# LAB MANUAL

# VISUAL PROGRAMMING LAB

# INDEX

## CSE-409 F Visual Programming Lab.

| L | T | P |
|---|---|---|
| - | - | 3 |

Class Work: 25
Exam: 50
Total: 75
Duration of Exam: 3 Hrs.

### Study of Visual Basic 6.0.NET and Visual C++ 6.0.NET.

1) Study Windows API's. Find out their relationship with MFC classes. Appreciate how they are helpful in finding complexities of windows programming.
2) Get familiar with essential classes in a typical (Document- view architecture) VC++ Program and their relationship with each other.
3) Create an SDI application in VC++ that adds a popup menu to your application which uses File drop down menu attached with the menu bar as the pop-up menu. The pop-up menu should be displayed on the right click of the mouse.
4) Create an SDI application in VC++ using which the user can draw atmost 20 rectangles in the client area. All the rectangles that are drawn should remain visible on the screen even if the window is refreshed. Rectangle should be drawn on the second click of the left mouse button out of the two consecutive clicks. If the user tries to draw more than 20 rectangles, a message should get displayed in the client area that " No more rectangles can be drawn"
5) Create an application in VC++ that shows how menu items can be grayed, disabled and appended at run time.
6) Write a program in VC++ to implement serialization of inbuilt and user defined objects.
7) Write a program in VC++ to create archive class object from CFile class that reads and stores a simple structure (record).
8) Make an Active X control in VC++ derived from a standard control.
9) Write a program in VB to implement a simple calculator.
10) Create a simple database in MS Access Database /Oracle and a simple database application in VB that shows database connectivity through DAO and ADO.
11) Write a simple program that displays an appropriate message when the illegal operation is performed using error handling technique in VB.
12) Write a program in VB to create a notepad.
13) Create a DLL in VB.

## Bright students may do the following exercises:

14) Write a program in VC++ to implement a simple calculator.
15) Write a program in VC++ to create a static link library and a dynamic link library.
16) Create a simple database in MS Access Database and a simple database application in VC++ that shows database connectivity through ADO model.
17) Make an Active X control of your own using VB.
18) With the help of VB, create an object of excel application and implement any action on it.

# HARDWARE REQUIRED

- P-IV/III PROCESSOR
- HDD 40GB
- RAM 128MB or above

# SOFTWARE REQUIRED

- Window 98/2000/ME/XP
- Visual Studio 6.0
- SQL Server/MS-Access

# RATIONAL BEHIND THE VPT LAB

Visual C++ comes within Microsoft Visual Studio .NET 2003. Visual Studio .NET also contains Visual Basic, Visual C#, and Visual J#. Using Visual Studio .NET, you can mix and match languages within one "solution". We will, however, focus on developing C++ code throughout these labs.

Visual Studio .NET is a package that contains all the libraries, examples, and documentation needed to create applications for Windows. Instead of talking about programs (as we did in CS110 and 170), we talk about projects and solutions. Solutions can contain several projects and projects typically contain multiple items or files. For more information about projects

Microsoft Visual C++ provides a powerful and flexible development environment for creating Microsoft Windows–based and Microsoft .NET–based applications. It can be used as an integrated development system, or as a set of individual tools. Visual C++ is comprised of these components:

Visual C++ makes use of this class library and facilitates the programmer by making available a good number of programming tools like menu editor and dialog editor for designing menus and dialog boxes respectively. It takes us away from the conventional method of starting from the scratch and coding similar programs & objects for normally every application. Visual C++ also provides the facility of various wizards that lead us through a step by step process of building the skeleton of application. It also has a set of integrated debugging tools. All these features make Visual C++ a complete programming environment specifically suited for system level applications. Once you start appreciating the great power of Visual C++, you can design your own editors, scribble applications or may be a whole GUI environment in a very short time.

Some of the features of Visual C++ are:

- Code reusability
- Application wizards for MFC applications, DLLs, ActiveX controls, ATL projects, ATL COM Objects and ISAPI extensions
- Integrated development environment
- Components and Controls Gallery to store and access reusable controls and components
- Portability across platforms

In addition to conventional graphical user-interface applications, Visual C++ enables developers to build Web applications, smart-client Windows-based applications, and solutions for thin-client and smart-client mobile devices. C++ is the world's most popular

systems-level language, and Visual C++ gives developers a world-class tool with which to build soft

**Visual C++ 6.0**

Visual C++ 6.0 in particular provides lot of new features, some of which are mentioned below.

. Visual C++ 6.0 provides a whole new set of wizards to build the skeleton applications.

. It provides easier application coding and building and a greater support for ActiveX1 and Internet Technologies.

. **Visual C++ 6.0 components:**

- VC++ Developer Studio - An integrated application, which provides a set of programming-tools.
- VC++ Runtime Libraries – These are the libraries that provide standard functions like *strlen and strcpy* etc. that can be called from a standard C or C++ functions.
- MFC and Template Libraries – The extensive C++ class library especially designed for creating GUI programs.
- VC++ Build Tools – It comprises of C/C++ Compiler, the Linker, resource compiler especially designed for compiling the resources and some other tools required for generating a 32-bit Windows programs.
- ActiveX
- Data Access Components – This includes Database drivers, controls and other tools required by VC++ to interact with Databases.
- Enterprise Tools – These are the advanced level Tools like Application Performance explorer or Visual Studio Analyzer.2
- Graphics – Consist of Bitmaps, Metafiles, Cursors and icons available for inclusion in the application

.

## Objectives:

- After completing this lab, students will upgrade their knowledge in the field of VC++.
- Students will also understand the concepts of Visual Basic programming .
- Getting more knowledge about windows programming.
- It clears the basics concepts of Object Oriented Programming(C++).
- Students will understand and deal with editors,tools,class libraries Debugging techniques and more.

## ***PROGRAM 1***

## ***OBJECT OF THE PROGRAM: Study Window's API and Their Relationship with MFC classes***

API is an acronym for Application Programming Interface. It is simply a set of functions that are part of Windows OS. Programs can be created by calling the functions present in the API. The Programmer doesn't have to bother about the internal working of the functions. By just knowing the function prototype and return value he can invoke the API Functions.

A good understanding of Windows API would help you to become a good Windows programmer. Windows itself uses the API to perform its amazing GUI magic.The Windows APIs are of two basic varieties :

→ API for 16-bit Windows(Win16 API)
→ API for 32-bit Windows(Win32 API)

Each of these have sub-APIs within it. If you are to program Windows 3.1 then you have to use Win16 API, whereas for programming Windows95 and WindowsNT you have to use Win32 API.

Win16 is a 16-bit API that was created for 16-bit processors, and relies on 16-bit values. Win32 is a 32-bit API created generation of 32-bit CPUs and it relies on 32-bit values.

| Win16 API | Win32 API | Description |
|---|---|---|
| USER.EXE | USER32.DLL | The USER components is responsible for window management, including messages, menus, cursors, communications, timer etc. |
| GDI.EXE | GDI32.DLL | The GDI management is the Graphics Device Interface; it takes care of the user interface And graphics drawing, including Windows metafiles , bitmaps, device contexts, and fonts. |
| KRNL386.EXE | KERNEL32.DLL | The KERNEL component handles the low level functions of memory, task, and resource Management that are the heart of Windows. |

The Win16 and Win32 APIs ere similar in most respects, but the Win16 API can be considered as a subset of Win32 API.Win32 API contains almost everything that the Win16 API has, and much more.

At its core each relies on three main components to provide most of the functionality of Windows. These core components along with their purpose are shown in the table given above.

Although the Win16 versions of these componets have .EXE extensions, they are actually all DLLs and cannot execute on their own.

The Win32 API has many advantages, some obvious and others that are not so obvious. The following lists major advantages that applications have when developed with the Win32 API and a 32-bit compiler (such as Microsoft's Visual C++) :

- **True multithreaded applications**.
  Win32 applications support true preemptive multitasking when running on Windows 95 and Windows NT.

- **32-bit linear memory.**
  Applications no longer have limits imposed by segmented memory. All memory pointers are based on the applications virtual address and are represented as a 32-bit integer.

- **No memory model.**
  The memory models (small, medium, large, etc.) have no meaning in the 32-bit environment. This means there is no need for near and far pointers, as all pointers can be thought of as far.

- **Faster .**
  A well-designed Win32 application is generally faster. Win32 applications execute more efficiently than 16-bit applications.

- **Common API for all platforms**. The Win32 API is supported on Windows 95, Windows NT on all supported hardware, Windows CE, and the Apple Macintosh.

**MFC ( Microsoft Foundation's Class)**

The C++ class library that Microsoft provides with its C++ compiler to assist programmers in creating Windows-based applications. MFC hides the fundamental Windows API in class hierarchies, so that, programmers can write Windows-based applications without needing to know the details of the native Windows API.

The MFC classes are coded in C++. It provides the necessary code for managing windows, menus and dialog-boxes. It also helps in carrying out basic tasks like performing basic input-output, storing the collection of data objects etc. It provides the basic framework for the application on which the programmer can build the rest of the customized code.

MFC Library is a collection of classes in hierarchical form, where classes are derived from some base class. For most of the classes provided by MFC, base class is `CObject` from which most of the classes are inherited. The rest few classes are independent classes. The classes in MFC Library can be categorized as:

- Root Class: `CObject`
- MFC Application Architecture Classes
- Window, Dialog, and Control Classes
- Drawing and Printing Classes
- Simple Data Type Classes
- Array, List, and Map Classes
- File and Database Classes
- Internet and Networking Classes
- OLE Classes
- Debugging and Exception Classes

In addition to these classes, the Microsoft Foundation Class Library contains a number of global functions, global variables, and macros.

The Microsoft Foundation Class Library (MFC) supplies full source code. The full source code is supplied in the form of Header files (.H) that are in the MFC\Include directory and implementation files (.Cpp) that are in the MFC\Src directory

MFC Library was especially designed to provide an object oriented interface to Windows, simultaneously providing the compatibility with the windows programs written in C language. The compatibility with C language was required as the Windows was developed long before MFC came into existence and hence all the applications for Windows were initially written in C, a large number of which is still in use.

MFC Library provides the following features:

- Significant reduction in the effort to write an application for Windows.
- Execution speed comparable to that of the C-language API.
- Minimum code size overhead.
- Ability to call any Windows C function directly.
- Easier conversion of existing C applications to C++.
- Ability to leverage from the existing base of C-language Windows programming experience.
- Easier use of the Windows API with C++ than with C.
- Easier-to-use yet powerful abstractions of complicated features such as ActiveX, database support, printing, toolbars, and status bars.
- True Windows API for C++ that effectively uses C++ language features.

**Advantages of MFC**

Microsoft Foundation Class Library greatly reduces the development time, makes the code more portable, provides support without reducing the programming freedom and flexibility and provides easy access to user interface elements and technologies like ActiveX and Internet programming which are otherwise very hard to program.
By using MFC, developers can add many capabilities to their applications in an easy, object-oriented manner. MFC simplifies database programming by providing Data Access Objects (DAO) and Open Database Connectivity (ODBC) and network programming through Windows Sockets.
MFC offers many advantages like:

- Application Framework
- Largest base of reusable C++ source code
- Integration with Visual C++
- Flexible and fast database access
- Support for Internet and ActiveX technology
- Support for messaging API
- Support for multithreading

MFC is not only library of classes. MFC is also an application framework.MFC helps to define the structure of an application and handles many routine chores on the application's behalf.

Starting with CWinApp, the class that represents that represents the application itself, MFC encapsulates virtually every aspect of a program's operation. The framework supplies the WinMain() function, and WinMain() in turn calls the application object's member functions to make the program go. One of the CWinApp member functions

called by WinMain()- Run() – encapsulates the message loop that literally runs the program.

## *PROGRAM 2*

## OBJECT OF THE PROGRAM: Study essential classes in Document View Architecture and Their Relationship with each other

### Parts of Application

The application source code generated for you by AppWizard actually consists of 4 major classes. All these classes can be seen by expanding the class listing in the Class View tab. For each of these classes corresponding header file (.h) and an implementation file (.cpp) is generated.

**The `MainFrame` Window Class**

This class is named `CMainFrame` and is derived from the MFC class `CFrameWnd`. This class manages the main window of the application that contains the window frame, Menu bar, Toolbar, Status bar, System menu and Minimise, Maximise and Close boxes and also contains the view window of the application.

**The Application Class**

This class named `C ProjectNameApp` is derived from the MFC class `CWinApp`. It manages the application as a whole by managing the tasks, like initialising the application instance, managing the message loop and performing final cleanup of the program.

**The Document Class**

This class named `C ProjectNameDoc` is derived from the MFC class `CDocument`. It is responsible for storing the program data as a single string and reading and writing this data to disk files.

**The View Class**

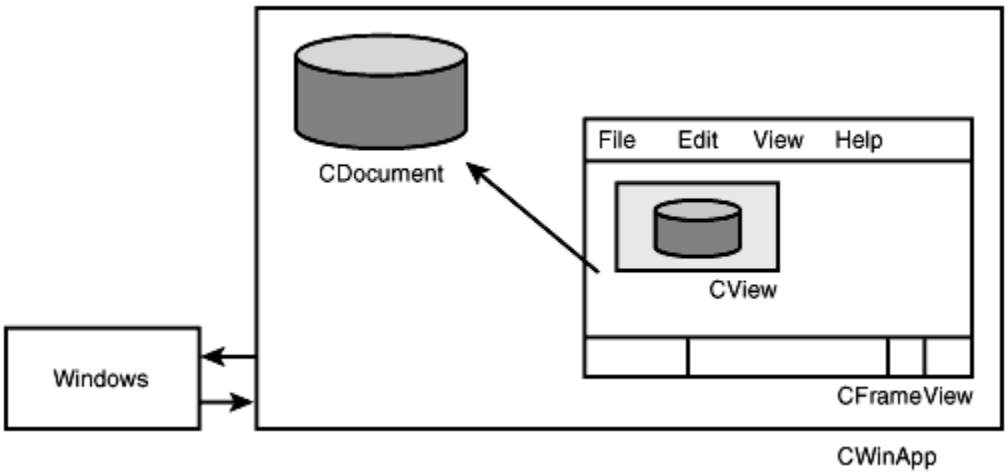
The class named `C projectNameView` is derived from `CView` class from the MFC classes.It manages the view window, which is used to display program data on the screen and also used in processing the input from the user.

MFC and AppWizard use the Document/View architecture to organize programs written for Windows. Document/View separates the program into four main classes:

- A document class derived from `CDocument`
- A view class derived from `CView`

- A frame class derived from `CFrameWnd`
- An application class derived from `CWinApp`

Each of these classes has a specific role to play in an MFC Document/View application. The document class is responsible for the program's data. The view class handles interaction between the document and the user. The frame class contains the view and other user interface elements, such as the menu and toolbars. The application class is responsible for actually starting the program and handling some general-purpose interaction with Windows. Figure shows the four main parts of a Document/View




*The Document/View architecture.*

Although the name "Document/View" might seem to limit you to only word-processing applications, the architecture can be used in a wide variety of program types. There is no limitation as to the data managed by `CDocument`; it can be a word processing file, a spreadsheet, or a server at the other end of a network connection providing information to your program. Likewise, there are many types of views. A view can be a simple window, as used in the simple SDI applications presented so far, or it can be derived from `CFormView`, with all the capabilities of a dialog box. You will learn about form views in Section 23, "Advanced Views."

## SDI and MDI Applications

There are two basic types of Document/View programs:

- SDI, or Single Document Interface
- MDI, or Multiple Document Interface

An SDI program supports a single type of document and almost always supports only a single view. Only one document can be open at a time. An SDI application focuses on a particular task and usually is fairly straightforward.

Several different types of documents can be used in an MDI program, with each document having one or more views. Several documents can be open at a time, and the open document often uses a customized toolbar and menus that fit the needs of that particular document.

## Why Use Document/View?

The first reason to use Document/View is that it provides a large amount of application code for free. You should always try to write as little new source code as possible, and that means using MFC classes and letting AppWizard and ClassWizard do a lot of the work for you. A large amount of the code that is written for you in the form of MFC classes and AppWizard code uses the Document/View architecture.

The Document/View architecture defines several main categories for classes used in a Windows program. Document/View provides a flexible framework that you can use to create almost any type of Windows program. One of the big advantages of the Document/View architecture is that it divides the work in a Windows program into well-defined categories. Most classes fall into one of the four main class categories:

- Controls and other user-interface elements related to a specific view
- Data and data-handling classes, which belong to a document
- Work that involves handling the toolbar, status bar, and menus, usually belonging to the frame class
- Interaction between the application and Windows occurring in the class derived from `CWinApp`

Dividing work done by your program helps you manage the design of your program more effectively. Extending programs that use the Document/View architecture is fairly simple because the four main Document/View classes communicate with each other through well-defined interfaces. For example, to change an SDI program to an MDI program, you must write little new code. Changing the user interface for a Document/View program impacts only the view class or classes; no changes are needed for the document, frame, or application classes.

## Documents

Documents are the basic elements that are created and manipulated by the application. Windows provides a graphic interface that gives a user a natural way to use the application. To implement this interface, the developer has to provide ways to see and interact with the information that the application creates and uses.
A document is simply a place to collect common data elements that form the processing unit for the application.

### Documents and the Application Relationship

The application class uses documents as a way to organize and present information to the user. Each application derived from the MFC defines at least one type of document that is a part of the application. The type of document and the number of documents that the application uses are defined in the code of the application class.
As we have already discussed, MFC supports two types of applications – MDI and SDI.
MDI - In applications that support Multiple Document Interface, a number of text files can be opened for editing at once; each in a different window. Each of the open files has a corresponding document. Also, in MDI, the same document can have multiple views, where a window is split. Each pane of the window can show a different portion of the data whereas this data is coming from a common source, the document.
SDI - In applications with Single Document Interface only one document is open at a time. A SDI application does not have a Window menu and the File menu does not have a Close option because only one document can be open at a time, opening a document automatically closes the current document. These are two common characteristics of a SDI application.

### Why Documents

The fundamental responsibility of the document is to store all the elements that constitute an application unit. An application may support more than one type of data like numbers, text, drawings etc.
The document also controls all the views associated with it. As the user opens and manipulates windows on the screen, views are created and the document is associated

with each view. The document is responsible for controlling and updating the elements within a given view.
The view may either request the document to draw its components or directly request the components to draw themselves. It must provide a device context where the drawing occurs. All elements of a drawing must be able to display themselves correctly upon request.

## Views

The view is the user's window to the document. The user interacts with a document using the view. Each active document will have one or more active views available on the display. Generally, each view is displayed in a single window.

### Why Views

The view gives the document a place to display information. It is the intermediary between the document, which contains the information and the user.
The view organises and displays the document information onto the screen or printer and takes in the user input as information or operation on the document.
The view is the active area of the document. It has two functions –

- → Acts as the display area for the document data and
- → Acts as the input area where the user interacts with the document, normally providing additional commands or data for the document to process.

All the information passes through the view before reaching the document. Therefore, if an application handles mouse messages in functions like `OnLButtonDown()`, it first receives the message, translates the information into an appropriate form and then calls the required document function to process the mouse command.

Although view is responsible for displaying the document and its information, there are two possibilities –

- → Let the view directly access the document's data elements
- → Create a document function that handles access to the appropriate data member

The choice of writing directly in the view has two advantages –

- →It is the responsibility of the view to handle document display, so having the code there is more appropriate
- → Placing the code in the view keeps all the code regarding the device context in the view where it seems natural.

A view is always tied to a single document.
One document can have several views.
Opening another window on the document can create two views of the same document.
Alternative ways can be presented to look at the same information by implementing different types of views for the document. For example, as we discussed in the beginning of the chapter, in MS Excel same data can be viewed either in the form of a table or in the form of different graphs.

## Gaining Access to Document Data from the View

The view accesses its document's data either with the `GetDocument()` function, which returns a pointer to the document or by making the view class a C++ friend of the document class.

The view then uses its access to the data to obtain the data when it is ready to draw or otherwise manipulate it.

For example, from the view's `OnDraw()` member function, the view uses `GetDocument()` to obtain a document pointer. Then it uses that pointer to access a `CString` data member in the document. The view passes the string to the `TextOut()` function.

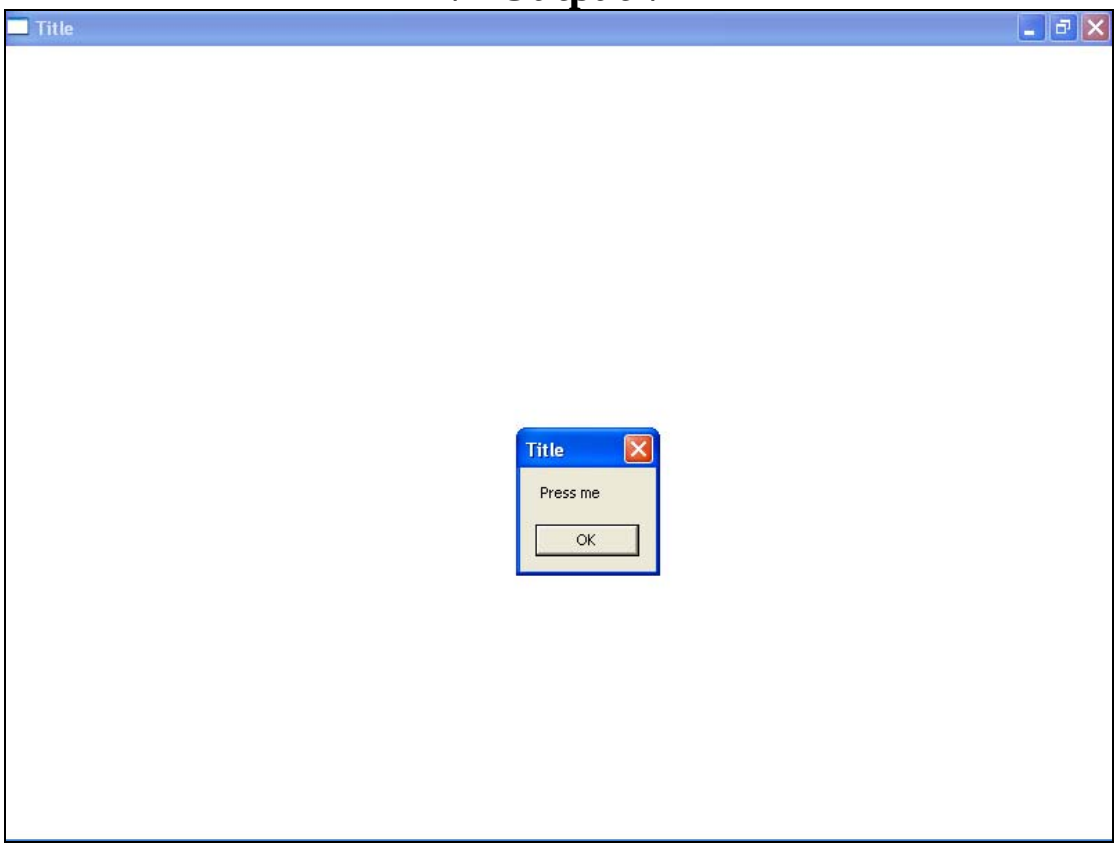
## ***PROGRAM 3***

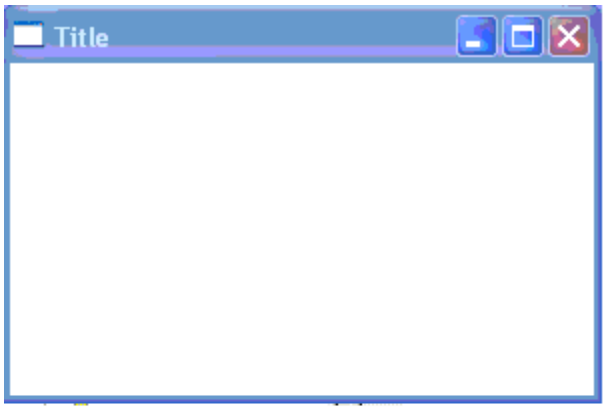
## ***OBJECT OF THE PROGRAM: Create Window & Interact with it***

```
#include<windows.h>

long _stdcall func(HWND,UINT,UINT,long);WNDCLASS a;

int _stdcall WinMain(HINSTANCE i,HINSTANCE j,char *k,int l)
{
        HWND h;
        MSG m;

        a.hInstance=i;                  //instance handle
        a.lpszClassName="sourabh";      //long pointer to class name
        a.lpfnWndProc=func;             //long pointer to Window procedure
        a.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
                                        //set background color of Window Client area

        RegisterClass(&a);              // Register the Window Class before use

        h=CreateWindow("preeti","Title",WS_OVERLAPPEDWINDOW,20,20,300,
        200,0,0,i,0);                   // Create Window Application
        ShowWindow(h,3);

        while(GetMessage(&m,0,0,0))     // extracts message from Message Queue
                DispatchMessage(&m);    // dispatches the message received
        return 0;
}

long _stdcall func(HWND w,UINT x,UINT y,long z)
{
        switch(x)
        {
        case WM_DESTROY:
                PostQuitMessage(0);     // used to terminate while loop in WinMain()
                break;
```

```
        case WM_LBUTTONDOWN:
                MessageBox(0,"Press me","Title",0);          //pops up message box
                break;
        default:

return DefWindowProc(w,x,y,z);
        }
        return 0L;
}
```

## /* Output*/




Title


Title
Press me
OK

## ***PROGRAM 4***

## ***OBJECT OF THE PROGRAM: Draw a free hand drawing as Mouse is Drag***

```
#include<windows.h>

long _stdcall func(HWND,UINT,UINT,long);
WNDCLASS a;
int flag=0,x1,y1,x2,y2;

int _stdcall WinMain(HINSTANCE i,HINSTANCE j,char *k,int l)
{
        HWND h;
        MSG m;

        a.hInstance=i;                  //instance handle
        a.lpszClassName="preeti";       //long pointer to class name
        a.lpfnWndProc=func;             //long pointer to Window procedure
        a.hbrBackground=(HBRUSH)GetStockObject(WHITE_BRUSH);
                                        //set background color of Window Client area

        RegisterClass(&a);              // Register the Window Class before use
        h=CreateWindow("preeti","Title",WS_OVERLAPPEDWINDOW,20,20,300,200,
        0,0,i,0);                       // Create Window Application
        ShowWindow(h,3);

        while(GetMessage(&m,0,0,0))     // extracts message from Message Queue
                DispatchMessage(&m);    // dispatches the message received
        return 0;
}

long _stdcall func(HWND w,UINT x,UINT y,long z)
{
        HDC d;

        switch(x)
        {
        case WM_LBUTTONDOWN:
                if(flag==0)
                {
                        x1=LOWORD(z);
                        y1=HIWORD(z);
                        flag=1;
                }
```

```
            break;

        case WM_MOUSEMOVE:
            if(flag==1)
            {
                x2=LOWORD(z);
                y2=HIWORD(z);
                d=GetDC(w);
                MoveToEx(d,x1,y1,0);  // current position is updated to (x1,y1)
                LineTo(d,x2,y2);          // draw line from current pos to (x2,y2)
                ReleaseDC(w,d);
                x1=x2;
                y1=y2;
            }
            break;

        case WM_LBUTTONUP:
            flag=0;
            break;

        case WM_DESTROY:
            PostQuitMessage(0);  // used to terminate while loop in WinMain()
            break;

        default:
            return DefWindowProc(w,x,y,z);
        }
        return 0L;
}
```

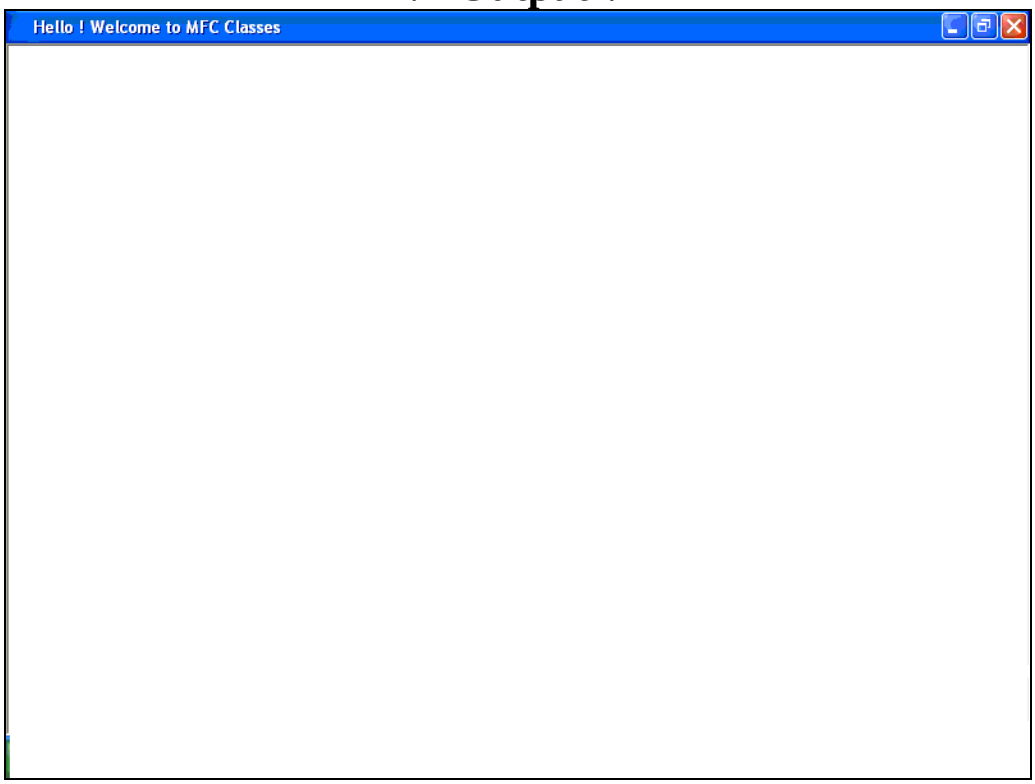
**/* Output*/**

## ***PROGRAM 5***

## ***OBJECT OF THE PROGRAM: Create Window of My Own Class (MFC)***

```
#include<afxwin.h>

class myframe:public CFrameWnd
{
public:
        myframe()
        {
                CString mywindowclass;        //MFC declared string class
                HBRUSH mybrush;

                mybrush=(HBRUSH)::GetStockObject(WHITE_BRUSH);

                mywindowclass = AfxRegisterWndClass(CS_HREDRAW|
                        CS_VREDRAW,AfxGetApp()->LoadStandardCursor(
                        IDC_CROSS),mybrush,AfxGetApp()->LoadStandardCursor(
                        IDI_EXCLAMATION));
                                //register class with windows

                Create(mywindowclass,"Hello ! Welcome to MFC Classes");
        }
};

class myapp:public CWinApp
{
public:
        int InitInstance()
        {
                myframe *p;
                p=new myframe;
                p->ShowWindow(3);
                m_pMainWnd=p;

                return 1;
        }
};

myapp a;
```

# /* Output*/



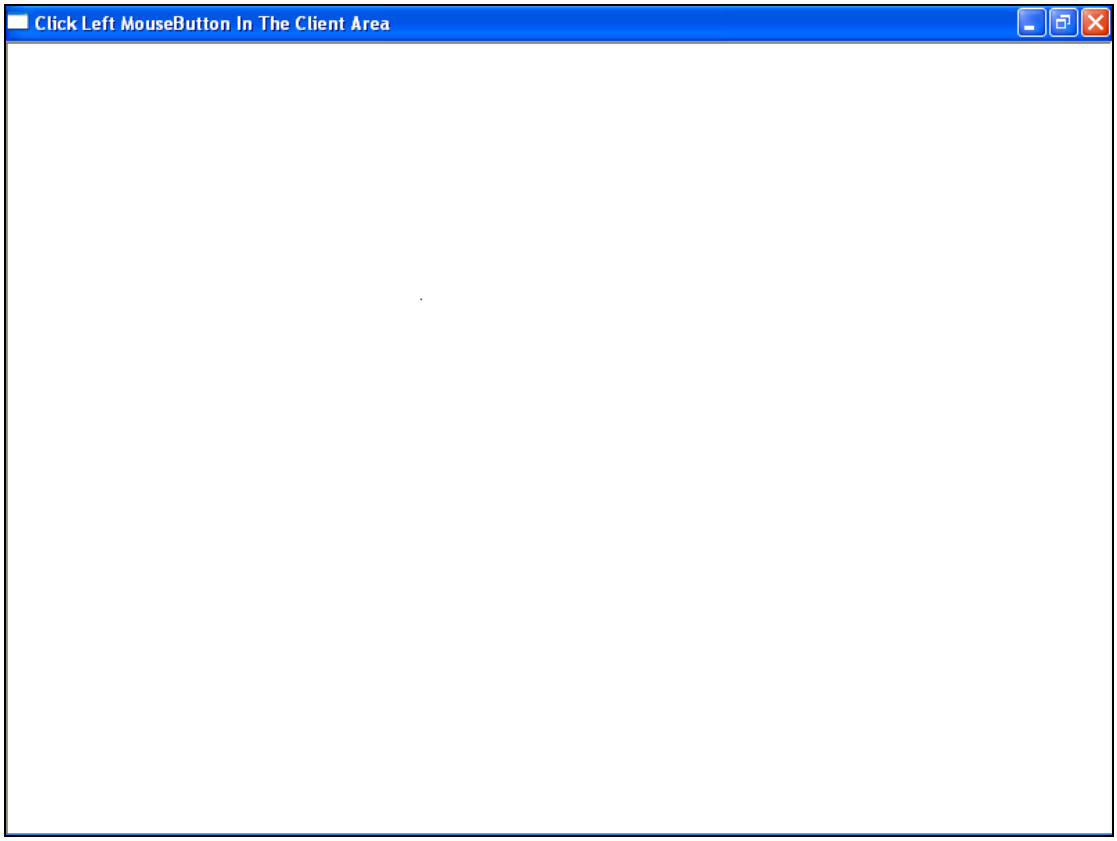
## ***PROGRAM 6***

## ***OBJECT OF THE PROGRAM: Line Drawing Using MFC Classes***

```
#include<afxwin.h>

class myframe:public CFrameWnd
{
private:
        CPoint startpoint,endpoint;

public:
         myframe()
        {
                Create(0,"Click Left MouseButton In The Client Area");
        }

void OnLButtonDown(UINT flag,CPoint pt)
      {
              endpoint=startpoint=pt;
       }

void OnMouseMove(UINT flag,CPoint pt)
      {
              CClientDC d(this);
              if(flag==MK_LBUTTON)
             {
                  d.SetROP2(R2_NOTXORPEN);
                                        // erase line
                  d.MoveTo(startpoint);
                  d.LineTo(endpoint);
                            //draw line
                  d.MoveTo(startpoint);
                  d.LineTo(pt);
                  endpoint=pt;
              }
      }

void OnLButtonUp(UINT flag,CPoint pt)
       {
```

```
                CClientDC d(this);
                 d.SetROP2(R2_COPYPEN);
                 d.MoveTo(startpoint);
                 d.LineTo(endpoint);
        }
DECLARE_MESSAGE_MAP()
};

BEGIN_MESSAGE_MAP(myframe,CFrameWnd)

    ON_WM_LBUTTONDOWN()
    ON_WM_MOUSEMOVE()
     ON_WM_LBUTTONUP()

END_MESSAGE_MAP()

class myapp:public CWinApp
{
public:
        int InitInstance()
        {
                myframe *p;
                p=new myframe;
                p->ShowWindow(1);
                 m_pMainWnd=p;

                   return 1;
         }
};

myapp a;
```

# /* Output*/



## *PROGRAM 7*

## *OBJECT OF THE PROGRAM: Creating a Notepad in VC++*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name Keystrokes in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Single Document` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : CKeystrokesApp, CMainFrame, CKeystrokesDoc, CKeystrokesView .
7. Declare the `StringData` variable on the document's header file, `KeystrokesDoc.h`, in the protected part.

```
class CKeystrokesDoc : public CDocument
{
protected: // create from serialization only
          CKeystrokesDoc();
          DECLARE_DYNCREATE(CKeystrokesDoc)
          CString StringData;
               .
               .
};
```

8. Initialize that string to an empty string -" "- in the document's constructor, which we find in `KeystrokesDoc.cpp`.

```
CKeystrokesDoc::CKeystrokesDoc()
{
          StringData=" ";
          // TODO: add one-time construction code here
}
```

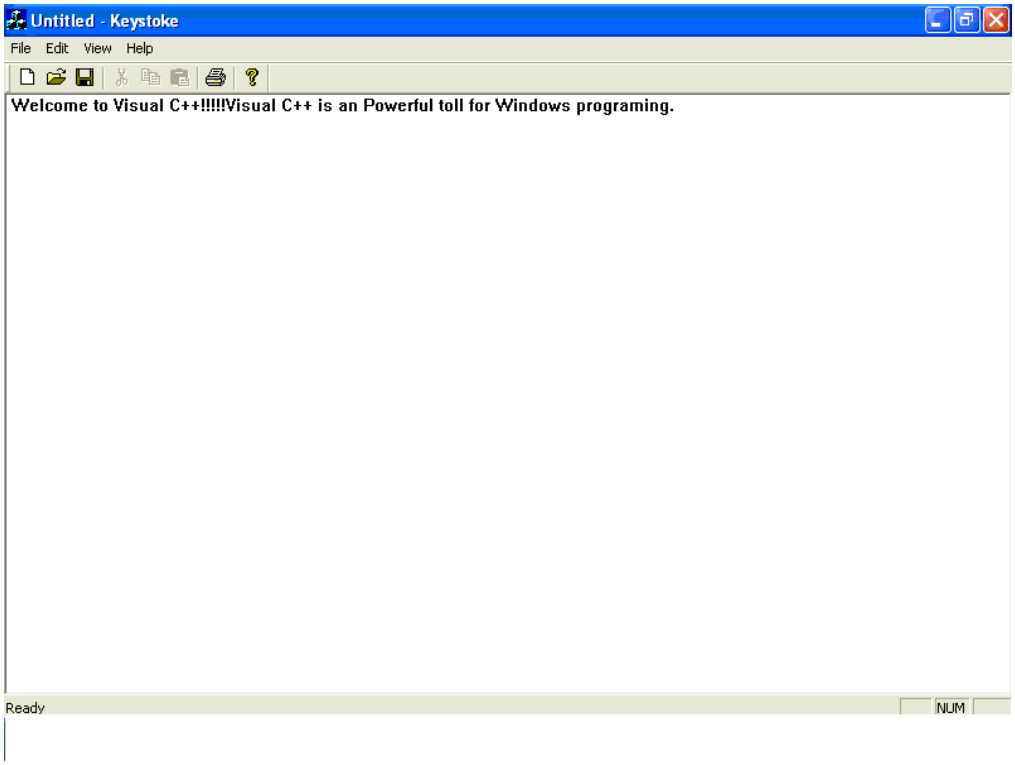
9. Add a new event handler –`OnChar( )`- to our view class which is called every time the user types the character.
10. The character the user typed is now in the `nChar` parameter, which we store in the our data string object, `StringData`. The object in our document, so we need a pointer(`pDoc`) to our document object.
11. Add the character `nChar` to the string `StringData`.

```
void CKeystrokesView::OnChar(UINT nChar, UINT nRepCnt,
     UINT nFlags)
{
```

```
        // TODO: Add your message handler code here and/or call
        default
                  CKeystrokesDoc* pDoc=GetDocument();
                  ASSERT_VALID(pDoc);
                  pDoc->StringData+=nChar;
                  Invalidate();

                  CView::OnChar(nChar, nRepCnt, nFlags);
        }
```

12. We'll handle the display of our data in the view's `OnDraw( )`. We need to draw the text string, which we do with `TextOut( )`.

```
        void CKeystrokesView::OnDraw(CDC* pDC)
        {
                  CKeystrokesDoc* pDoc = GetDocument();
                  ASSERT_VALID(pDoc);

                  pDC->TextOut(0,0,pDoc->StringData);
                  // TODO: add draw code for native data here
        }
```



## *PROGRAM 8*

### *OBJECT OF THE PROGRAM: Creating a Blinking Cursor*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `Carets` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Single Document` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : `CCaretsApp, CMainFrame, CCaretsDoc, CCaretsView .`
7. Declare the `StringData` variable on the document's header file, `CaretsDoc.h`, in the protected part.

```
class CCaretsDoc : public CDocument
{
protected: // create from serialization only
          CCaretsDoc();
          DECLARE_DYNCREATE(CCaretsDoc)
          CString StringData;
               .
               .
};
```

8. Initialize that string to an empty string -" "- in the document's constructor, which we find in `CaretsDoc.cpp`.

```
CCaretsDoc::CCaretsDoc()
{
          StringData=" ";
          // TODO: add one-time construction code here
}
```

9. Add a new event handler –`OnChar( )`- to our view class which is called every time the user types the character.
10. The character the user typed is now in the `nChar` parameter, which we store in the our data string object, `StringData`. The object in our document, so we need a pointer(`pDoc`) to our document object.
11. Add the character `nChar` to the string `StringData`.

```
void CCaretsView::OnChar(UINT nChar, UINT nRepCnt,
     UINT nFlags)
{
// TODO: Add your message handler code here and/or call
default
          CCaretsDoc* pDoc=GetDocument();
          ASSERT_VALID(pDoc);
          pDoc->StringData+=nChar;
```

```
            Invalidate();

            CView::OnChar(nChar, nRepCnt, nFlags);
    }
```

12. Set a boolean variable named `CaretCreated` in view object to keep track that caret has been created or not & `CaretPosition` & `x,y`.

```
        class CCaretsView : public CView
        {
        protected: // create from serialization only
            CCaretsView();
            DECLARE_DYNCREATE(CCaretsView)
            CPoint CaretPosition;
            int x,y;
            boolean CaretCreated;
        // Attributes
            .
            .
        };
```

13. Set `CaretCreated` to false in the View's Constructor.

```
        CCaretsView::CCaretsView()
        {
        CaretCreated=false;
        // TODO: add construction code here
        }
```

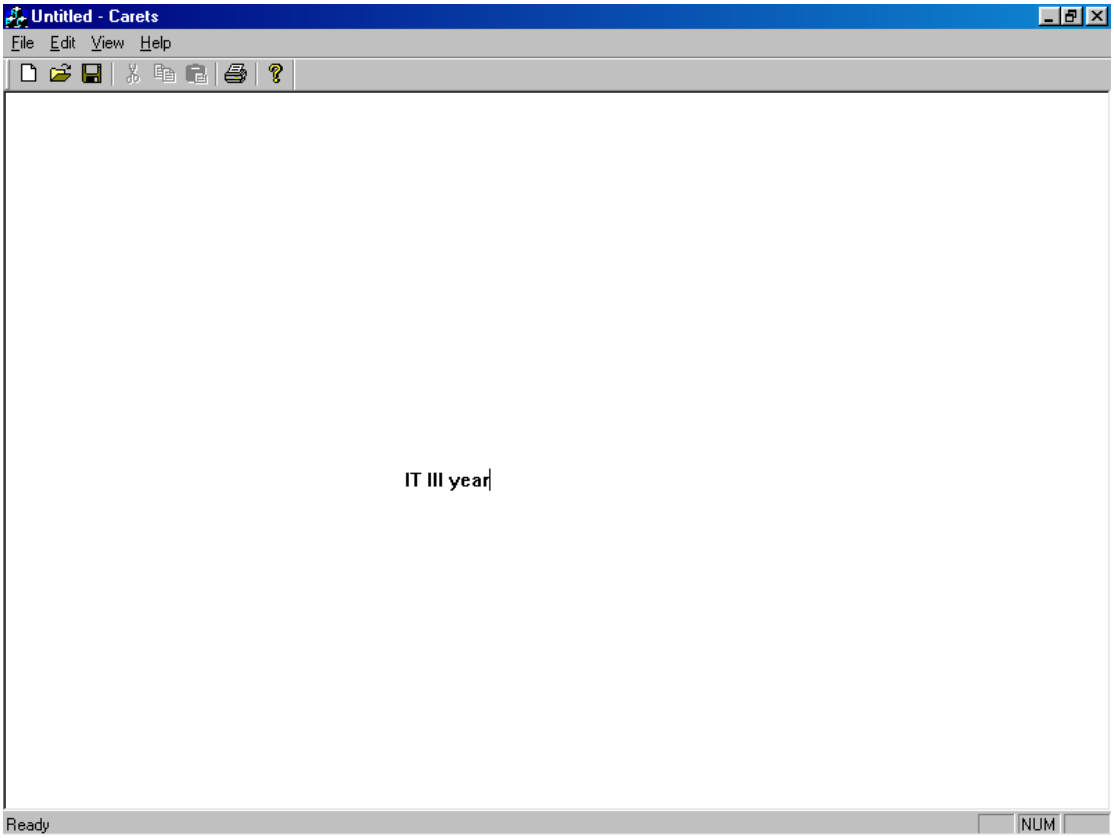
14. We're ready to create our new caret. We'll make the caret the same height and width as our text.We call `CreateSolidCaret()` to actually create the caret.
15. We'll store caret's position in a new `CPoint` object named `CaretPositio`n. The `CPoint` class has two data members `x` & `y` which holds the position of Caret.
16. Initially, set caret's position to (0,0) in OnDraw().
17. We set the caret's position with `SetCaretPos()`, show caret on the screen with `ShowCaret()`, and set the `CaretCreated` Boolean flag to true.
18. In next step the caret is to move as the user types text.
19. Place the caret at the end of displayed text string.
20. To display the caret at the end of the text string, we first hide it using `HideCaret().`
21. Set the `CaretPosition` & Show the Caret.

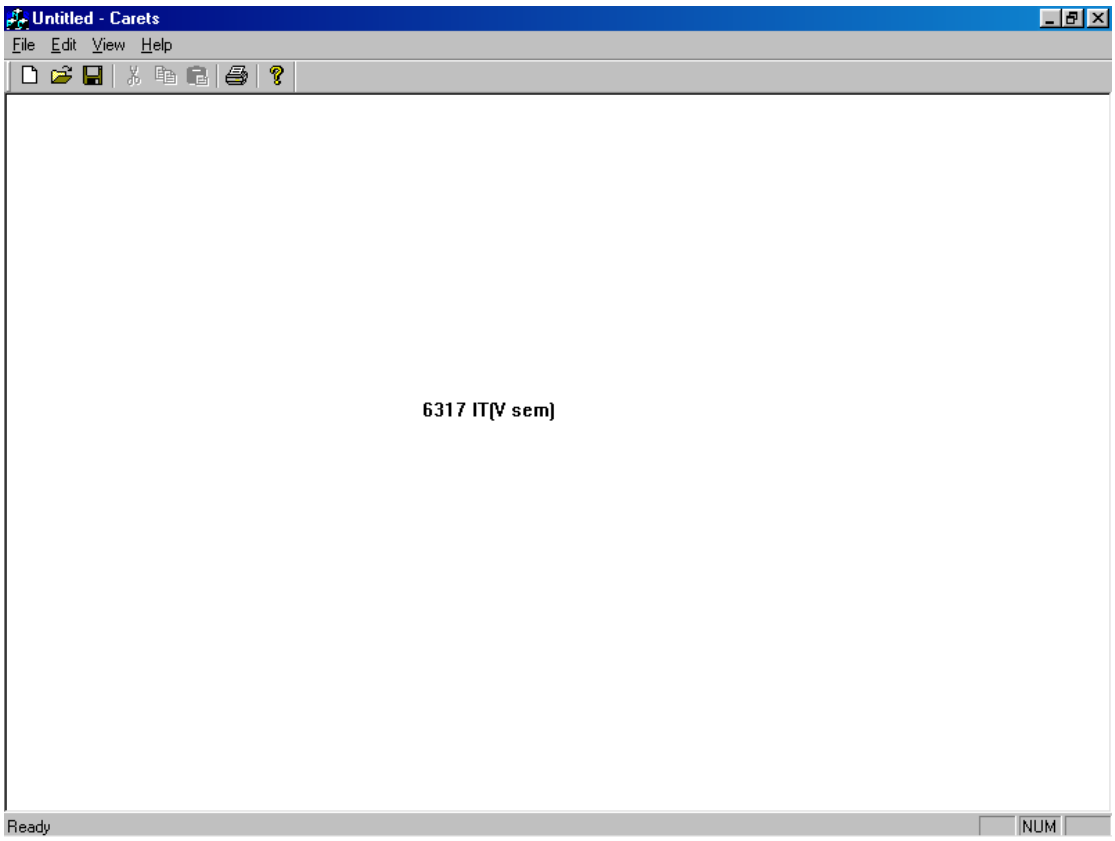
```
        void CCaretsView::OnDraw(CDC* pDC)
        {
            CCaretsDoc* pDoc = GetDocument();
            ASSERT_VALID(pDoc);
            if(!CaretCreated)
            {
                TEXTMETRIC textmetric;
```

```
                pDC->GetTextMetrics(&textmetric);

                CreateSolidCaret(textmetric.tmAveCharWi
                dth/8,textmetric.tmHeight);//(14)
                CaretPosition.x=CaretPosition.y=0;
                     //(16)
                SetCaretPos(CaretPosition);//(17)
                ShowCaret();              //(17)
                CaretCreated=true;        //(17)
           }
           pDC->TextOut(x,y,pDoc->StringData);
                                 //(18)
           CSize size=pDC->GetTextExtent(pDoc
           ->StringData);      //(19)
           HideCaret();        //(20)
           CaretPosition.x=x+size.cx;
           CaretPosition.y=y;
           SetCaretPos(CaretPosition);   //(21)
           ShowCaret();                  //(21)

           // TODO: add draw code for native data here
      }
```

22. To handle left mouse button down we select LButtonDown, point parameter , an Object of the CPoint class, holds mouse's present location.
23. Store the variables in x & y.

```
void CCaretsView::OnLButtonDown(UINT nFlags, CPoint
point)
{
   // TODO: Add your message handler code here
and/or call default
        x=point.x;
        y=point.y;

        CCaretsDoc* pDoc=GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->StringData.Empty();
        Invalidate();
        CView::OnLButtonDown(nFlags, point);
   }
```


Untitled - Carets
File Edit View Help
IT III year
Ready
NUM


Untitled - Carets
File Edit View Help
6317 IT(V sem)
Ready
NUM

## *PROGRAM 9*

## *OBJECT OF THE PROGRAM: Creating a Menu, Dialog Box and Adding Shortcut &Accelerator Keys, Status Bar, Tools to the Menu items*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name DME in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Single Document` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : CDMEApp, CMainFrame, CDMEDoc, CDMEView .
7. Click Resources tab in the VC++ viewer window to open the Menu Editor, find the folder marked Menu & open it. Double click the entry in that folder, IDR_MAINFRAME, Opening the Menu Editor.
8. Add new menu item, Display to the Edit menu.
9. Double click this new menu item, opens the Menu Item Properties box. Place the caption &Display, & makes Display a shortcut menu. In Prompt Box write "Display the message". This text appear in the status bar when user lets mouse rest on this menu item.
10. To add Accelerator folder to Display menu item.Open Accelerator folder from the viewer window.Double-click the IDR_MAINFRAME entry, open Accelerator editor.
11. Double click on the last blank entry, opens the `Accel` Properties box. Select ID , ID_EDIT_DISPLAY & connect Ctrl+Alt+F9(VK_F9).
12. Open ClassWizard, ID_EDIT_DISPLAY is listed in the Object Ids box. Click ID_EDIT_DISPLAY, then click the Command entry in the Message Box. This makes class wizard suggest a name for the event handler of OnEditDisplay()- click OK to accept the name.
13. Declare the `StringData` variable on the document's header file, DME`Doc.h`.

```
class CDMEDoc : public CDocument
{
public:
     CString StringData;
     .
};
```

14. Initialize that string to an empty string -" "- in the document's constructor.

```
CDMEDoc::CDMEDoc()
{
     StringData=" ";
}
```

15. Double click on OnEditDisplay() in the Class Wizard Member functions box,

```
void CDMEView::OnEditDisplay()
{// TODO:  Add your command handler code here
}
```

16. To Create new Dialog Box, select `Resources` item in `Insert` menu. Select `dialog` entry & click the `New` button.
17. Add controls, drag and drop, a button & a text box(or edit box), onto the dialog box.
18. Select the control(button) by clicking & type new caption(Click me) & control ID IDC_BUTTON1.In the same way, text box has IDC_EDIT1.
19. Create class, from `Class Wizard`, click `Add Class`. Type name `Dlg` for new class. Clicking OK opens Class Wizard.Select IDC_BUTTON1 from Object ID & Double click the BN_CLICKED entry in Message Box.
20. Start `Class Wizard`, & click the `Member Variable` tab. Now select text box control,IDC_EDIT1, click add variable button. Give name m_text in the Member variable Name box, Value in Category Box, CString in the Variable text box. Click OK.
21. The variable m_text is connected to the IDC_EDIT1 using special method that class wizard had added to our Dlg dialog box class.

```
void Dlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(Dlg)
    DDX_Text(pDX, IDC_EDIT1, m_text);
    //}}AFX_DATA_MAP
}
```

22. Place text in the text box.

```
void Dlg::OnButton1()
{
    m_text="Welcome to VC++";
    UpdateData(false);
}
```

23. Add code to OK button .

```
void Dlg::OnOK()
{
    UpdateData(true);
    CDialog::OnOK();
}
```

24. To display the dialog box we have to connect it to “Display” item in the Edit menu. Create new object dlg of our dialog box’s class `Dlg`. Use `DoModal()` method, this method returns an integer value, which we stored in results.
25. At this point, dialog box appears on the the screen. If user click OK, string from the text box will be stored in our document. And we place the text in the dialog’s box `m_text` variable in the `StringData` object.

```
void CDMEView::OnEditDisplay()
{
    Dlg dlg;  //24
    int result=dlg.DoModal();  //24
    if(result==IDOK)           //25
```

```
        {
            CDMEDoc* pDoc = GetDocument();
            ASSERT_VALID(pDoc);
            pDoc->StringData=dlg.m_text; //25
            Invalidate();
        }
    }
```

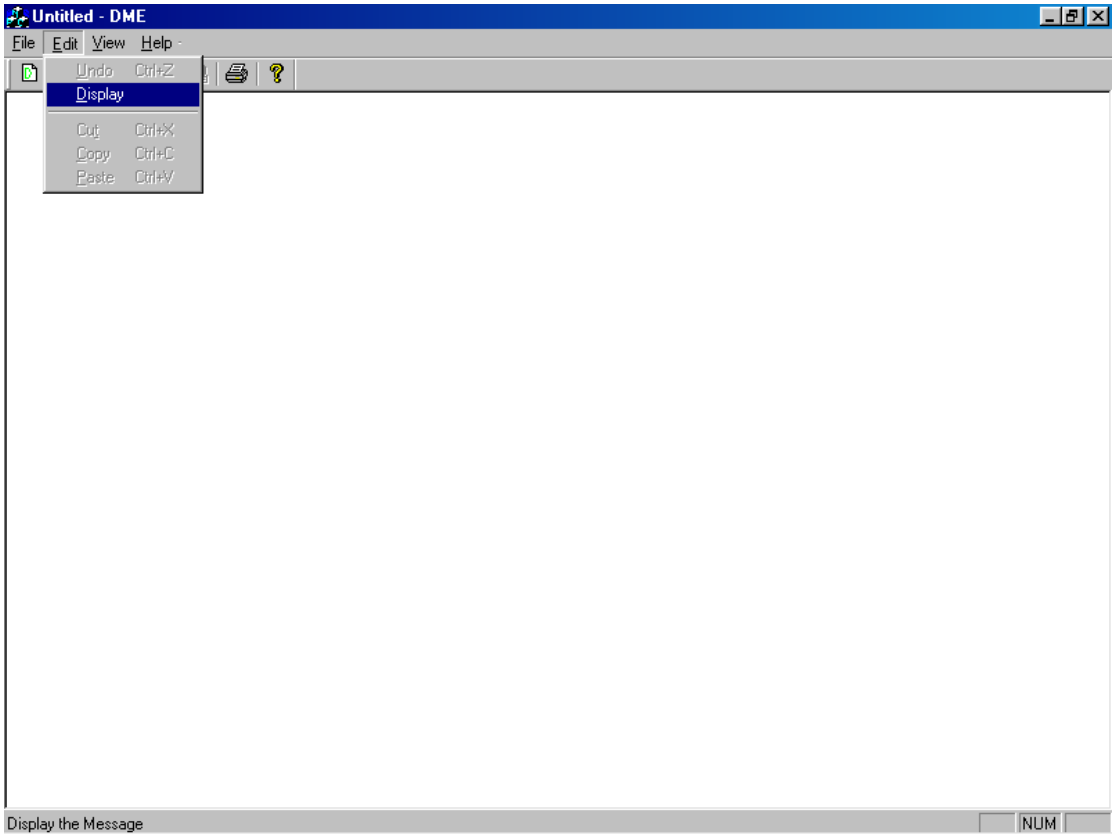
26. In view's OnDraw() method, we draw the text from the dialog box:

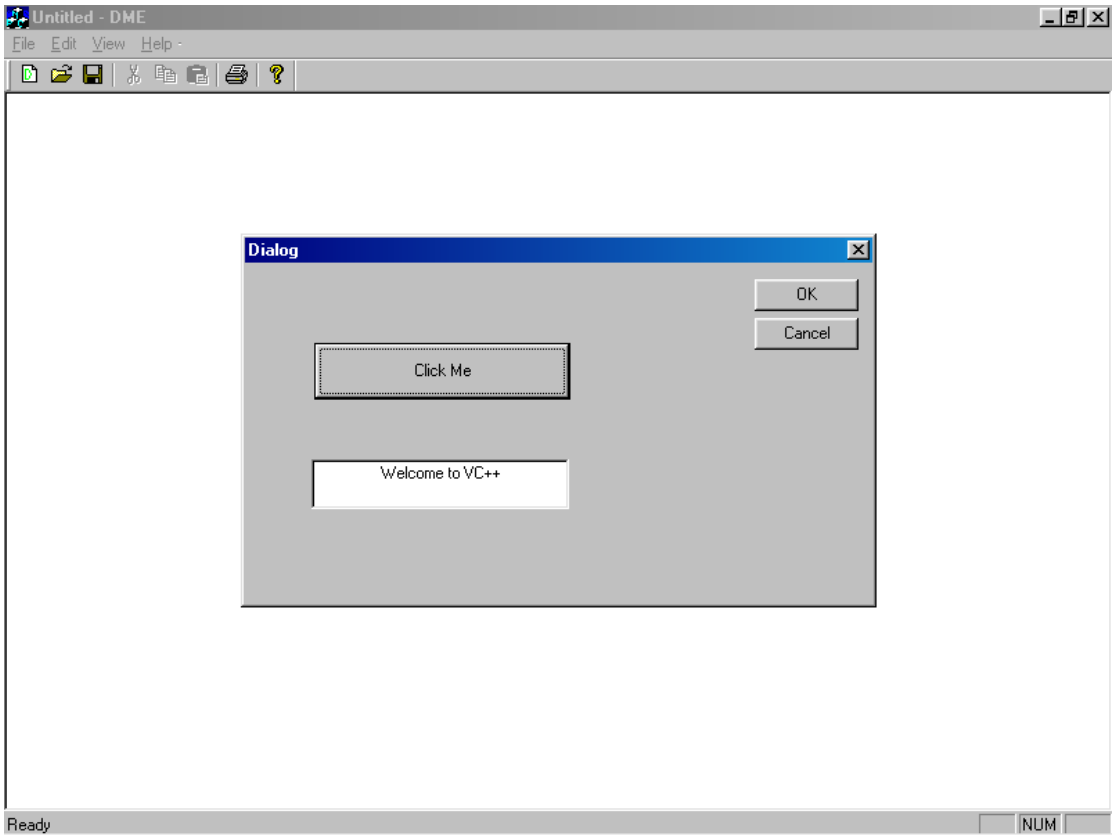
```
void CDMEView::OnDraw(CDC* pDC)
{
    CDMEDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    pDC->TextOut(0,0,pDoc->StringData);
}
```

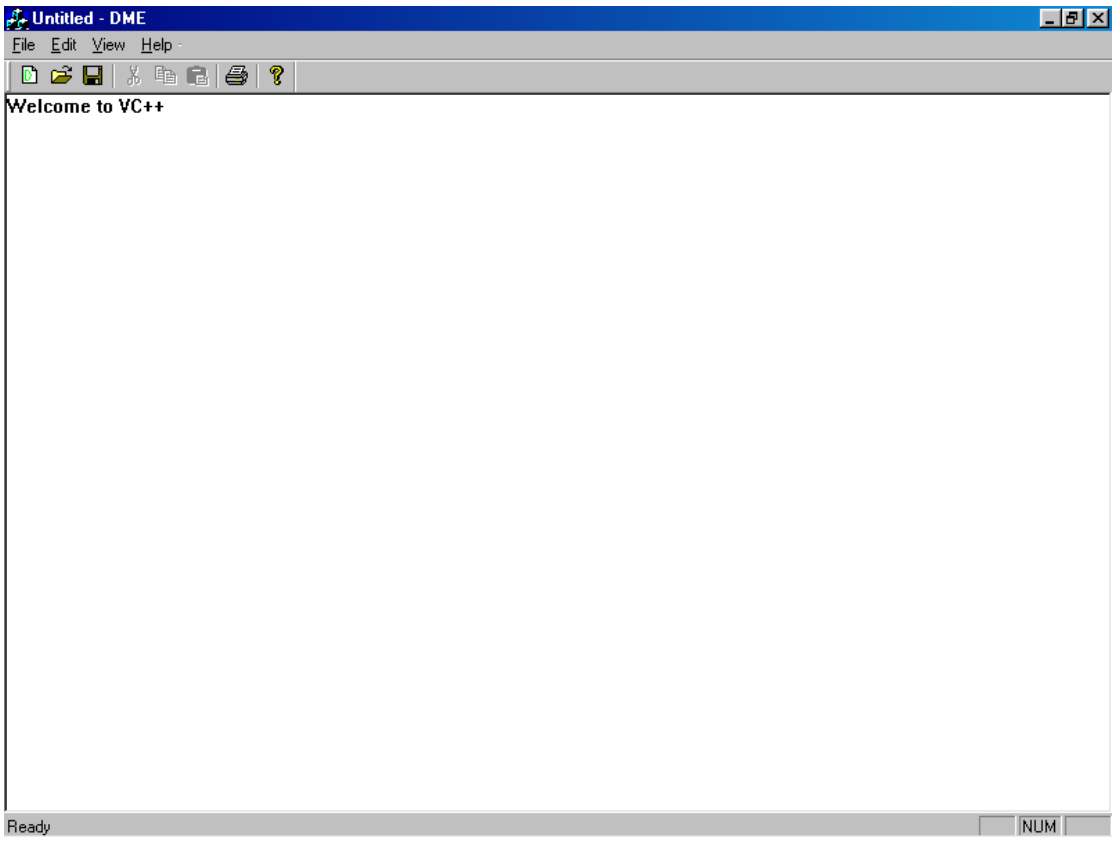
We can Open the item Display from Edit menu by following ways:

(i) Edit→Display.
(ii) Using shortcut key Alt+E then D.
(iii) Using Accelerator key Alt+Ctrl+F9.
(iv) Using first button of the toolbar.

When user highlights a menu item, a message appears at the bar at the bottom of our program's window to provide more information about the item.




Untitled - DME
File Edit View Help -
Dialog
OK
Cancel
Click Me
Welcome to VC++
Ready
NUM


Untitled - DME
File Edit View Help
Welcome to VC++
Ready
NUM

## *PROGRAM 10*

## *OBJECT OF THE PROGRAM: Serializing your own Objects & Class*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name Serialize in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Single Document` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : CSerializeApp, CMainFrame, CSerializeDoc, CSerializeView .
7. Click `Project→Add to Project→New`. A list of file types appear in the dialog box. Choose `C/C++ Header File`(be sure that "`Add to Project`" checkbox is checked. Type file name **CData.h** in the Edit Box.
8. Open the `CData.h` file and place the code :

```
class CData
{
private:
     CString data;
public:
     CData()
     { data=CString(" "); }
     void AddText(CString text)
     { data+=text; }
     void DrawText(CDC* pDC)
     { pDC->TextOut(0,0,data); }
     void ClearText()
     { data=" "; }
};
```

   → `CData()` Constructor to initialize the string to empty .
   → `AddText()` to add more text to the end of string.
   → `DrawText()` to draw string in the device context.
   → `ClearText()` to clear the string.
9. Include this new file in the documents' header & create a new object of `CData` class named `DataObject`.
10. Now we add code to `OnChar()`, which adds the newly typed character to internal data in `DataObject` using the `AddText()` method.

```
void CSerializeView::OnChar(UINT nChar, UINT nRepCnt,
UINT nFlags)
{
     CSerializeDoc* pDoc = GetDocument();
     ASSERT_VALID(pDoc);
```

```
        pDoc->DataObject.AddText(CString(nChar));
        Invalidate();

        CView::OnChar(nChar, nRepCnt, nFlags);
    }
```

11. We draw the text in the `DataObject` object using the `DrawText()` method in the `OnDraw()` method.

```
    void CSerializeView::OnDraw(CDC* pDC)
    {
        CSerializeDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        pDoc->DataObject.DrawText(pDC);
    }
```

12. To make class serializable, add code to `CData` class (i)- make it derived from MFC `CObject` class.(ii)- Also include DECLARE_SERIAL macro to add the declarations of the methods use for serialization. (iii)-Override `CObject`'s Serialize() method.

```
    class CData : public CObject       (i)
    {
        private:
            CString data;
            DECLARE_SERIAL(CData);   (ii)
        public:
            CData()
            { data=CString(" "); }
            void AddText(CString text)
            { data+=text; }
            void DrawText(CDC* pDC)
            { pDC->TextOut(0,0,data); }
            void ClearText()
            { data=" "; }
            void Serialize(CArchive& archive);(iii)
    };
```

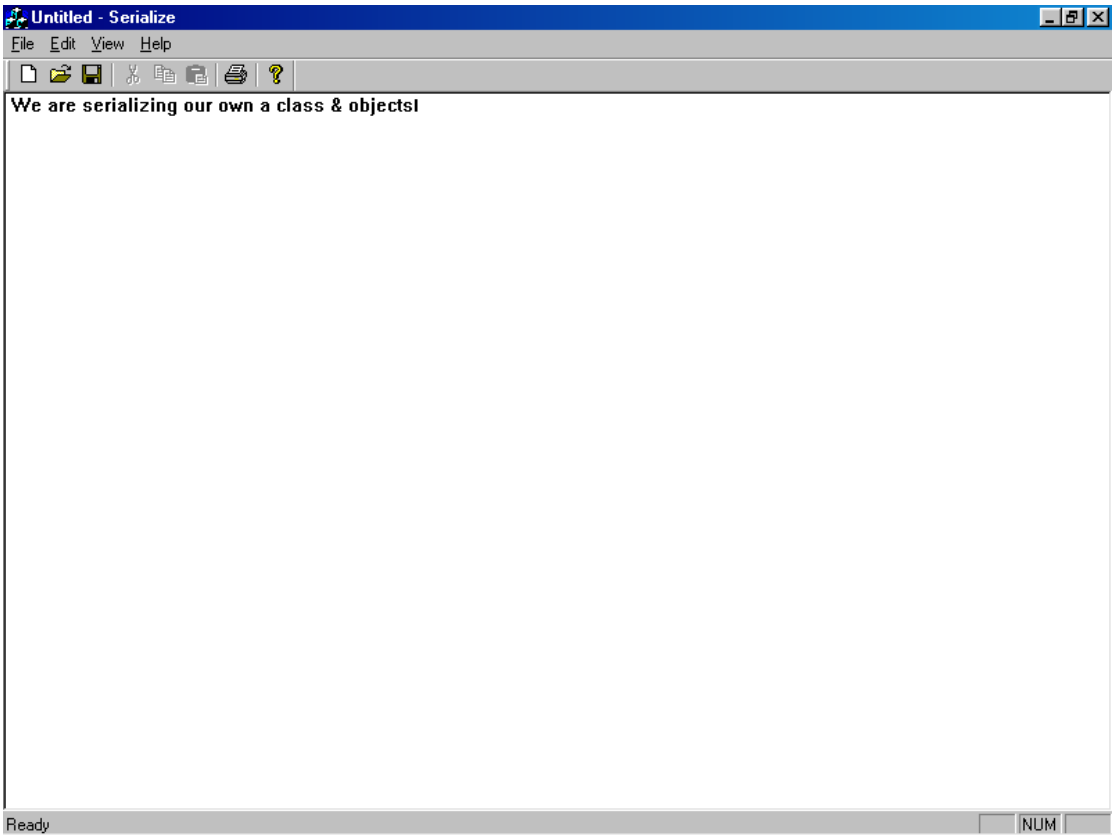
13. To write new version of Serialize, add new file to the project : `CData.cpp`.
    (i) Call base class's `Serialize()` method(). (ii) Serialize our `CString` data member using << & >> . (iii) Include IMPLEMENT_SERIAL macro, which adds additional methods for serialization.

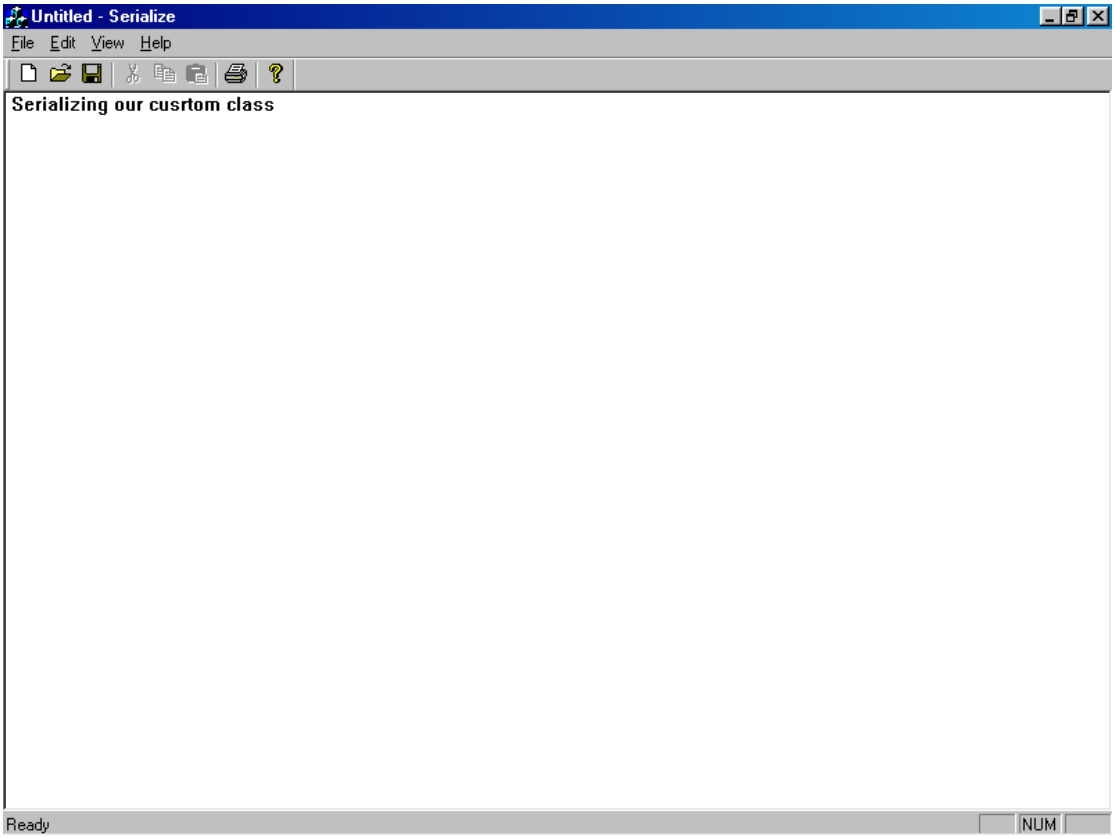
```
    void CData::Serialize(CArchive& archive)  (i)
    {
        CObject::Serialize(archive);
        if(archive.IsStoring())
        {
            archive<<data;                  (ii)
        }
```

```
        else
        {
            archive>>data;
        }
    }
    IMPLEMENT_SERIAL(CData,CObject,0);          (iii)
```

14. To serialize our `DataObject` object, we have to call its `Serialize()` method in document's `Serialize()` method :

```
    void CSerializeDoc::Serialize(CArchive& ar)
    {
        DataObject.Serialize(ar);
    }
```


Untitled - Serialize
File Edit View Help
We are serializing our own a class & objects!
Ready
NUM


Untitled - Serialize
File Edit View Help
Serializing our cusrtom class
Ready
NUM

## *PROGRAM 11*

## *OBJECT OF THE PROGRAM:FILE HANDLING: Create,Open ,Read,Write,Modify and Close a file*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name file in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Single Document` in the AppWizard, click `Finish`.
6. Take character strings in array of strings named `OutString[] & InString[]` .

```
// fileDlg.h : header file
     .
     .
protected:
     HICON m_hIcon;
     char OutString[4][20];
     char InString[20];
```

7. Place the strings use in four character arrays. We'll use standard C `strcpy()` function to fill each character array in `OnInitDialog()`.
8. Add two text boxes and a button. Connect the member variable m_text1 to text in the top text box, and m_text2 to the text in the bottom text box.We send out the string in the top text box.

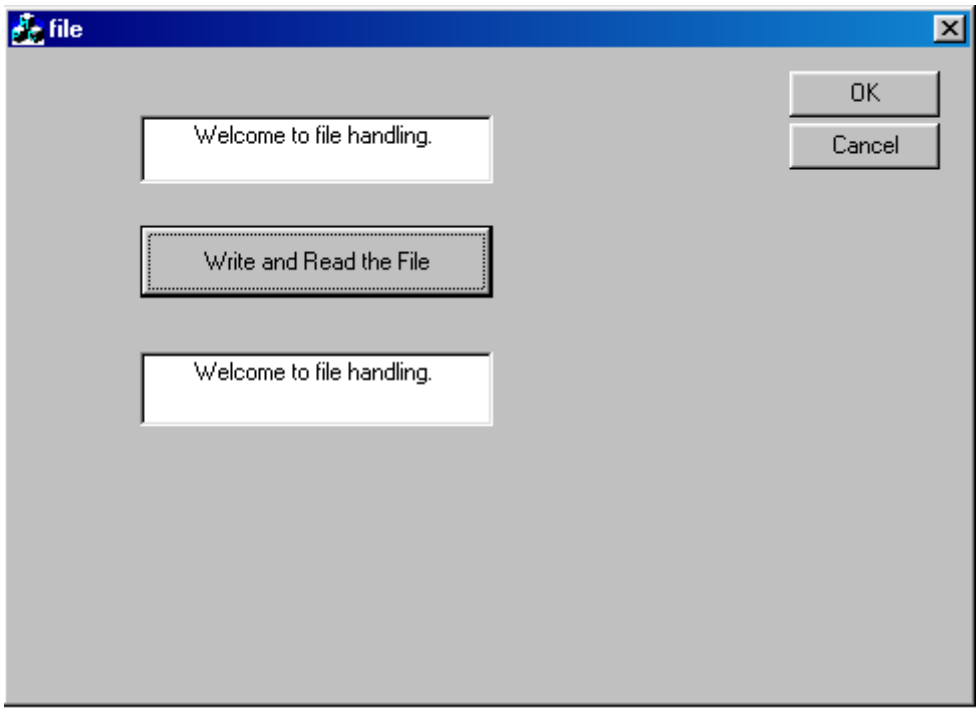
```
BOOL CFileDlg::OnInitDialog()
{
     CDialog::OnInitDialog();
     strcpy(OutString[0],"Welcome ");    // 7
     strcpy(OutString[1],"to ");
     strcpy(OutString[2],"file ");
     strcpy(OutString[3],"handling. ");

     m_text1=CString(OutString[0])+CString(
     OutString[1])+CString(OutString[2])+
     CString(OutString[3]);              // 8
     UpdateData(false);
     .
     .
     .
}
```

9. To write four character arrays we use `CFile` class and create object of that class named `OutFile` in `OnButton1()`.
10. File is open and ready to write to.We loop over four strings, writing them to the file

with `CFile Write()` method.

11. To read the character strings in data.txt back in, we'll use `CFile Read()` method. We create new `CFile` object `InFile` to read the file.
12. We loop over four string. We use `CFile Seek()` method to position the file pointer at the beginning of each successive string.
13. This method returns the num of characters actually read, and we store that value in the integer NumChar.
14. The record we just read in is in the character array InString[]. We add that record to the text in the second text box & then closes the file data.txt.

```
void CFileDlg::OnButton1()
{
     CFile OutFile("data.txt",CFile::
     modeCreate|CFile::modeWrite);    // 9
     for(int i=0;i<4;i++)
     {
          OutFile.Write(OutString[i],20); //10
     }
     OutFile.Close();

     CFile InFile("data.txt",CFile::modeRead);
                                        //11
     for(i=0;i<4;i++)                   //12
     {
          InFile.Seek(20*i,CFile::begin);  //12
          int NumChar=InFile.Read(InString,20);
                                             //13
          m_text2+=CString(InString);   //14
     }
     UpdateData(false);                 //14
     InFile.Close();
}
```



## *PROGRAM 12*

## *OBJECT OF THE PROGRAM: Creating a Multiple Document Interface*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `Multiview` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Multiple Document` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : CKeystrokesApp, CMainFrame, CKeystrokesDoc, CKeystrokesView .
7. Declare the `StringData` variable on the document's header file, `MultiviewDoc.h`, in the protected part.

```
class CMultiviewDoc : public CDocument
{
          CString StringData;
               .
               .
};
```

8. Initialize that string to an empty string -" "- in the document's constructor, which we find in `MultiviewDoc.cpp`.

```
CMultiviewDoc::CMultiviewDoc()
{
          StringData=" ";
}
```

9. We have to store data on the disk:

```
void CMultiviewDoc::Serialize(CArchive& ar)
{
     if (ar.IsStoring())
     {
          ar<<StringData;
     }
     else
     {
          ar>>StringData;
     }
}
```

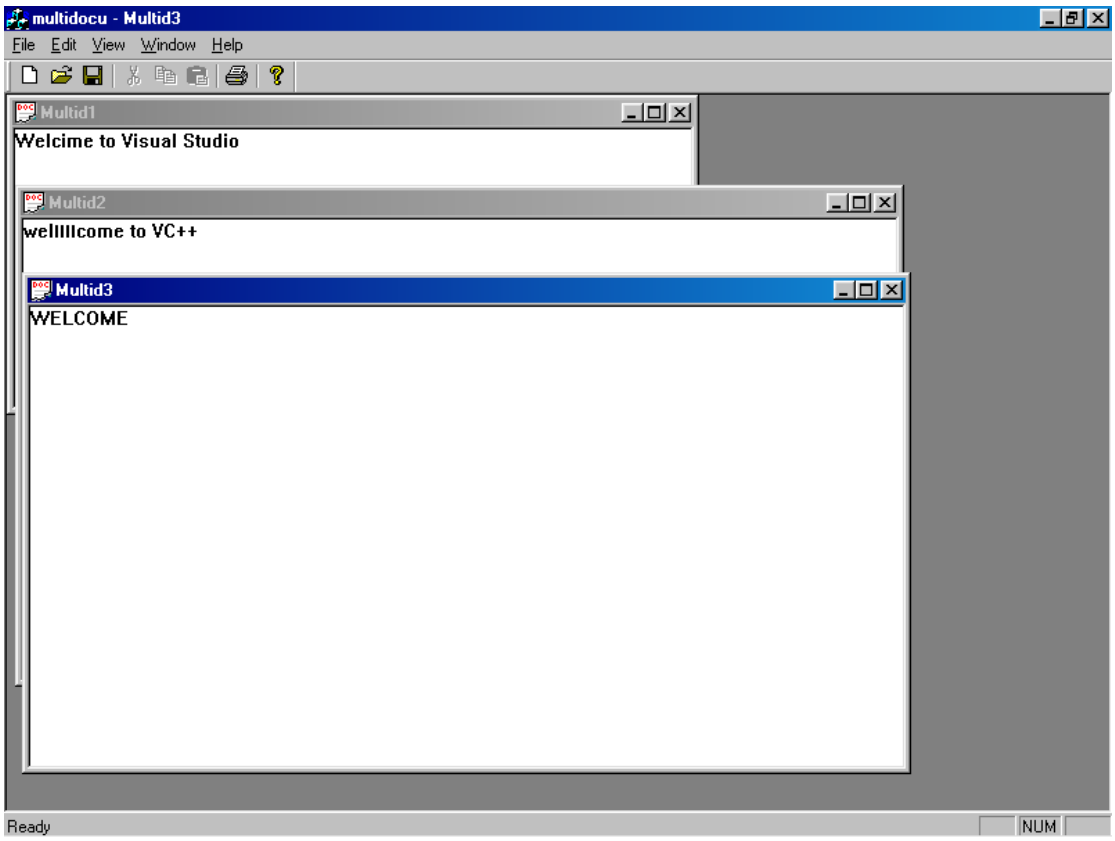
10. Add a new event handler –`OnChar( )`- to our view class which is called every time the user types the character.
11. The character the user typed is now in the `nChar` parameter, which we store in the our data string object, `StringData`. The object in our document, so we need a pointer(`pDoc`) to our document object.

12. Add the character `nChar` to the string `StringData`.
13. We'll handle the display of our data in the view's `OnDraw( )`. We need to draw the text string, which we do with `TextOut( )`.

```
void CMultiviewView::OnDraw(CDC* pDC)
{
          CMultiviewDoc* pDoc = GetDocument();
          ASSERT_VALID(pDoc);

          pDC->TextOut(0,0,pDoc->StringData);
}
```

14. Using `UpdateAllViews()` method, we're able to refresh all the views into this document now that user has typed a new character.
15. Set the document's modified flag using `SetModifiedFlag(),` means that if user tries to close the current document without saving it, the program will ask if they want to save it before the data in it is lost.

```
void CMultiviewView::OnChar(UINT nChar, UINT nRepCnt,
     UINT nFlags) //10 11
{
          CMultiviewDoc* pDoc=GetDocument();
          ASSERT_VALID(pDoc);
          pDoc->StringData+=nChar; //12
          Invalidate();
          pDoc->UpdateAllViews(this,0L,NULL);//14
          pDoc->SetModifiedFlag();      //15
          CView::OnChar(nChar, nRepCnt, nFlags);
}
```


multidocu - Multid3
File Edit View Window Help
Multid1
Welcime to Visual Studio
Multid2
wellllIcome to VC++
Multid3
WELCOME
Ready
NUM

## *PROGRAM 13*

## *OBJECT OF THE PROGRAM: Creating a Dynamic Link Library*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(dll)` entry in the `New` dialog box.
3. Give the new program the name `dll` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Open dll.cpp file and write the function that is to be exported here,

```
int _stdcall add(int a,int b)
{ return(a+b); }
```

6. In the dll.def file,

```
EXPORTS
     add @ 1
        ; Explicit exports can go here
```

7. Compile the application. dll.dll file is created and will be found in your debug folder.

8. Copy the dll.dll file from debug folder of dll.And paste into debug folder of view application wizard(exe).
9. In the view.cpp file under #include statements write the prototype of the functions-

```
typedef int(CALLBACK *lpfn) (int,int);
```

10. In OnDraw() function –

```
void CViewView::OnDraw(CDC* pDC)
{
     CViewDoc* pDoc = GetDocument();
     ASSERT_VALID(pDoc);
     HINSTANCE hDll;
     int sum;
     lpfn f1;
     hDll=LoadLibrary("dlll");
     if(hDll!=NULL)
     {
          f1=(lpfn)GetProcAddress(hDll,"add");
          if(!f1)
               FreeLibrary(hDll);
          else
          {
               sum=f1(3,4);
               char ch[2];
               sprintf(ch,"%d",ch);
               pDC->TextOut(10,10,sum);
}
```

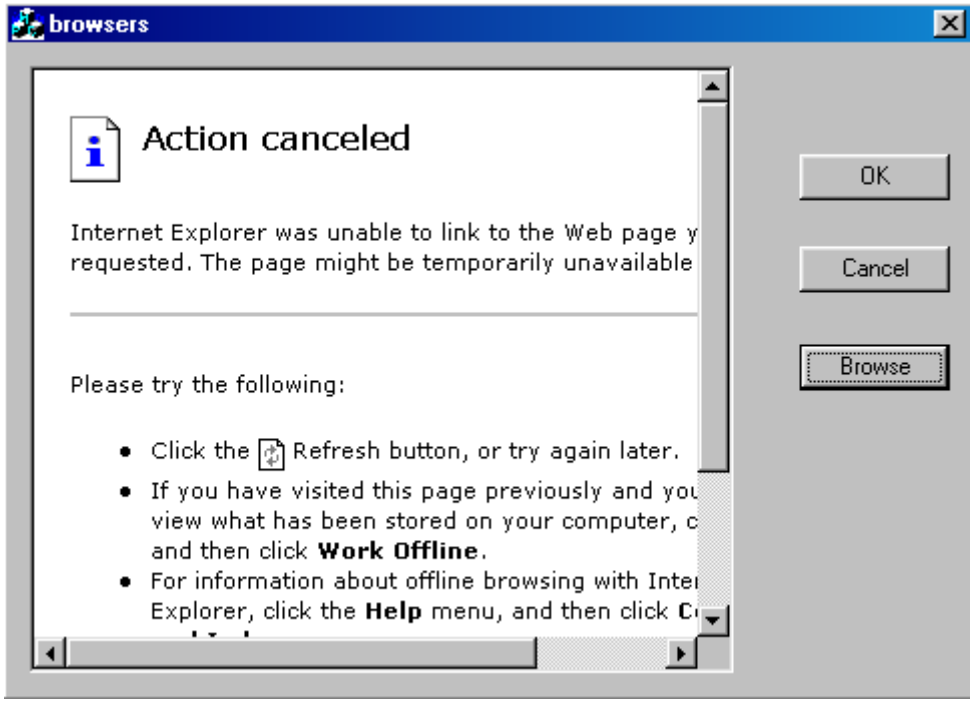
## *PROGRAM 14*

## *OBJECT OF THE PROGRAM: Creating a Web Browser*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `browser` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Dialog Based` in the AppWizard, click `Finish`.
6. To add a new Microsoft Web Browser control to our program.
   `Select Project` → `Add to Project` → `Components and Controls`.
   This opens `Visual C++ Components and Controls Gallery.`
7. Now double-click the entry marked `Registered ActiveX` Controls to open the list of `ActiveX` controls in your system.
8. Click the `Microsoft Web Browser` control, then click the `Insert` button.
9. `Visual C++` asks you what class you want for this new control; accept the default suggestion.
10. Open the main Dialog Window in the dialog editor. The browser control appears at bottom in the toolbox.
11. Drag a new control of that type to the dialog window under design, sizing it. This gives the browser control the `ID IDC_EXPLORER1`.
12. Using `ClassWizard,` connect a member variable to this control, naming it m_browser.
13. Add a button with caption Browse to the dialog window and connect an event handler method to that button, OnButton1().
14. When the user clicks the Browse button, we can navigate our Web Browser to www.microsoft.com.

```
void CBrowsersDlg::OnButton1()
{
     m_browser.Navigate(
     http://www.microsoft.com,0,0,0,0);

}
```

15. When you click browse button, the Web Browsers navigate to the Microsoft home page


browsers
Action canceled
Internet Explorer was unable to link to the Web page y
requested. The page might be temporarily unavailable
Please try the following:
Click the Refresh button, or try again later.
If you have visited this page previously and you
view what has been stored on your computer, c
and then click Work Offline.
For information about offline browsing with Inter
Explorer, click the Help menu, and then click C
OK
Cancel
Browse

## *PROGRAM 15*

## *OBJECT OF THE PROGRAM: Creating Internet Applications using HTTP*

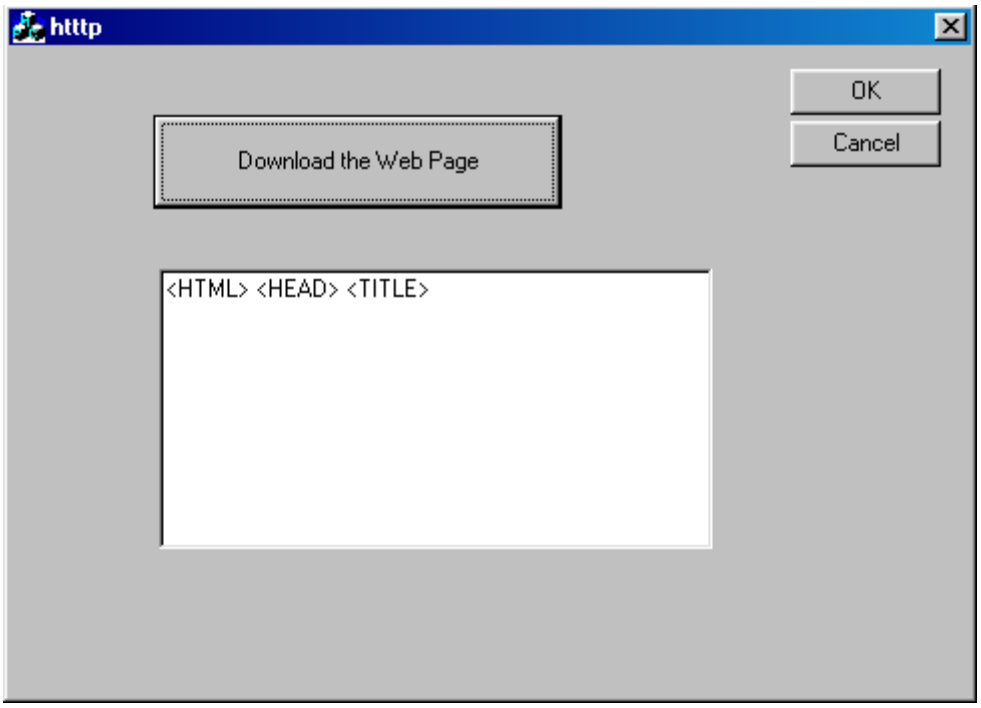
1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `http` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Dialog Based` in the AppWizard, click `Finish`.
6. Add a `text box` and a `button` with the `"Download the Web Page"` caption .
7. Connect an event handler, `OnButton1()`, to the button.
8. We'll create a new Internet Session. This Internet Session is actually an object of the `MFC CInternetSession` class, and this class is the basis of Visual C++ Internet support.
9. Also add the line `#include<afxinet.h>` to make sure we can use the Internet components.

```
// httpDlg.cpp : implementation file
.
.
#include "afxinet.h"
.
.
```

10. We've set aside a pointer, `pInternetSession`, for our new `Internet session` and we create the session now.
11. If computer is not connected to the Internet, the program will display a connect box and make the connection.If connection failed, we should terminate the program.
12. We'll use the `CInternetSession` class's `OpenURL()` method to open a web page for HTTP transfer. This returns a pointer to a file object of class `CStdioFile` and save that pointer as `pFile`.
13. Now we have a pointer to a file object representing the Web page we want to work with, and we can treat it just like a file.
14. If we want to download the file's first 1000 bytes, we do it by setting up a `buffer` for our data and using the `Read()` method.
15. To display the data received, we connect a member variable, `m_text`, to the `text box` in our program and places the downloaded text in that `text box`.
16. We close the file we've opened and `Internet Session`.

```
void CHtttpDlg::OnButton1()
{
     CInternetSession* pInternetSession;     //10

     pInternetSession=new CInternetSession(); //10
```

```
        if(!pInternetSession)                    //11
        {
             AfxMessageBox("Could not establish Internet
             Session",MB_OK);
             return;
        }

        CStdioFile* pFile=NULL;                  //12

        char* buffer;                            //14
        buffer=new char[1000];                   //14

        pFile=pInternetSession-
        >OpenURL(CString("http://www.microsoft.com"));
                                                 //12
        pFile->Read(buffer,1000);                //14

        m_text=CString(buffer,1000);             //15
        UpdateData(false);

        pFile->Close();                          //16
        pInternetSession->Close();               //16

}
```



## *PROGRAM 16*

## *OBJECT OF THE PROGRAM: Creating Internet Applications using FTP*

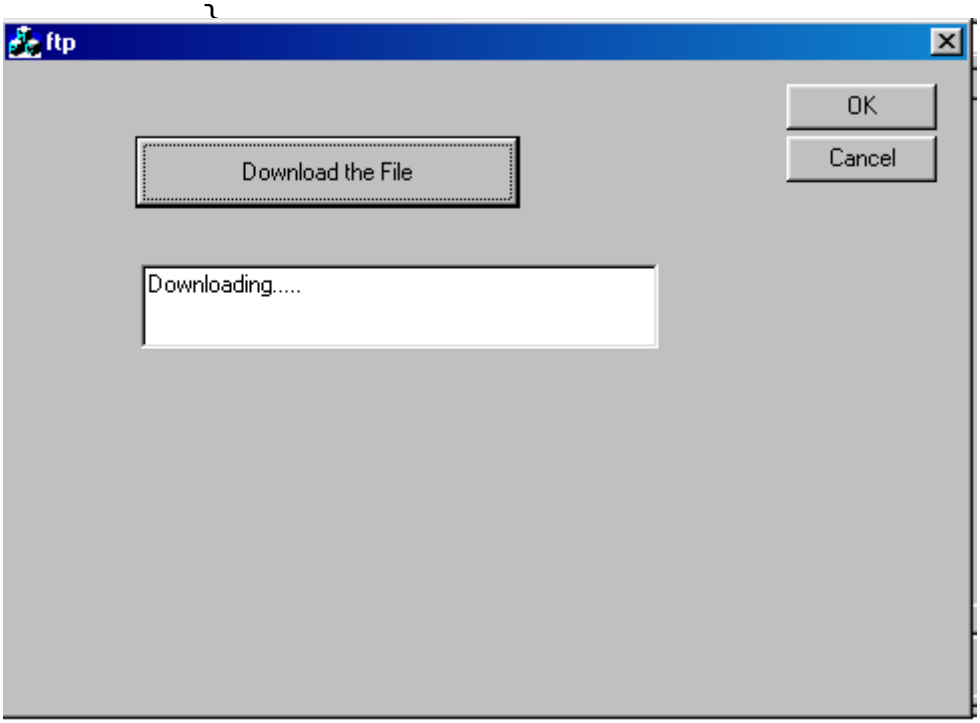
1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `ftp` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Dialog Based` in the AppWizard, click `Finish`.
6. Add a `text box` and a `button` with the `"Download the File"` caption .
7. Connect an event handler, `OnButton1()`, to the button and member variable, `m_text`, to the text box.
8. We'll create a new Internet Session. This Internet Session is actually an object of the `MFC CInternetSession` class, and this class is the basis of Visual C++ Internet support.
9. Also add the line `#include<afxinet.h>` to make sure we can use the Internet components.

```
// httpDlg.cpp : implementation file
.
.
#include "afxinet.h"
.
.
```

10. We've set aside a pointer, `pInternetSession`, for our new `Internet session` and we create the session now.
11. If computer is not connected to the Internet, the program will display a connect box and make the connection.If connection failed, we should terminate the program.
12. We create an object of class `pFtpConnection named pFTPConnection.` This class represents the `FTP` support in VC++.
13. To create this new object, we call the `InternetSession` class's `GetFtpConnection()` method to make an anonymous `FTP` connection to the `Microsoft site`, passing that method the name of the `FTP` site we want to connect to, `ftp.microsoft.com`.
14. If we were unsuccessful in connecting to the FTP site, we display an error message and finish up.
15. Otherwise, we indicate that we have started the downloading process by placing the message `"Downloading…"` in the `text box`.
16. We close the FTP connection and the Internet session at the end of OnButton1().

```
void CFtpDlg::OnButton1()
{
     CInternetSession* pInternetSession;     //10
     CFtpConnection* pFTPConnection;         //12
```

```
pInternetSession=new CInternetSession();  //11

if(!pInternetSession)                     //11
{
     AfxMessageBox("Could not establish Internet
     session",MB_OK);
     return;
}

pFTPConnection=pInternetSession->
GetFtpConnection(CString("ftp.microsoft.com"));
                                          //13

if(!pFTPConnection)                       //14
{
     AfxMessageBox("Could not establish FTP
     connection. ",MB_OK);
     return;
}

else                                      //15
{
     m_text="Downloading.....";
     UpdateData(false);
}

pFTPConnection->
GetFile(CString("disclaimer.txt"),CString("discl
aimer.txt"));

pFTPConnection->Close();                 //16
pInternetSession->Close();               //16

}
```



## *PROGRAM 17*

## *OBJECT OF THE PROGRAM: Creating an ActiveX Control*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC ActiveX ControlWizard` entry in the `New` dialog box.
3. Give the new program the name `boxer` in the Project name box.
4. Accept all the defaults by pressing the `Finish button.`
5. We have to divide the control into four rectangles, naming those rectangles box1 to box4 and declaring them in BoxerCtl.h:

```
class CBoxerCtrl : public COleControl
{
     .
     .
     .
     .
     // Implementation
protected:
     ~CBoxerCtrl();
     CRect box1;
     CRect box2;
     CRect box3;
     CRect box4;
     .
     .
     .
};
```

6. Now, we have four rectangles to divide the control up in OnDraw(), placing the four boxes at upper left, upper right, lower left, and lower right.
7. Then we draw the four rectangles.

```
void CBoxerCtrl::OnDraw(
CDC* pdc, const CRect& rcBounds, const CRect&
rcInvalid)
{
     pdc->FillRect(rcBounds,
     CBrush::FromHandle((HBRUSH)GetStockObject(WH
     ITE_BRUSH)));

     box1=CRect(rcBounds.left,rcBounds.top,
     rcBounds.right/2,rcBounds.bottom/2);  //6

     box2=CRect(rcBounds.left,rcBounds.bottom/2,r
     cBounds.right/2,rcBounds.bottom);   //6
```

```
            box3=CRect(rcBounds.right/2,rcBounds.top,
            rcBounds.right,rcBounds.bottom/2);   //6

            box4=CRect(rcBounds.right/2,rcBounds.bottom/
            2,rcBounds.right,rcBounds.bottom);  //6

            pdc->Rectangle(&box1);              //7
            pdc->Rectangle(&box2);              //7
            pdc->Rectangle(&box3);              //7
            pdc->Rectangle(&box4);              //7
      }
```

8. To handle mouse click we add an eventhandler `LButtonDown()` using ClassWizard.
9. We can record which of our flag rectangles the user clicked, and which rectangle to fill with color- by setting up four new Boolean flags, fill1 to fill4, in the `CBoxerCtl` header.

```
   class CBoxerCtrl : public COleControl
      {
            .
            .
            // Implementation
      protected:
            ~CBoxerCtrl();
            CRect box1;
            CRect box2;
            CRect box3;
            CRect box4;

            boolean fill1;
            boolean fill2;
            boolean fill3;
            boolean fill4;
            .
            .
      };
```

10. Set those flags to false in the control's constructor :

```
      CBoxerCtrl::CBoxerCtrl()
      {
            InitializeIIDs(&IID_DBoxer,&IID_DBoxerEvents);
            fill1=fill2=fill3=fill4=false;
      }
```

11. In `OnLButtonDown()`, we can set the boolean fill flags using the handy `CRect` method `PtInRect()`, which returns true if the point you pass to it is in a certain rectangle, such as our box1 to box4 objects:

```
      void CBoxerCtrl::OnLButtonDown(UINT nFlags,
      CPoint point)
```

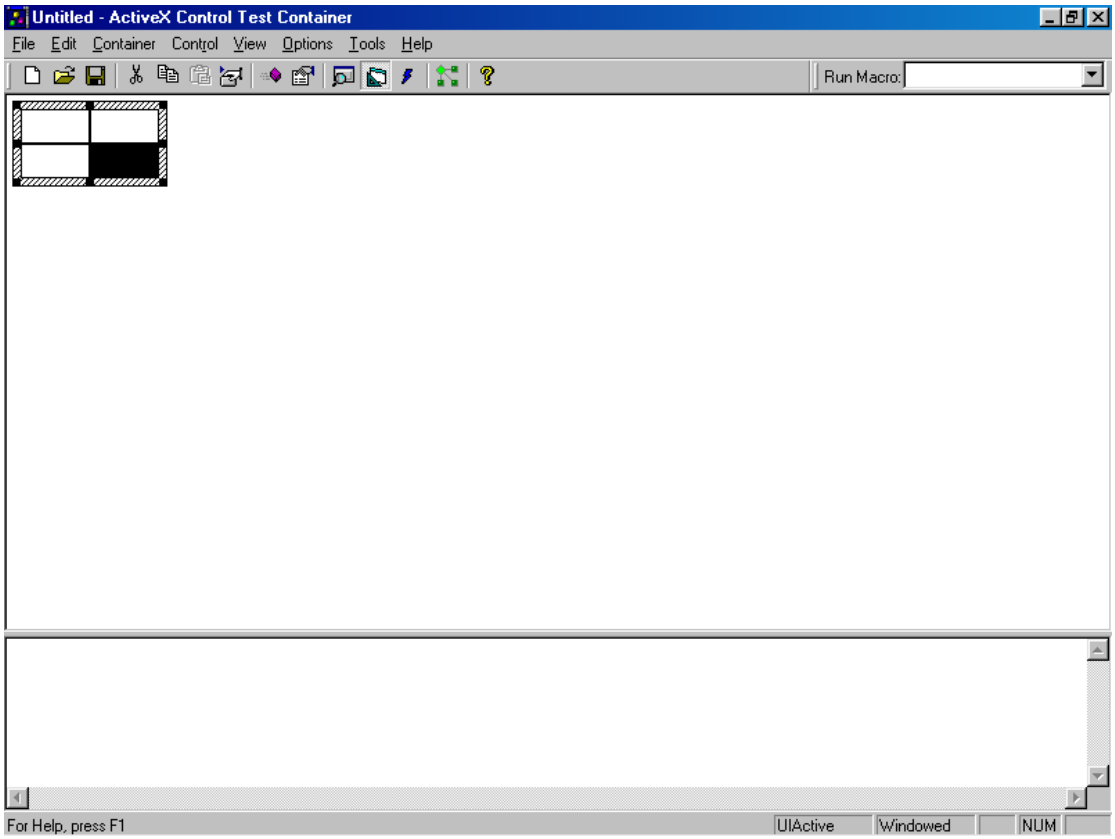
```
{
     fill1=box1.PtInRect(point);
     fill2=box2.PtInRect(point);
     fill3=box3.PtInRect(point);
     fill4=box4.PtInRect(point);
     Invalidate();

     COleControl::OnLButtonDown(nFlags, point);
}
```

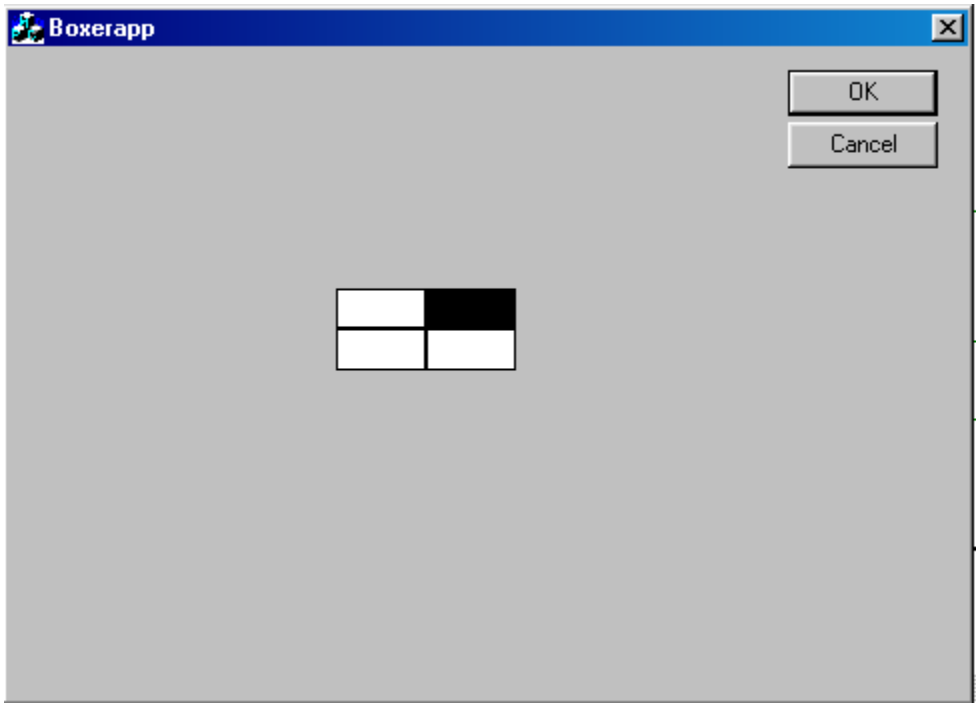
12. `Invalidate()` calls the OnDraw(), there we check the four Boolean fill flags and fill the corresponding rectangle using the `CDC` method `FillSolidRect()`:

```
void CBoxerCtrl::OnDraw(
CDC* pdc, const CRect& rcBounds, const
CRect& rcInvalid)
{
     pdc->FillRect(rcBounds,
     CBrush::FromHandle((HBRUSH)GetStockObject(WH
     ITE_BRUSH)));


     box1=CRect(rcBounds.left,rcBounds.top,
     rcBounds.right/2,rcBounds.bottom/2);

     box2=CRect(rcBounds.left,rcBounds.bottom/2,r
     cBounds.right/2,rcBounds.bottom);

     box3=CRect(rcBounds.right/2,rcBounds.top,
     rcBounds.right,rcBounds.bottom/2);

     box4=CRect(rcBounds.right/2,rcBounds.bottom/
     2,rcBounds.right,rcBounds.bottom);

     pdc->Rectangle(&box1);
     pdc->Rectangle(&box2);
     pdc->Rectangle(&box3);
     pdc->Rectangle(&box4);


     if(fill1) pdc->
     FillSolidRect(&box1,RGB(0,0,0));
     if(fill2) pdc->
     FillSolidRect(&box2,RGB(0,0,0));
     if(fill3) pdc->
     FillSolidRect(&box3,RGB(0,0,0));
     if(fill4) pdc->
     FillSolidRect(&box4,RGB(0,0,0));
```

```
}
```

13. To test "boxer", start Build→Build Boxer .ocx to build the boxer.ocx and registered that control with Windows.
14. Select the ActiveX Control Test Container item in the tools menu, which brings up the very useful test container tool.
15. Select Insert New Control item in the test container's Edit menu and Double click the Boxer control in the Insert Control box that appears.This inserts our ActiveX control in the test container.
16. Click one of the boxes in the control,shading it. Clicking another rectangle shades that rectangle instead.
17. To embed the ActiveX in a program.Create a new dialog based program now named Boxerapp. To insert a control of Boxer type in this program : select Project→Add to Project→Components and Controls, opening the Components and Controls Gallery.
18. Double click on the Registered ActiveX Controls. Select the Boxer Control entry and click insert button.This inserts the Boxer control into the dialog's editor's toolbox, where it has an icon OCX.




Boxerapp
OK
Cancel

## *PROGRAM 18*

## *OBJECT OF THE PROGRAM: Creating a Dialog based Application*

1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `dialogbased` in the Project name box.
4. Click `OK` to starts the Visual C++ AppWizard.
5. Click the option marked `Dialog Based` in the AppWizard, click `Finish`.
6. Here, the AppWizard indicates four classes will be created : CdialogbasedApp, CMainFrame, CdialogbasedDoc, CdialogbasedView .
7. Click the Resources tab, open the Dialog folder, and click the entry for our program main window, IDD_BUTTONS_DIALOG. This opens the dialog editor.

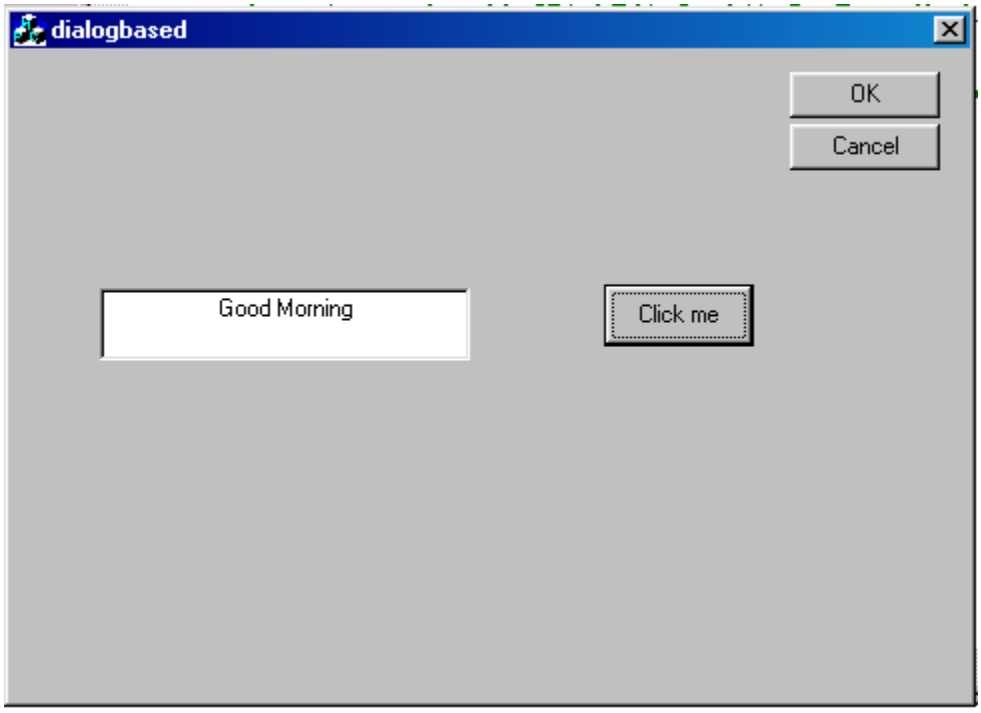
8 Add a new button with the caption "Click me" to our code.

9. Open Class Wizard to connect Click me button to our code.
10. Use Class Wizard to connect the button to an event handler, OnButton1().
11. Double-click IDC_BUTTON1 in the Object IDs box, and double click BN_CLICKED in the Message Box. This creates OnButton1() :

```
void  Cdialog::OnButton1()
{

}
```

12. Add a member variable, m_edit to EDIT box i.e IDC_EDIT1.
13. Add following code to OnButton1():

```
void  Cdialog::OnButton1()
{
      m_edit.SetWindowText(CString("Good Morning "));
}
```


dialogbased
OK
Cancel
Good Morning
Click me

## *PROGRAM 19*

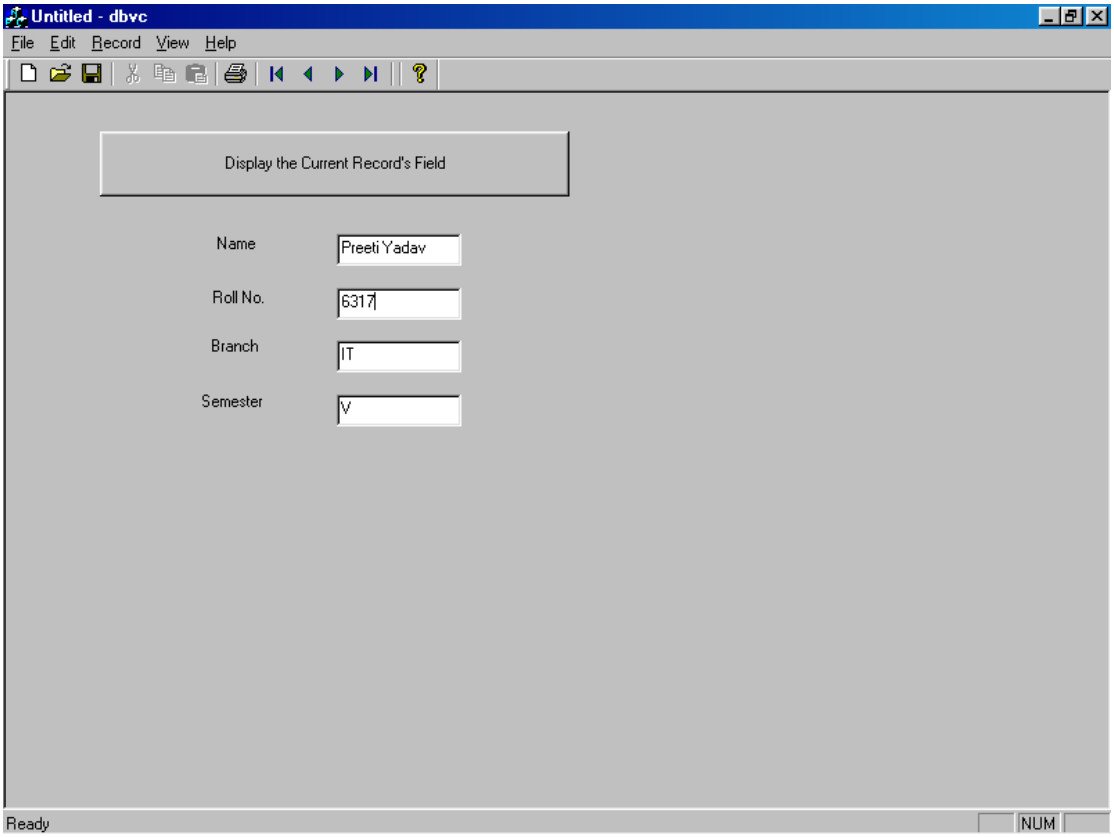
## *OBJECT OF THE PROGRAM: Database Connectivity in VC++*

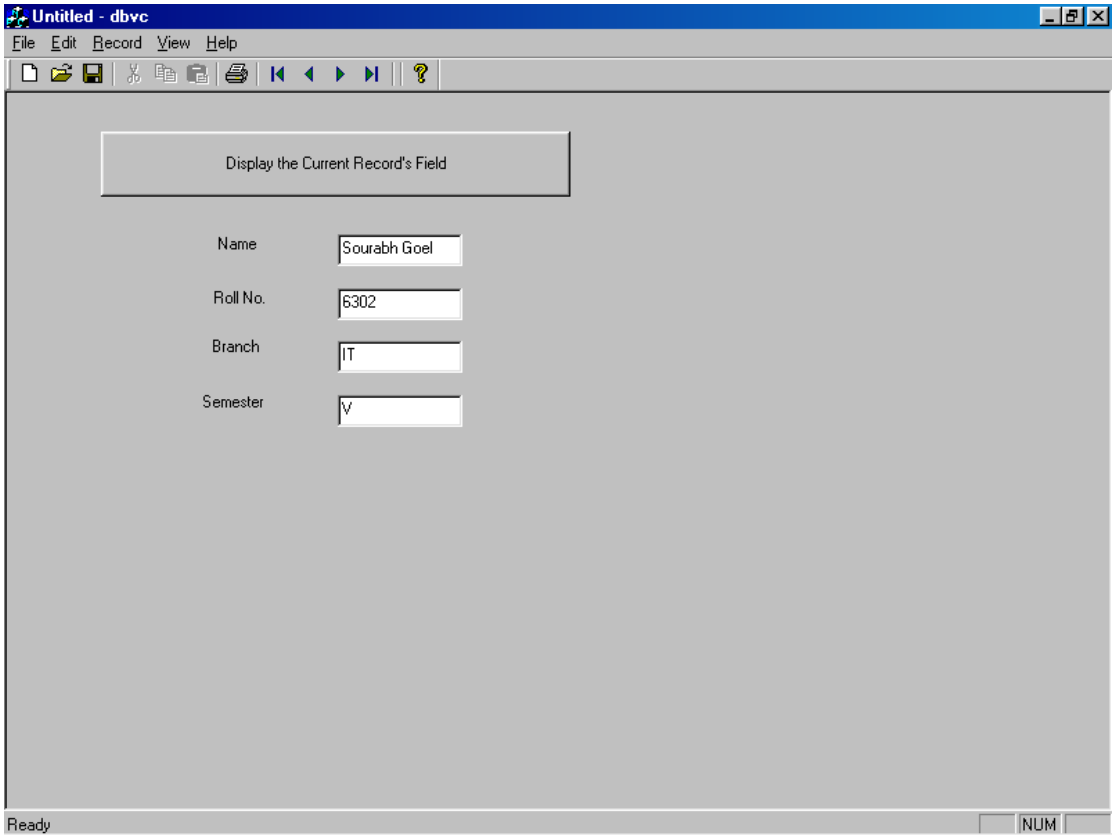
1. Open Visual C++ and click the `New` item in the File menu, opening the New dialog box.
2. Now select the `MFC AppWizard(exe)` entry in the `New` dialog box.
3. Give the new program the name `dbvc` in the Project name box.
4. Click Next button, There is a Question " What Database support would you like to include?", click the radio button labeled Database View with File Support.
5. Next, click the button labeled Data Source to open the Database Options box.
6. We select the DAO button, and specify the dbs.mdb filesas our Data source.
7. When the Select Database Tables appears, click the Student table and the OK button.

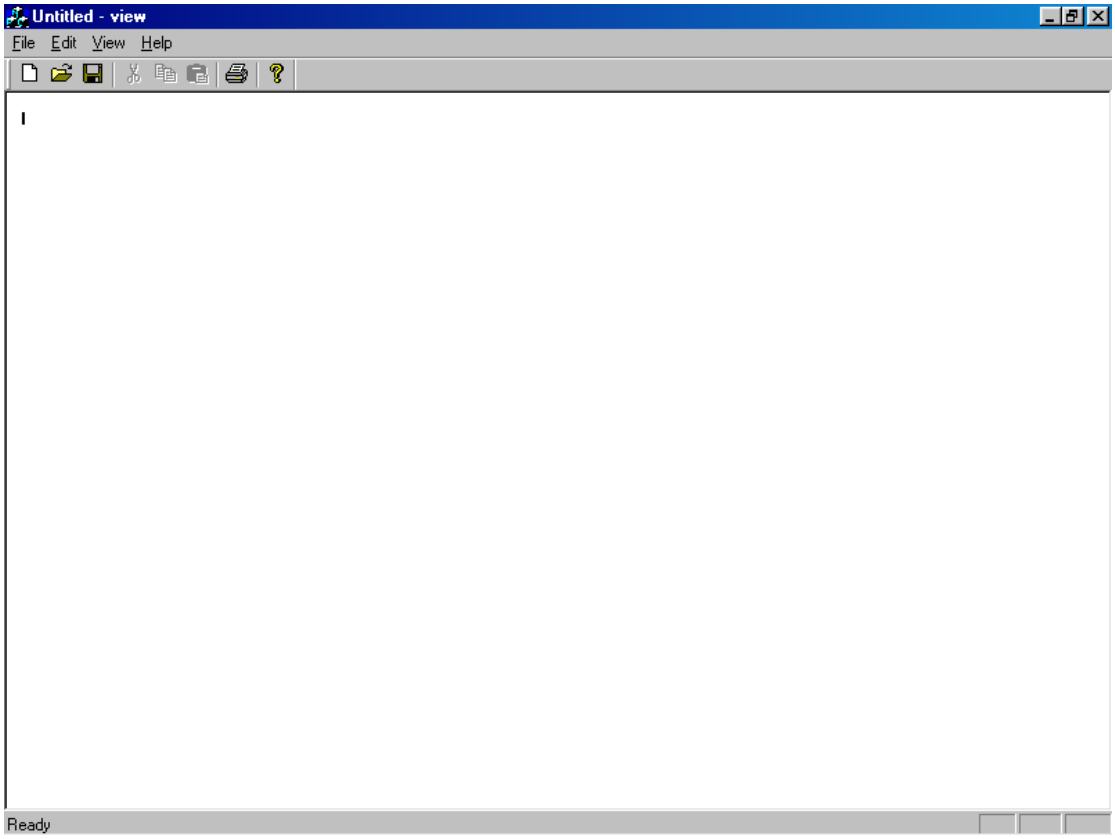
Note :

8. Now we click the OK button and finish button in App Wizard, letting AppWizard create the New Program.
9. Open the main view, IDD_DBVC_FORM, in the dialog editor, and add the controls : two text boxes & a button with caption " Display the current record's fields ".
10. Using Class Wizard, we connect an event handler, OnButton1() to the button & four member variable m_text1, m_text2, m_text3, m_text4 to four text boxes & four static box to four text boxes.
11. Add codes to OnButton1():

```
void CDbvcView::OnButton1()
{
     m_text1=m_pSet->m_Name;
     UpdateData(false);

     m_text2=m_pSet->m_Roll_No;
     UpdateData(false);

     m_text3=m_pSet->m_Branch;
     UpdateData(false);

     m_text4=m_pSet->m_Semester;
     UpdateData(false);

}
```


Untitled - dbvc
File Edit Record View Help
Display the Current Record's Field
Name
Preeti Yadav
Roll No.
6317
Branch
IT
Semester
V
Ready
NUM


Untitled - dbvc
File Edit Record View Help
Display the Current Record's Field
Name
Sourabh Goel
Roll No.
6302
Branch
IT
Semester
V
Ready
NUM


Untitled - view
File Edit View Help
Ready

## *PROGRAM 19*

## *OBJECT OF THE PROGRAM: Keyboard Handling*

**1**. Open Visual C++ and click the New item in the File menu, opening the New dialog box.
2. Now select the MFC AppWizard(exe) entry in the New dialog box.
3. Give the new program the name Keystrokes in the Project name box.
4. Click OK to starts the Visual C++ AppWizard.
5. Click the option marked Single Document in the AppWizard, click Finish.
6. Here, the AppWizard indicates four classes will be created : CKeystrokesApp, CMainFrame, CKeystrokesDoc, CKeystrokesView .
7. Declare the StringData variable on the document's header file, KeystrokesDoc.h, in the protected part.

```
class CKeystrokesDoc : public CDocument
{
protected: // create from serialization only
                CKeystrokesDoc();
                DECLARE_DYNCREATE(CKeystrokesDoc)
                CString StringData;
                        .
                        .
};
```

8. Initialize that string to an empty string -" "- in the document's constructor, which we find in KeystrokesDoc.cpp.

```
CKeystrokesDoc::CKeystrokesDoc()
{
                StringData=" ";
                // TODO: add one-time construction code here
}
```

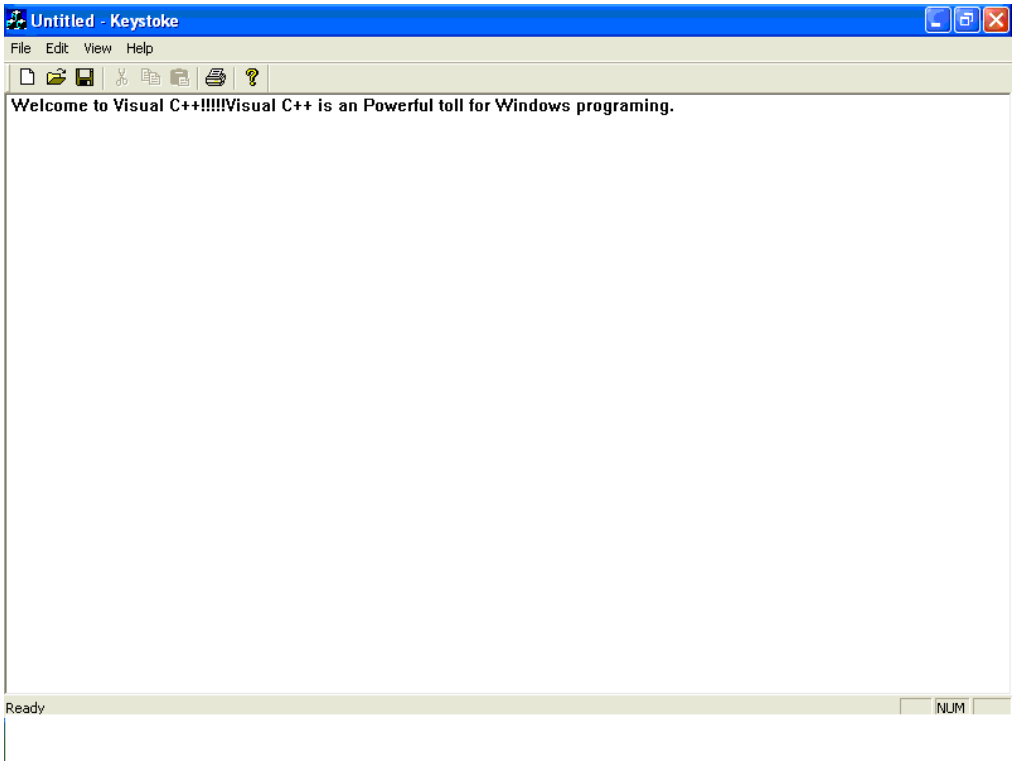
9. Add a new event handler –OnChar( )- to our view class which is called every time the user types the character.
10. The character the user typed is now in the nChar parameter, which we store in the our data string object, StringData. The object in our document, so we need a pointer(pDoc) to our document object.
11. Add the character nChar to the string StringData.

```
void CKeystrokesView::OnChar(UINT nChar, UINT nRepCnt,
        UINT nFlags)
{
// TODO: Add your message handler code here and/or call default
                CKeystrokesDoc* pDoc=GetDocument();
                ASSERT_VALID(pDoc);
```

```
                pDoc->StringData+=nChar;
                Invalidate();

                CView::OnChar(nChar, nRepCnt, nFlags);
        }
```

12. We'll handle the display of our data in the view's OnDraw( ). We need to draw the text string, which we do with TextOut( ).

```
        void CKeystrokesView::OnDraw(CDC* pDC)
        {
                CKeystrokesDoc* pDoc = GetDocument();
                ASSERT_VALID(pDoc);

                pDC->TextOut(0,0,pDoc->StringData);
                // TODO: add draw code for native data here
        }
```



## ***PROGRAM 20***

## ***OBJECT OF THE PROGRAM: Adding Carets***

**1**. Open Visual C++ and click the New item in the File menu, opening the New dialog box.
2. Now select the MFC AppWizard(exe) entry in the New dialog box.
3. Give the new program the name Carets in the Project name box.
4. Click OK to starts the Visual C++ AppWizard.
5. Click the option marked Single Document in the AppWizard, click Finish.
6. Here, the AppWizard indicates four classes will be created : CCaretsApp, CMainFrame, CCaretsDoc, CCaretsView .
7. Declare the StringData variable on the document's header file, CaretsDoc.h, in the protected part.

```
class CCaretsDoc : public CDocument
{
protected: // create from serialization only
                CCaretsDoc();
                DECLARE_DYNCREATE(CCaretsDoc)
                CString StringData;
                        .
                        .
};
```

8. Initialize that string to an empty string -" "- in the document's constructor, which we find in CaretsDoc.cpp.

```
CCaretsDoc::CCaretsDoc()
{
                StringData=" ";
                // TODO: add one-time construction code here
}
```

9. Add a new event handler –OnChar( )- to our view class which is called every time the user types the character.
10. The character the user typed is now in the nChar parameter, which we store in the our data string object, StringData. The object in our document, so we need a pointer(pDoc) to our document object.
11. Add the character nChar to the string StringData.

```
void CCaretsView::OnChar(UINT nChar, UINT nRepCnt,
        UINT nFlags)
{
// TODO: Add your message handler code here and/or call default
                CCaretsDoc* pDoc=GetDocument();
                ASSERT_VALID(pDoc);
                pDoc->StringData+=nChar;
                Invalidate();
```

```
        CView::OnChar(nChar, nRepCnt, nFlags);
}
```

12. Set a boolean variable named CaretCreated in view object to keep track that caret has been created or not & CaretPosition & x,y.

```
class CCaretsView : public CView
{
protected: // create from serialization only
        CCaretsView();
        DECLARE_DYNCREATE(CCaretsView)
        CPoint CaretPosition;
        int x,y;
        boolean CaretCreated;
// Attributes
        .
        .
};
```

13. Set CaretCreated to false in the View's Constructor.

```
CCaretsView::CCaretsView()
{
CaretCreated=false;
// TODO: add construction code here
}
```

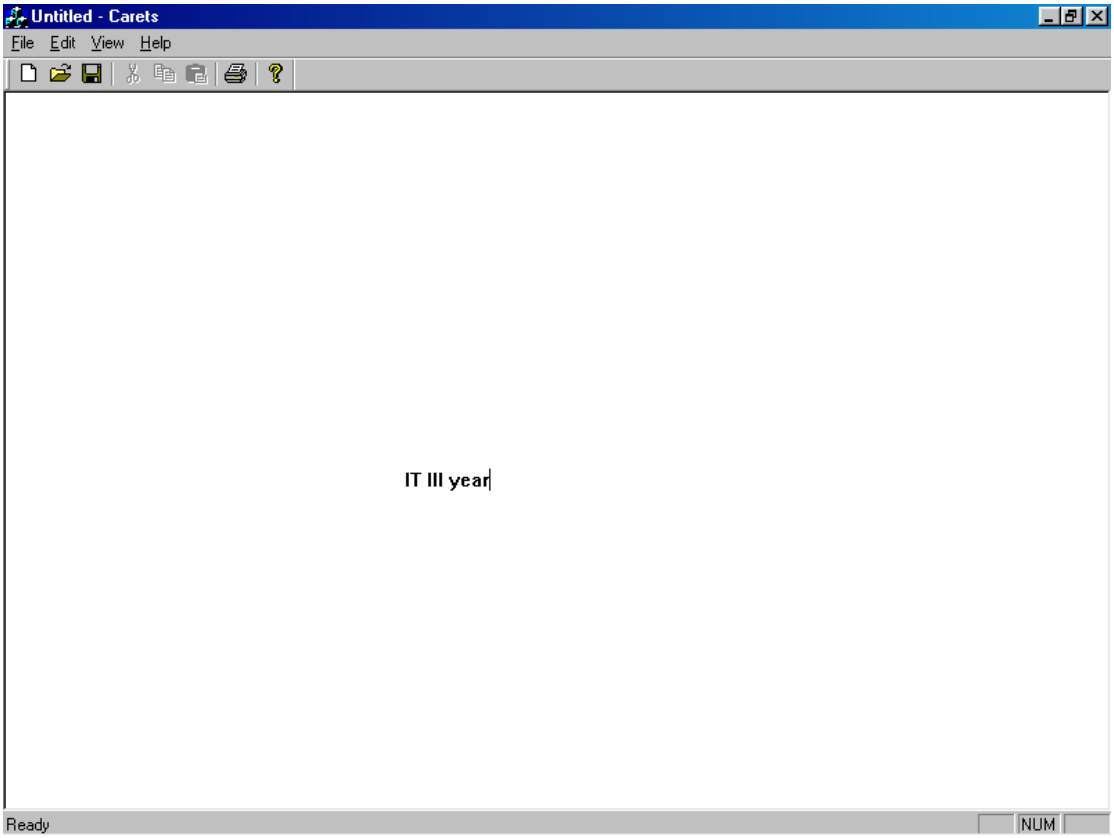
14. We're ready to create our new caret. We'll make the caret the same height and width as our text.We call CreateSolidCaret() to actually create the caret.
15. We'll store caret's position in a new CPoint object named CaretPosition. The CPoint class has two data members x & y which holds the position of Caret.
16. Initially, set caret's position to (0,0) in OnDraw().
17. We set the caret's position with SetCaretPos(), show caret on the screen with ShowCaret(), and set the CaretCreated Boolean flag to true.
18. In next step the caret is to move as the user types text.
19. Place the caret at the end of displayed text string.
20. To display the caret at the end of the text string, we first hide it using HideCaret().
21. Set the CaretPosition & Show the Caret.

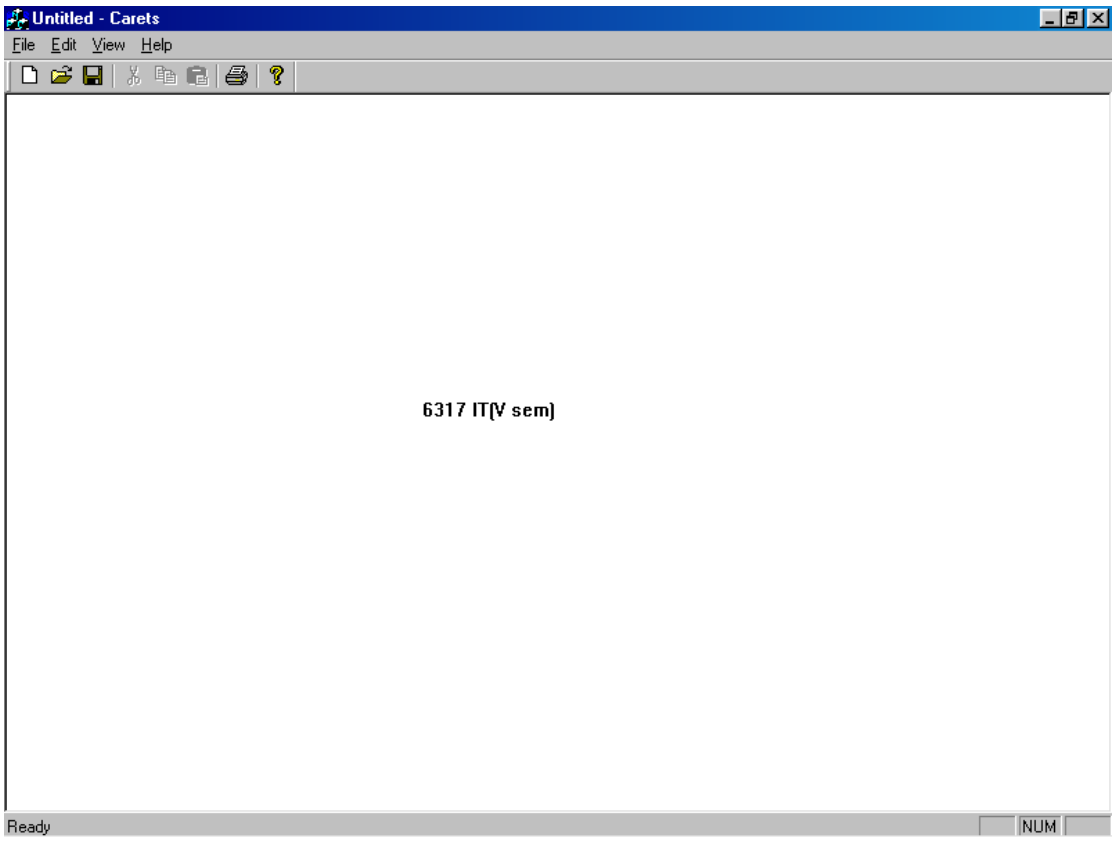
```
void CCaretsView::OnDraw(CDC* pDC)
{
        CCaretsDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if(!CaretCreated)
        {
                TEXTMETRIC textmetric;
                pDC->GetTextMetrics(&textmetric);
```

```
                    CreateSolidCaret(textmetric.tmAveCharWidth/8,textmetric
                    .tmHeight);//(14)
                    CaretPosition.x=CaretPosition.y=0;
                            //(16)
                    SetCaretPos(CaretPosition);//(17)
                    ShowCaret();                    //(17)
                    CaretCreated=true;          //(17)
                }
                pDC->TextOut(x,y,pDoc->StringData);
                                                //(18)
                CSize size=pDC->GetTextExtent(pDoc
                ->StringData);          //(19)
                HideCaret();            //(20)
                CaretPosition.x=x+size.cx;
                CaretPosition.y=y;
                SetCaretPos(CaretPosition);   //(21)
                ShowCaret();                            //(21)

                // TODO: add draw code for native data here
        }
```

22. To handle left mouse button down we select LButtonDown, point parameter , an Object of the CPoint class, holds mouse's present location.
23. Store the variables in x & y.

```
void CCaretsView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
            x=point.x;
            y=point.y;

            CCaretsDoc* pDoc=GetDocument();
            ASSERT_VALID(pDoc);
            pDoc->StringData.Empty();
            Invalidate();
            CView::OnLButtonDown(nFlags, point);
    }
```


Untitled - Carets
File Edit View Help
IT III year
Ready
NUM


Untitled - Carets
File Edit View Help
6317 IT(V sem)
Ready
NUM

# REFERENCES


- Microsoft Visual C++ by Steven Holzner
- Visual C++ Programming ,2nd edition by Steven Holzner
- Visual Basic Programming by Steven Holzner
- MSDN Help
- The Complete Reference  by Sahoo
- Visual C++:From the ground up By Muller

# NEW IDEAS BESIDES UNIVERSITY SYLLABUS

Apart from the University syllabus these were the list of programs that were performed by the students to upgrade their skills in VC++ and VB.

1. Create an SDI application in VC++ using which the user can draw at most 20 rectangles in the client area. All the rectangles that are drawn should remain visible on the screen even if the window is refreshed. Rectangle should be drawn on the second click of the left mouse button out of the two consecutive clicks. If the user tries to draw more than 20 rectangles, a message should get displayed in the client area that "No more rectangles can be drawn".

2. Write a program in VB to create a notepad.

3. Write a program in VC++ to implement a simple calculator.

4. Create an SDI application in VC++ that adds a popup menu to your application which uses File drop down menu attached with the menu bar as the pop-up menu. The pop-up menu should be displayed on the right click of the mouse.

5. Write a program in VC++ to create a static link library and a dynamic link library.

6. Write a program in VC++ to create a static link library and a dynamic link library.

7. Create a simple database in MS Access Database and a simple database application in VC++ that shows database connectivity through ADO model.

8. Make an Active X control of your own using VB.

10. With the help of VB, create an object of excel application and implement any action on it.

# FAQs

Q1. What do you mean by MFC.What is their relationship with APIs?

Q2. What are the four classes of MFC?Explain their functionality.

Q3. Explain document/view architecture.

Q4. What is message passing?

Q5. What do you mean by event oriented programming?

Q6. What is the function of GetDocument()?

Q7. Explain the function of WM_CHAR message.

Q7. What is the function of ASSERT_VALID()?

Q8. Where do you define the co-ordinates of a mouse?

Q9. What is the function of OnChar()?

Q10. How will you add a shortcut key to a menu item?

Q11. How will you add a toolbar to a menu item?

Q12.How will you add a accelerator key to a menu item?

13. How will you create a dialog box?

Q14. What is the base class of a dialog box?

Q15. How will you create a radio button?

Q16. How will you create a check box?

Q17. How will you create a list box?

Q18. How will you create a combo box?

Q19. What do you mean by Serialization?

Q20. How will you serialize an object?

Q21. How will you serialize a class?

Q22. What is MDI?

Q23. What is SDI?

Q24. What do you mean by WM_LBUTTONDOWN?

Q25. What is an ActiveX control?

Q26. How will you create an Active X control?

Q27. How will you create a web browser?

Q28. How will you create an internet application using HTTP?

Q29. How will you create an internet application using FTP?

Q30. How will you connect a database to your application in VC++?

Q31. How will you connect a database to your application in VB?

Q32. What do you mean by OLE?

Q33. What are Sliders?

Q34. What is the function of DoModal()?

Q35. What is File Handling?

Q36. What are the five modes of handling files?

Q37. How will you debug a program?

Q38. Create a simple application in VB.

Q39. What is the difference between MFC and API?

Q40. What are the two varieties of API?

Q41. What is the Function Of ASSERT_VALID?

Q42. What are the four classes of VC++?

Q43. With which extension you save your document
in case of serialization?

Q44. In which method you will display your document?

Q45. In serialization,what is the function of ‘ar’and it is the object of which class?