CLASS 11 (Feb. 14)

Building a Text Editor (Simple NotePad)

Start a new project called TextEditor
Rename the Form frmTextEditor and make it the Startup Form
Add a multiline TextBox - txtEdit

The TextBox normally displays without scrollbars
This can be changed in Design mode or in code
To set properties in code, use the Form Load event
A With block would simplify the coding a bit

```
Private Sub frmTextEditor_Load(...
    txtEdit.Multiline = True
    txtEdit.ScrollBars = ScrollBars.Both
    txtEdit.WordWrap = False
    txtEdit.AcceptsTab = True
End Sub
```

If the WordWrap is left as True (default), the horizontal scrollbar will not appear
Instead, the lines just wrap around
The AcceptsTab property allows the Tab key to work inside the TextBox

Form Location

If you want the Form to appear in a specific location, you can set it using a statement like:

```
Me.Location = New Point(195, 195)
```

The Point arguments are pixels, not twips as were used with VB6
Change the settings to (0, 0) and the Form should be in the top left corner

Form Border Style

There are seven Form Border Styles available for a Form
The border style not only affects the appearance of the Form but also its control buttons in the top right corner

None	No control buttons, no border Of little value
FixedSingle	Normal border, can Minimize or Maximize User cannot control the size
Fixed3D	Same as FixedSingle for functionality Has shadowed border style to simulate 3D
FixedDialog	Same appearance and functionality as FixedSingle Used to create modal dialogs (keep the focus)
Sizable	Has control buttons, can be set to any size

FixedToolWindow	Fixed window, only has Close button – no Min or Max Cannot be resized
SizableToolWindow	Same as FixedToolWindow but can be resized

The two that are most important are:
- Sizable – normal default setting
- FixedDialog – used to create modal or popup forms

The Text Editor should have a sizable window
But the TextBox does not respond to resizing, just the window

Resizing a Control to Fit the Form

Normally, a Windows program can be resized by the user to:
- fill the entire screen
- occupy some part of the screen
- be minimized and sit on the task bar as an icon

You can delete the Max and Min buttons in Design Mode
- MaximizeBox = False
- MinimizeBox = False

Or choose a non-resizable Form Border Style as shown above

But even though we can disable these buttons, we usually don't want to
- resizing and moving windows is something users expect to be able to do

Some controls should be resized to fit the Form window
- e.g., a TextBox control used in a simple text editor (like Notepad)

We will use the Form Resize event rather than Load event
- this way the code will run every time the user resizes the window

The simplest approach is to directly set the location and size to match the Form:

```
Private Sub frmTextEditor_Resize(...

    txtEdit.Size = New Size(Me.Width, Me.Height)

End Sub
```

This is helpful but the bottom line in the TextBox is not fully visible
- This is the old VB 6 approach
- It can be improved by subtracting the height of the Title Bar but there is a better way

You can delete or comment out the Resize event and return to the Load event code
The VB .Net method to fill the form with the TextBox is using the Dock property:

```
txtEdit.Dock = DockStyle.Fill
```

This should work quite well with no letters cut off

Controls can also be docked to just one side, e.g.:

```
txtEdit.Dock = DockStyle.Top
```

The effect may not be exactly what was expected

txtEdit is now the full width of the Form even though it was not docked to either side
only the Height of txtEdit remains constant when the Form is resized

Sometimes it's the border size that you want to remain constant

might hold Buttons, Toolbars, etc.

Add a Button (btnExit) to the bottom right of the Form in Design Mode

Leave a small border around it to the bottom and right

Position txtEdit so there is a small border on the left and top, larger to the bottom and right

Leave room for the Button in the bottom right corner

Run the program again – the Button should be visible on top of the TextBox
This is normally not what we want

Use the anchor property to keep the borders constant
Delete or comment out the existing code in the Load event and use:

```
txtEdit.Anchor = AnchorStyles.Left Or AnchorStyles.Top Or _
    AnchorStyles.Right Or AnchorStyles.Bottom

btnExit.Anchor = AnchorStyles.Right Or AnchorStyles.Bottom
```

Note that the Button is unaffected when we resize using the Left and Top borders
The TextBox is affected whichever border is used for resizing

Delete the Button the related code

Menus

Most Windows products have menus (including Visual Studio)
NotePad has a similar menu, but with fewer options

We will create a similar menu for our text editor, but with even fewer options
The menu structure we want will look like this (items in bold are used later):

File		Edit		Format	Help
New	Ctrl-N	Cut	Ctrl-X	**Font**…	Help Topics
Open…	Ctrl-O	Copy	Ctrl-C	**Color**…	About Editor
Save	Ctrl-S	Paste	Ctrl-V		
Save As…		Delete			
---		---			
Print…	Ctrl-P	Find…	Ctrl-F		

Exit

While not required, there are a few common conventions you should follow when creating menus

- Menu titles are one word only, capitalized
- Menu items should not exceed three words, each one capitalized
- Access and shortcut keys should match other Windows products when applicable
- So should positioning, etc. – make it easy for the users
- Add an ellipsis (…) after items that will need additional information

The MainMenu control is in the ToolBox

- draw it on the Form
- doesn't matter where or what size you make it
- the Menu icon will appear at the bottom below the Form and a Type Here box at the top
- you can rename the MainMenu1 as mmuMain or mnuMain
  - "mmu" refers to Main MenU and then mnu is used for menu items
  - others just use "mnu" for menus, submenus and items

If you don't see the Type Here box, just click the mmuMain control near the bottom

In the "Type Here" box at the top, enter "File"

- Or better still, &File to get the Access key
- A new Type Here box will automatically appear to the right as you add new menu titles
- Add the four titles from the structure above (we are going across first, not down)

If you want to change any of the menu titles, just click on them and change the Properties
All the names should be changed from MenuItem1 to mnuFile, etc.

If they are in the wrong order, just drag them into the correct position

- So you can insert a new title by adding it at the end and then moving it

The next step is to add the items beneath each title
There is also a Type Here box beneath each title

- enter the name of the menu item (not what will appear on the screen)
- e.g., below mnuFile should be mnuFileNew, mnuFileOpen, mnuFileSave, etc.
- once they are named, you can change the Text Property to what you want the user to see
- add the Access Keys where you want them
- the Shortcut key Property should also be set where one is required
- these will not show in Design mode but will appear when you run the program

Note that Access keys do not appear in more recent versions of Windows (2000, XP)
But if you press the Alt key they will appear

Menus also have bars that divide menu items into sections
For example, we want one before and after the Print option
To do this, simply enter a hyphen (-) as the Text
For a name, I would suggest something like mnuFileSep1, mnuFileSep2, etc.

Sep for Separator
sometimes you want to refer to them in code (e.g., hide them)

There are three additional properties that are commonly used
Checked is useful to indicate the state (e.g., Bold could be On or Off)
RadioButton is similar but designed for situations where the user is selecting one of many
This is less common in menus
Enabled can be set to False to gray out an item – usually those that are not relevant
e.g., you don't want Paste to appear until there is something on the Clipboard
Visible can be used to make an option disappear entirely
Useful when certain options are not available to some users
e.g., a Management Report menu could be hidden from most users

Code can also be used for settings
Access keys are normally set in Design Mode because they are part of the text property
But you could change whether a menu item is visible or enabled, or add a Shortcut key

```
mnuUserAdd.Enabled = False
mnuUserAdd.Shortcut = Shortcut.Alt6
```

This is obviously not a standard, or very intuitive, choice of shortcut key!
This works with the 6 in the top row

Context Menus

Windows also support Context Menus
These become visible when you right-click a Form
For example, try right-clicking in NotePad
Six menu options should appear (some may be disabled) with a couple separators:
Undo

Cut
Copy
Paste
Delete

Select All

Context Menus have no Access keys
Even if you try to add them, they will not appear in the menu

While we could create this menu in VB, it's already present for TextBox controls
And no coding is needed – each option already works
So do the standard shortcut keys – try cutting (Ctrl-X) and pasting (Ctrl-V) some text

However, you might want to create a different Context Menu
Start by adding the Context Menu from the Toolbox

It is not located near the Main Menu (even though that would seem to make more sense)
Name it “cmuContext” if you used “mmuMain” or just “mnuContext”

Double click on the Context Menu control and you can edit it much the same as regular menus
Name the menu items as “mnuContextUndo” etc.

Open the code window and the context menu items will appear with the Controls in the left combo box
They are coded the same way as regular menu items

The Context Menu must be attached to the TextBox (or other Control or the Form)
Otherwise you will still see the default Context Menu
Use the ContextMenu property for this purpose

Coding menu items

You can create Click events in Code by just double clicking a menu item
just like creating code for a Button or other controls

If your application has both a Button and menu item that do the same thing:
delete the Button unless users really want it
have one call the other’s procedure – usually code the menu
menus are usually more comprehensive
unusual to have a button with no menu equivalent but the opposite is common
don’t duplicate the code (danger of changing only one of them during maintenance)

Write code for the Exit menu item

```
Private Sub mnuFileExit_Click(...

    Application.Exit
    End

End Sub
```

Building a Toolbar

Toolbars are built as a combination of two controls
need the ImageList control and the Toolbar control

Start by selecting the ImageList icon in the Toolbox and drag an ImageList control to the Form
will appear below the form, not on it
prefix is "ils" (e.g., name it ilsToolbarMain)
some applications have many toolbars and can display several at once
we will do just one
there are also other applications for ImageLists so the name should reflect its role
key property is "Images" - brings up dialog box to add icons
Add “Members” by clicking the Add button

for standard icons, look in Program Files/Microsoft Visual Studio .Net/Common7/
Graphics/Bitmaps/TlBr_W95
examine other Windows packages to see standard orders and arrangements
for example, Word uses the following (plus some additional icons)
New, Open, Save, Print
Cut, Copy, Paste
Bold, Italics, Underline
Left, Center, Right (alignment)
the order does not matter (but it's probably simpler to use the display order)
use can use icons from any source – and even create your own
icons are copied into the program – you do not need to put them in the bin folder

Before drawing the Toolbar, move the TextBox away from the top
you can reposition it just below the Toolbar later

Next step is to create the Toolbar
select the toolbar icon in the Toolbox and draw the Toolbar
will automatically be full width and at the top of the form no matter how you draw it
this can be changed using the Anchor property
"tlb" is MS prefix, some use "tbr"
name it tlbMain

The TextBox txtEdit can now be repositioned back beneath the Toolbar

The first step to creating the Toolbar is to select the Image List
Only one option should appear

Key property is "Buttons" - brings up dialog form to add icons
start by clicking on Add to create the right number of toolbar buttons needed
then click each toolbar button and change some Properties
name should be tlbMainNew, tblMainOpen, etc.
Image Index lets you select the desired image
Text lets you put a word under the image (but this is typically not used anymore)
Put a name in the Tag field (e.g., New, Open, Copy, etc.)
This is needed for the code to be written later
Tool Tip text can be added if desired

To add a separator (blank space between buttons)
add a Button to serve as the Separator placeholder
choose Style Separator
leave the Image Index as None
name it tlbMainSep1 or something else appropriate

You should now be able to view the Toolbar above txtEdit
However, the Toolbar might be covering up the top of txtEdit
Enter some text into the editor

If you don't see anything, press return a few times and try again
If this works, then the top few lines are being covered by the Toolbar

There are a few solutions for this:
One is to create the Toolbar first, and then add txtEdit
The second is right-click txtEdit and then select "Bring to Front"

Both these methods are essentially doing the same thing
If the TextBox is added after the toolbar or is in front of it, it respects the space needed by the toolbar

Another solution is to anchor txtEdit to the Toolbar
To do this, position txtEdit so it fits exactly under the Toolbar and takes up the full size of the Form
Then anchor it to all four sides:

```
txtEdit.Anchor = AnchorStyles.Left Or AnchorStyles.Top Or _
    AnchorStyles.Right Or AnchorStyles.Bottom
```

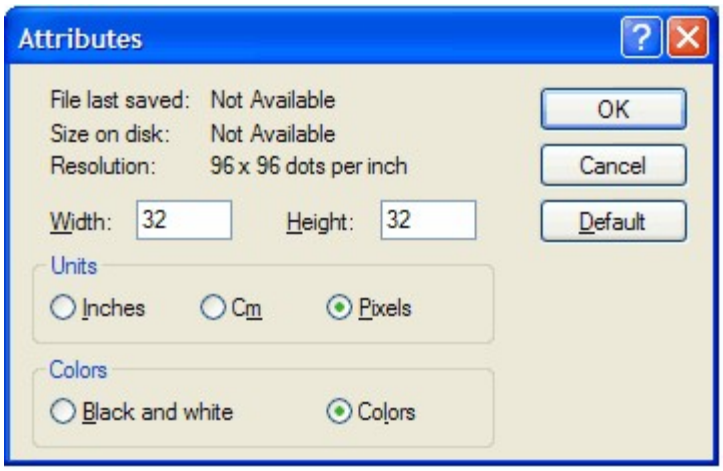
Making Toolbar Icons

There are many icons available through Visual Studio and others on the Internet

This suggestion for making your own comes from Microsoft
It can be useful when clients have a logo or similar image they want to use

It's shockingly easy to create your own icons in Windows XP. Let's do it: Click **Start**, click **All Programs**, click **Accessories**, and then click **Paint**. On the Image menu, click **Attributes**. Type **32** for both the **Width** and **Height** of the document, and make sure that **Pixels** is selected under **Units**. Click **OK** to create a new 32x32-pixel document: the size of an icon.

Now add type, color, or do whatever you'd like to your image. I like to shrink photos (headshots work best) to 32x32 and simply paste them into my Paint document. When you're finished, open the **File** menu and click **Save As**. Use the dialog box to choose where you want to save your file, then give it a name followed by ".ico" (without the quotes), and click **Save**. (The extension ".ico" tells Windows that it's an icon file.) You just created an icon! Now you can change any shortcut or folder to your own icon—just browse to it on your hard drive.



We can also write a VB program that will resize images

Toolbar Code

As with menu items, code must be added for each toolbar button
But there is an important difference
    the event is clicking the Toolbar, not an individual button

Code must then figure out which button was clicked and execute appropriate statements
    usually done with a Select Case statement
    the Case statements refer to the Text or Tag property defined earlier
    you cannot refer to the button name

Normally we code the button to call the corresponding menu event
Since we have no menu events coded yet, this example uses Message Boxes to prove it works

Double click on the Toolbar and this event procedure appears:

```
Private Sub tlbMain_ButtonClick(ByVal sender As System.Object, _
    ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) _
    Handles tlbMain.ButtonClick
```

The key parameter is "e" – it tells us which button was clicked
Normally, each toolbar button calls the corresponding menu function

```
Select Case e.Button.Tag
    Case "New"
        mnuFileNew_Click (sender, e)
    Case "Copy"
        mnuFileCopy_Click (sender, e)
    Case Else
        MessageBox.Show (...
End Select
```

If a Toolbar button has no menu equivalent, you will need to write code for it
If the code required is more than a few lines, create a separate procedure

Enabling/Disabling Toolbar Icons

Disabling the entire toolbar is simple:

```
tlbMain.Enabled = False
```

You can also hide it but it will still occupy the space unless you write more code

```
tlbMain.Visible = False
```

Sometimes you just want to disable one or a few icons and leave the rest enabled
Toolbar Button numbering starts with 0

```
        tlbMain.Buttons(3).Enabled = False
```

The difficulty is knowing which button is which
This is more difficult if you allow the user to add or delete buttons
But even if the user can't change the toolbar, a future programmer might and that should be easy to do
Managing toolbars is easier if they are created in code instead of design mode
    But that goes beyond what we will cover here

Color

We can the Color Dialog box to allow users to select their own preferred colors
To allow them to set the color of the text:

Add the Color Dialog control from the Toolbox and name it dlgColor

```
Private Sub mnuFormatColor_Click(...

    dlgColor.ShowDialog()
    txtEdit.BackColor = dlgColor.Color

End Sub
```

Note that the Color Dialog box always has a color selected so that dlgColor.Color always exists
    The default color is set to match the current color in the code below

Any color can be set this way by choosing the appropriate object and property

There are a few options that can be set to customize the color selection
For example, to use the current color as the default and to prevent the user creating custom colors use:

```
With dlgColor
    .Color = txtEdit.BackColor
    .AllowFullOpen = False
    .ShowDialog()
    txtEdit.BackColor = .Color
End With
```

The Custom Colors button still appears but it is disabled

Fonts

The Font dialog is quite similar to Color, but has more properties to consider

Add the Font Dialog control from the Toolbox and name it dlgFont
The code is very similar:

```
Private Sub mnuFormatFont_Click(...

    dlgFont.ShowDialog()
    txtEdit.Font = dlgFont.Font

End Sub
```

There are additional options, including color
But you have to handle setting the color yourself – it won't happen without the last statement

```
With dlgFont
    .Color = txtEdit.ForeColor
    .ShowColor = True
    .ShowDialog()
    txtEdit.Font = .Font
    txtEdit.ForeColor = .Color
End With
```