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

## 10.7. Lớp Cursors

Lớp Cursors cung cấp tập các con trỏ có hình dạng khác nhau sử dụng trong ứng dụng Windows Forms như bảng 10.10. Thông thường, khi ứng dụng trong những tình trạng khác nhau thì lập trình viên sẽ thay đổi trạng thái của con trỏ cho phù hợp để người dùng dễ nhận biết.

Bảng 10.11: Bảng mô tả các thuộc tính thường dùng của lớp Clipboard

Thuộc tính lớp Cursors	Mô tả
<i>Cursors.AppStarting</i>	
<i>Cursors.Default</i>	
<i>Cursors.No</i>	
<i>Cursors.SizeAll</i>	
<i>Cursors.SizeNESW</i>	
<i>Cursors.SizeNS</i>	
<i>Cursors.SizeNWSE</i>	
<i>Cursors.SizeWE</i>	
<i>Cursors.WaitCursor</i>	

<i>Cursors.Hand</i>	
<i>Cursors.Help</i>	
<i>Cursors.NoMove2D</i>	
<i>Cursors.NoMoveHoriz</i>	
<i>Cursors.NoMoveVert</i>	
<i>Cursors.PanEast</i>	
<i>Cursors.PanNE</i>	
<i>Cursors.PanNorth</i>	
<i>Cursors.PanNW</i>	
<i>Cursors.PanSE</i>	
<i>Cursors.PanSouth</i>	

Ví dụ 10.6: Trên form sao chép văn bản hình 10. 9. Thiết lập thuộc tính Cursor sao cho khi người dùng rê chuột vào TextBox1 và TextBox2 thì con trỏ sẽ hiển thị ở dạng No

 .Hướng dẫn:

- Viết mã lệnh cho sự kiện MouseMove của TextBox1 như sau:

```
private void txt1_MouseMove(object sender, MouseEventArgs e)
{
    Cursor.Current = Cursors.No;
}
```

- Viết mã lệnh cho sự kiện MouseMove của TextBox2 như sau:

```
private void txt2_MouseMove(object sender, MouseEventArgs e)
{
    Cursor.Current = Cursors.No;
}
```

## CHƯƠNG 11: BÀI TẬP TỔNG HỢP

Câu 1:

Viết chương trình trắc nghiệm giúp học tiếng anh như sau:

- Khi khởi động chương trình sẽ hiển thị Form “Đăng nhập” như hình 11.1



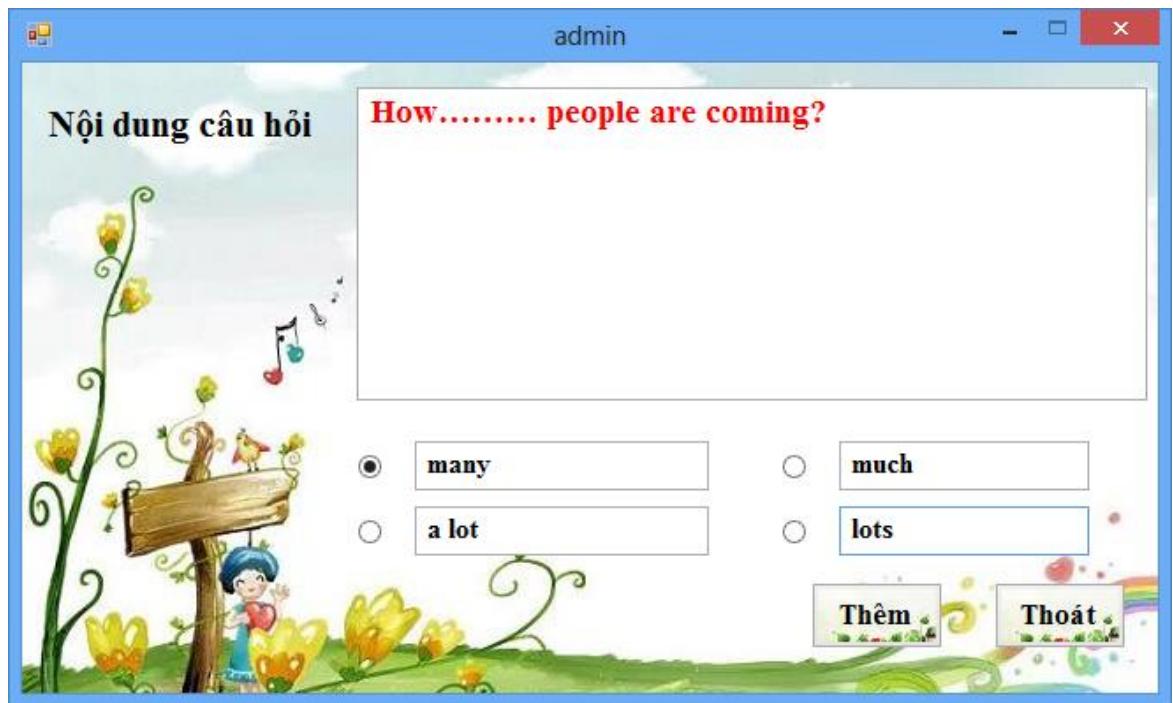
Hình 11.1: Form đăng nhập

- Nhấn Button “Đăng ký” để đăng ký một tài khoản mới đăng nhập vào chương trình trắc nghiệm. Form “Đăng ký” như hình 11.2.



Hình 11.2: Form đăng ký

- o Nếu người dùng đăng ký tên đăng nhập sai thì sẽ hiện ErrorProvide “Tên đăng nhập đã có”.
- o Nhấn Button “Đăng ký” để hoàn tất việc đăng ký tài khoản, khi đó tên đăng ký sẽ được lưu trong tập tin có tên TaiKhoan.txt để lưu trữ.
- Tại form “Đăng nhập” như hình 11.1, nếu người dùng đăng nhập với tài khoản admin và mật khẩu tài khoản là admin thì sẽ hiển thị form “Tạo câu hỏi trắc nghiệm” như hình 11.3.



Hình 11.3: Giao diện form tạo câu hỏi

- Người dùng có thể gõ nội dung câu hỏi và nội dung câu trả lời, đồng thời chọn RadioButton tương ứng với đáp án đúng.
- Nhấn Button “Thêm” để thêm câu hỏi vào tập tin có tên Cauhoi.txt.
- Tại form “Đăng nhập” như hình 11.1, nếu người dùng đăng nhập với tài khoản khác admin và tồn tại trong tập tin Taikhoan.txt thì sẽ hiển thị form “Chọn số câu hỏi trắc nghiệm” như hình 11.4. Nếu tài khoản đang nhập không có trong Takhoan.txt thì hiện thông báo “Bạn đăng nhập chưa đúng tài khoan”.



Hình 11.4: Giao diện form lựa chọn số câu hỏi trắc nghiệm và thời gian làm bài

- Khi người dùng đã chọn xong số câu hỏi trắc nghiệm và thời gian làm bài thì nhấn Enter sẽ hiển thị được form làm bài trắc nghiệm như hình 11.5



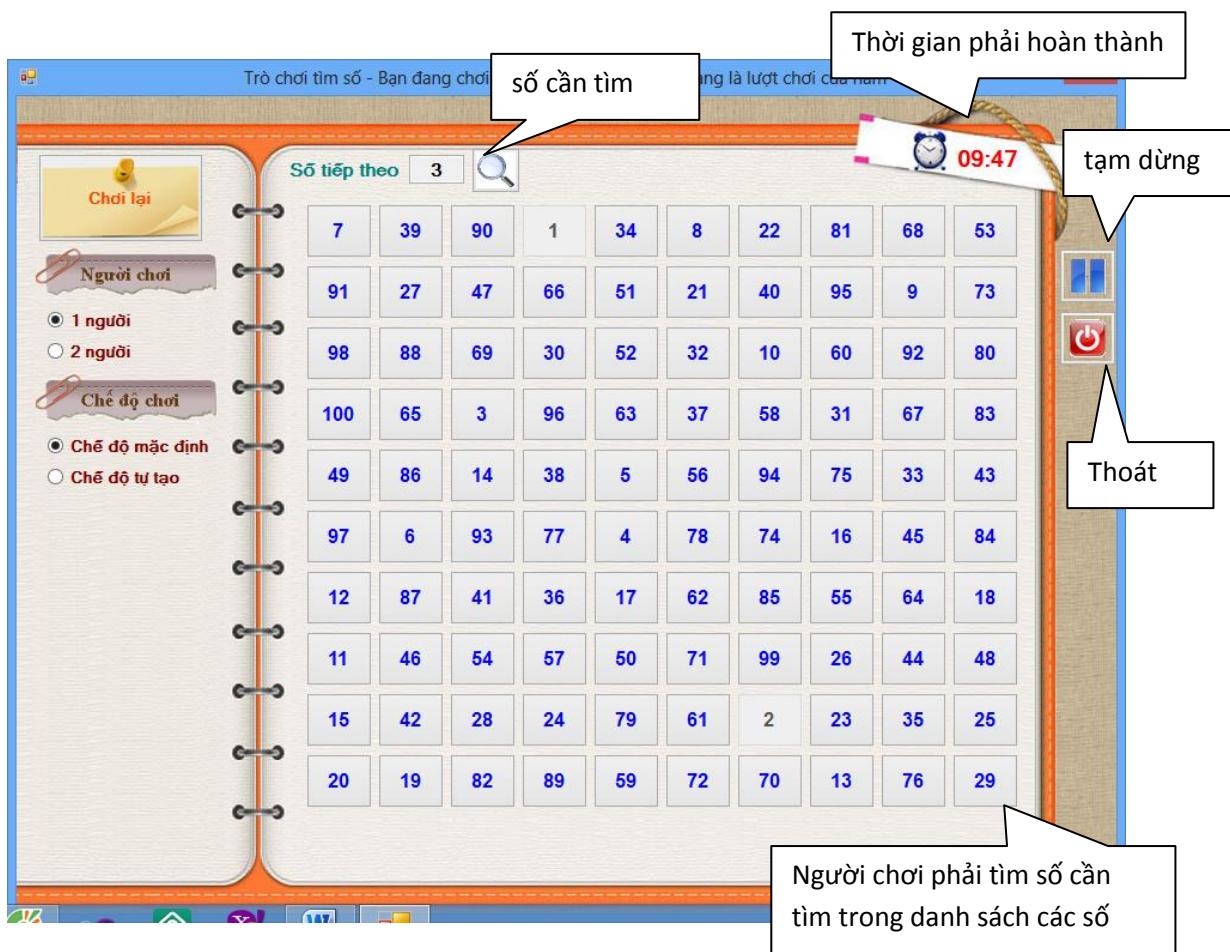
Hình 11.5: Giao diện form làm bài trắc nghiệm tiếng anh

- Bên tay trái giao diện 11.5 sẽ hiển thị các Button, số lượng Button tương ứng với số lượng câu hỏi trắc nghiệm đã chọn. Khi nhấp vào một Button bất kỳ thì sẽ hiển thị lên nội dung câu hỏi, câu hỏi này được lấy từ tập tin Cauhoi.txt.
- Trên giao diện làm bài có thanh ProgressBar sử dụng để mô tả thời gian làm bài mà người dùng chọn. Nếu hết thời gian sẽ tự động nộp bài.
- Người dùng có thể nộp bài trước khi thời gian kết thúc bằng cách nhấn Button “Nộp Bài”. Khi đó sẽ hiển thị giao diện như hình 11.6 để cho biết kết quả làm bài của người dùng.



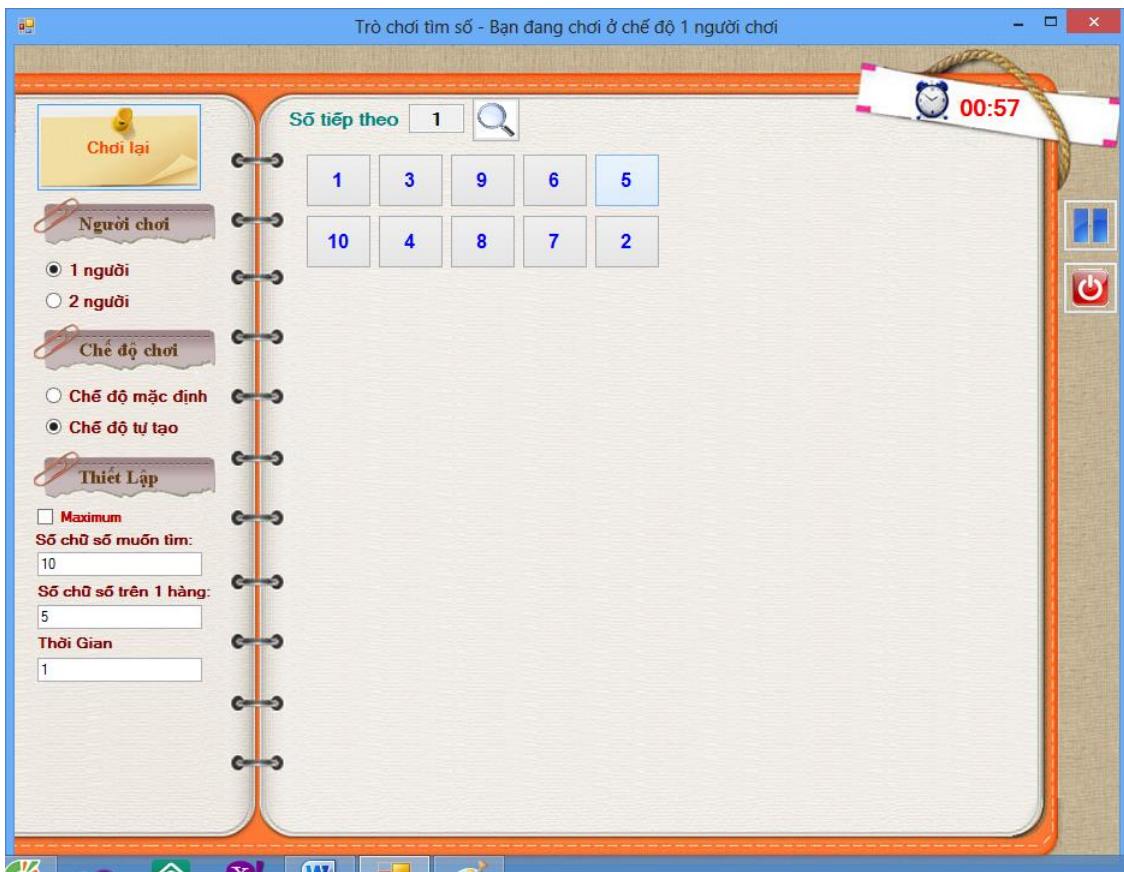
Hình 11.6: Kết quả làm bài của người dùng

Câu 2: Viết chương trình trò chơi tìm số như hình 11.7.



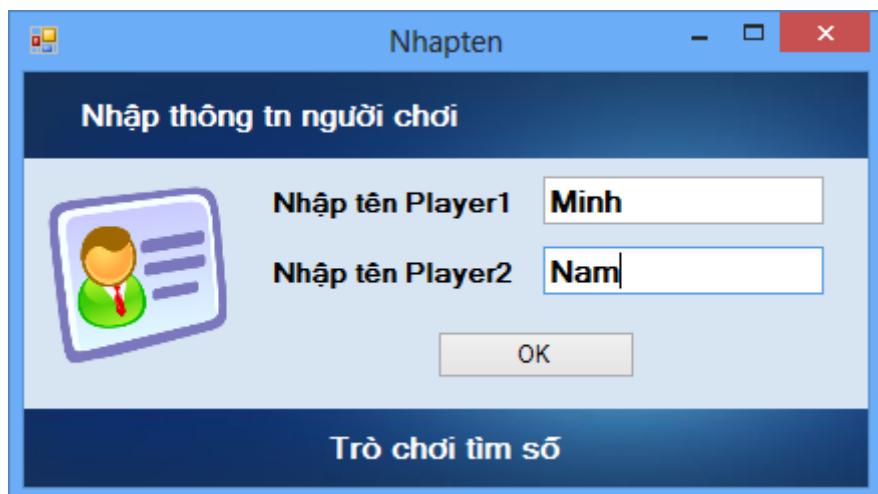
Hình 11.7: Giao diện trò chơi tìm số

- Trong trò chơi tìm số:
  - o Chọn chế độ chơi:
    - Chơi mặc định: khi đó chương trình sẽ hiển thị 100 số ở vị trí ngẫu nhiên bất kỳ. Khi đó người chơi sẽ phải tìm kiếm số cần tìm trong danh sách các số phát sinh ở các vị trí ngẫu nhiên đó. Khi tìm được người chơi sẽ nhấn vào số đó, số đó sẽ lập tức bị chuyển sang chế độ đã được chọn (nghĩa là không còn nhấn được nữa). Thời gian chơi ở chế độ mặc định là 10 phút, nếu hết 10 phút mà không tìm được tất cả 100 số thì người chơi thua cuộc.
    - Chế độ tự tạo: người dùng sẽ chọn số chữ số muốn tìm, số lượng số trên một hàng và thời gian phải hoàn thành việc tìm kiếm. Giao diện như hình 11.8.



Hình 11.8: Giao diện trò chơi chế độ tự chọn

- Chọn số người chơi: Nếu chọn hai người chơi thì sẽ hiển thị form như hình 11.9 để 2 người chơi nhập tên vào.



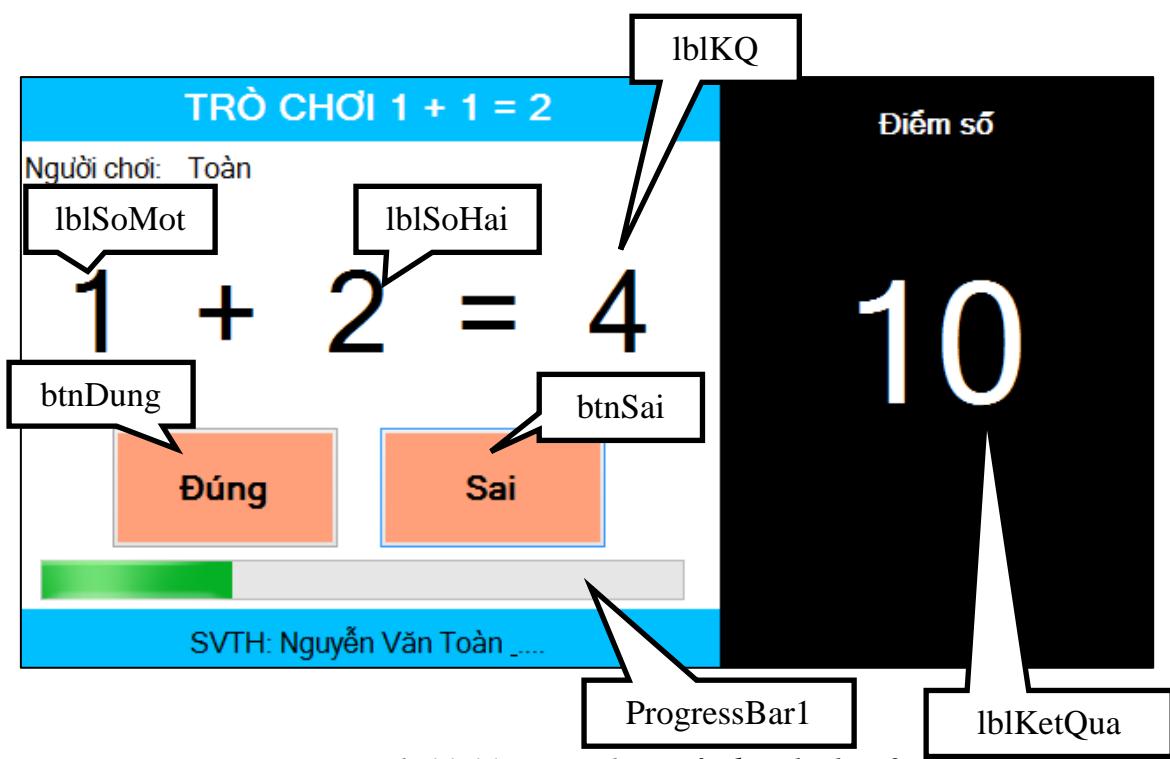
Hình 11.9: Nhập tên của hai người chơi

- Tuần tự người chơi thứ nhất sẽ tìm số trước, thời gian để tìm ra một số là 5 giây, nếu hết 5 giây mà người chơi thứ nhất chưa tìm ra sẽ chuyển lượt cho người chơi thứ hai.
- Với mỗi lần tìm được số thì điểm số của người chơi sẽ tăng lên 1 như hình 11.10



Hình 11.10: Chương trình tìm số hai người chơi

Câu 3: Viết chương trình trò chơi  $1 + 1 = 2$  có giao diện như hình 11.11.



Hình 11.11: Giao diện trò chơi  $1+1 = 2$

Yêu cầu:

- Chương trình sẽ phát sinh ngẫu nhiên các số nguyên trong Label: lblSoMot, lblSoHai, lblKQ.
- Người chơi sẽ xem kết quả phát sinh ở biểu thức và chọn biểu thức đúng hay sai.
  - o Nếu đúng thì dùng chuột chọn nút “Đúng” trên giao diện chương trình hoặc nhấn nút “Left Arrow” trên bàn phím.
  - o Nếu sai thì dùng chuột chọn nút “Sai” trên giao diện chương trình hoặc nhấn nút “Right Arrow” trên bàn phím.
- Mỗi lần chọn đúng thì sẽ được cộng một điểm vào lblKetQua và tiếp tục phát sinh ngẫu nhiên 1 biểu thức mới. Nếu chọn sai sẽ bắt đầu chơi lại trò chơi.
- Thời gian để tính toán và chọn đúng hay sai một biểu thức là 4 giây, được thể hiện ở thanh ProgressBar.

## TÀI LIỆU THAM KHẢO

- [1] Phạm Hữu Khang , *Lập trình Windows Forms, tập 2*, NXB Lao động và Xã hội, 2008.
- [2] Phương Lan (chủ biên), Hoàng Đức Hải, *Lập trình Windows với C#.NET*, NXB Lao động và Xã hội, 2002.
- [3] Mathew A. Stoecker, Steve Stein (2009) - *EXAM 70-505 Windows Forms Application Development - Training Kit*, Microsoft Press, 2009.
- [4] <http://123doc.vn/document/917110-tai-lieu-huong-dan-lap-trinh-vb-net-chuong-18-lam-viec-voi-may-in-pdf.htm?page=6>
- [5] <http://msdn.microsoft.com/en-us/library/ms123401.aspx>
- [6]<http://chienuit.wordpress.com/2010/10/11/parse-tryparse-convert-v-casting-p-ki%E1%BB%83u/>
- [7] [http://csharpvn.com/BaiViet\\_DSBaiViet.aspx?Id=750&Page=2](http://csharpvn.com/BaiViet_DSBaiViet.aspx?Id=750&Page=2)
- [8] <http://doc.edu.vn/tai-lieu/bai-giang-lap-trinh-windows-form-59158/>
- [9] <http://z14.invisionfree.com/dotnet/ar/t6.htm>
- [10]<http://diendan.congdongcviet.com/threads/t18339::application-doevents-tren-csharp-co-y-nghia-gi.cpp>
- [11] <http://hoccungdoanhnhiep.com/tai-lieu/bai-7-dieu-khien-treeview>
- [12] <http://info24h.vn/lap-trinh-xu-ly-giao-dien-trong-winform-voi-c-183.html>
- [13] <http://orchird.wordpress.com/2009/03/30/java/>
- [14] <http://support.microsoft.com/kb/821777/vi-vn>
- [15] <http://tailieu.tv/tai-lieu/chuyen-de-2-cong-nghe-net-6964/>
- [16] <http://teachyourself.vn/tim-hieu-cac-dieu-khien-co-ban/>
- [17] <http://text.123doc.vn/document/2993-tai-lieu-ve-cong-nghe-net.htm>
- [18] <http://text.123doc.vn/document/703165-tai-lieu-ve-cong-nghe-textbox.htm>
- [19] <http://text.123doc.vn/document/702839-dieu-khien-dac-biet.htm>
- [20]<http://text.123doc.vn/document/703163-tai-lieu-ve-cong-nghe-mot-so-dieu-khien-xay-dung-menu.htm>
- [21]<http://thuthuatso1.info/@forum/threads/su-dung-control-treeview-trong-c.14869.html>
- [22] <http://thuthuatso1.info/@forum/threads/bai-19-c-color-dialog-box.14986.html>

- [23] <http://voer.edu.vn/c/ung-dung-windows-voi-windows-form/9f00d32d/43570247>
- [24] <http://www.goclaptrinh.com/cach-chuyen-doi-kieu-du-lieu-trong-c.html>
- [25] <http://www.doko.vn/tai-lieu/bai-giang-ve-lap-trinh-c-157004>
- [26] <http://www.youtube.com/watch?v=UTuI-eWJQUY>
- [27] <http://www.c10mt.com/2011/09/lab-lttq-bai54.html>
- [28] <http://www.hoccungdoanhnhiep.com/tai-lieu/bai-6-dieu-khien-listview>
- [29] <http://www.trithucsangtao.vn/tu-hoc-winform-c-qua-vi-du-listview-449.html>
- [30] <http://www.hoccungdoanhnhiep.vn/tai-lieu/bai-2-dieu-khien-datetimepicker>
- [31] <http://www.codeproject.com/Articles/20050/FireFox-like-Tab-Control>
- [32] <http://www.thuvien-it.net/home/tinhoc/view.asp?threadid=730>
- [33] <http://www.trithucsangtao.vn/tu-hoc-winform-c-qua-vi-du-listview-449.html>
- [34] <http://www.dotnetperls.com/messagebox-show>
- [35] <http://www.slideshare.net/hareesseni/chuong-3-windows-forms>
- [36] [http://www.ddth.com/showthread.php/51653-What-s-Application-DoEventS\(\)](http://www.ddth.com/showthread.php/51653-What-s-Application-DoEventS())
- [37] <http://www.expressmagazine.net/development/717/su-dung-control-listview-trong-c-using-listview-control-c>
- [38] <http://www.trithucsangtao.vn/tu-hoc-winform-c-qua-vi-du-treeview-listview-423.html>
- [39] <http://www.codeproject.com/Articles/91387/Painting-Your-Own-Tabs-Second-Edition>