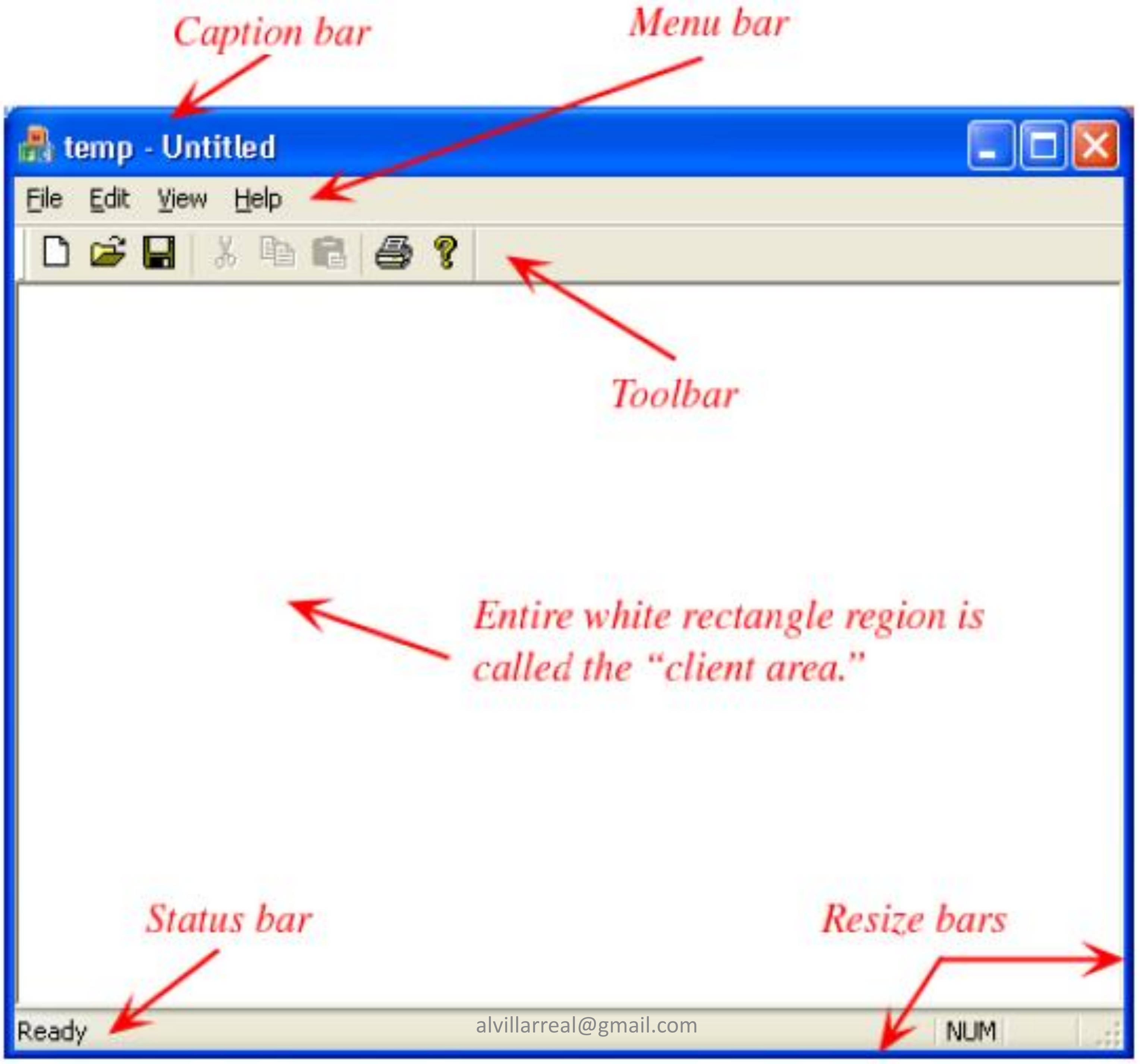
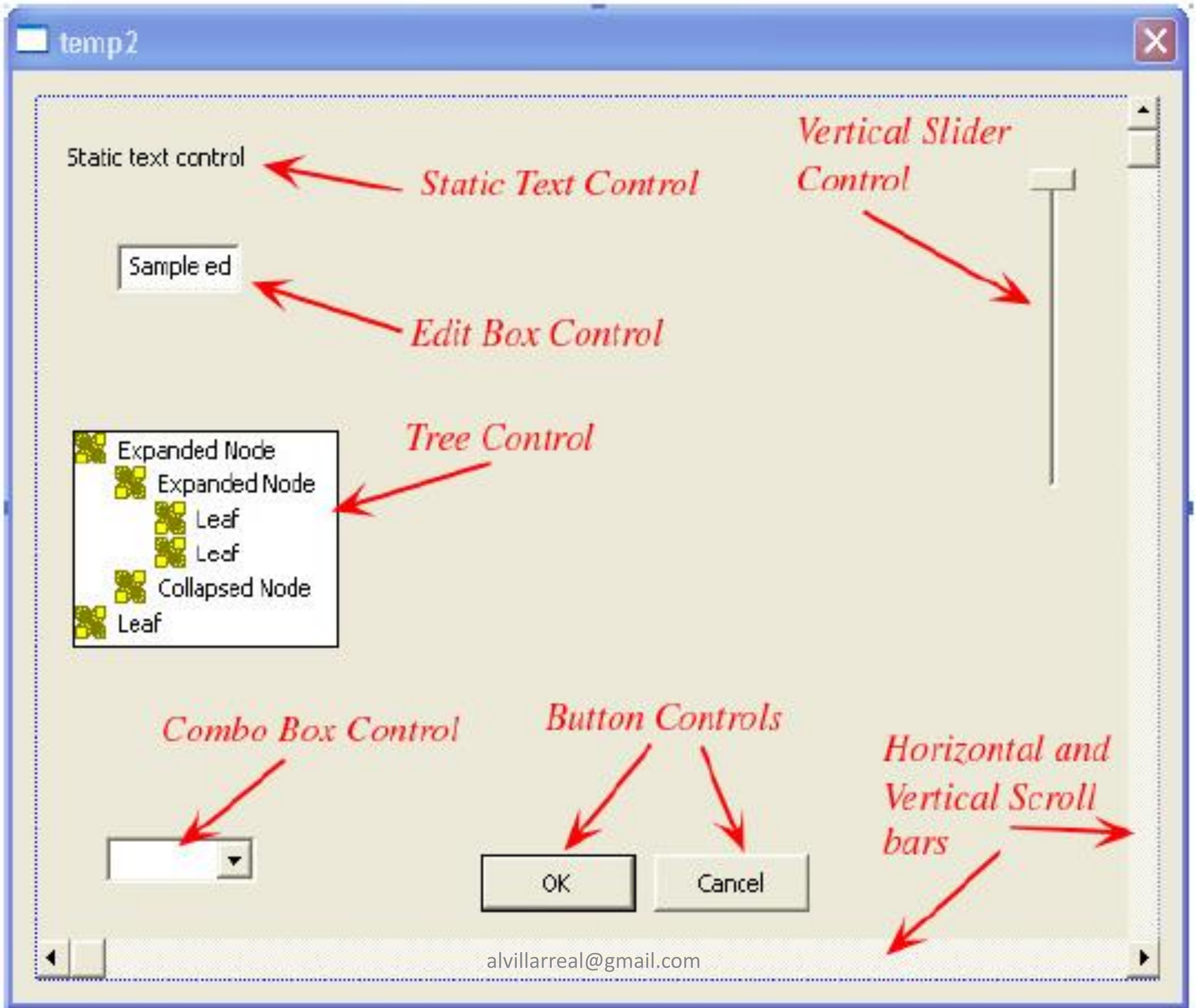
Win32 API (Application Programming Interface).

Programación 2

MGTI Alejandro Villarreal Mojica


Caption bar
Menu bar
temp - Untitled
File
Edit
View
Help
Toolbar
Entire white rectangle region is called the "client area."
Status bar
Resize bars
Ready
NUM


temp2
Static text control
Static Text Control
Sample ed
Edit Box Control
Vertical Slider Control
Tree Control
Expanded Node
Expanded Node
Leaf
Leaf
Collapsed Node
Leaf
Combo Box Control
Button Controls
OK
Cancel
Horizontal and Vertical Scroll bars

GUI

- The primary characteristic of a Win32 application is the **GUI** (**Graphical User Interface**).
- GUI refers to the graphics with which the user interacts—the menus, buttons, scroll bars, etc. Instead of interacting with a console, users now interact with a GUI.

Your First Windows Program.

```
#include <Windows.h>

int WINAPI WinMain(
             HINSTANCE hInst,
             HINSTANCE hPrev,
             LPSTR cmdLine,
             int showCmd)
{
char titulo[] = "Aviso importante";
int opc = 0;
opc = MessageBox(0, "mensaje", titulo, MB_ICONINFORMATION +
                                        MB_OKCANCEL);
if (opc == IDOK)
{
   MessageBox(0, "LE DIO OK", titulo,
   MB_ICONQUESTION|MB_YESNOCANCEL);
}
```

Parameters

The first parameter, hInstance of type HINSTANCE, is essentially a value that identifies the application for Windows—an application ID, if you will, that Windows OS passes into your application when it begins. It is necessary to ID various applications because Windows can be running several different applications at once. Note that the Win32 API defines the type HINSTANCE.

Parameters

The <u>second parameter</u> hPrevInstance is no longer used in 32-bit Windows programming—it is a legacy of 16-bit Windows.

The <u>third parameter</u> cmdLine is a string of command line arguments. (PSTR is essentially a typedef for a char*). Command line arguments are string arguments a user can pass into an application before it starts. Command line arguments typically give the application special instructions on how it should execute.

Parameters

Finally, the fourth parameter, showCmd, is an integer that specifies how the main application window should be initially shown.

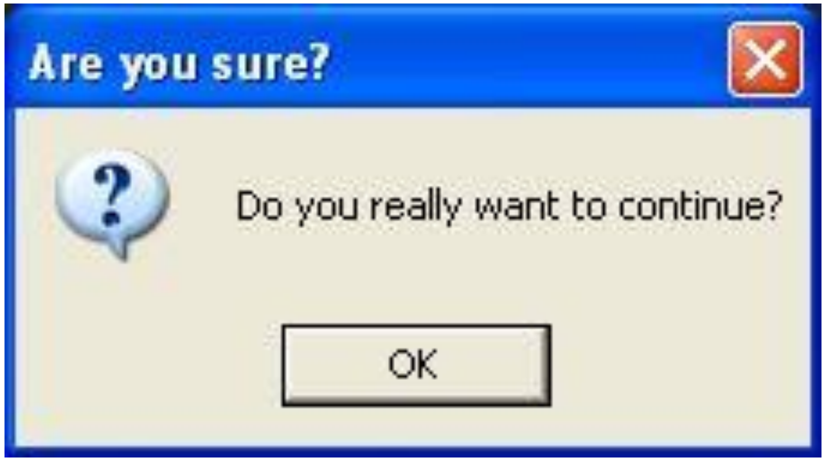
For example, should it be maximized, minimized or normal? The Windows OS makes the decision based on a variety of factors.

For instance, if you try to launch an application while the system is busy, Windows will pass in a value for showCmd indicating that it, perhaps, be 99 minimized. In summary, the parameters to WinMain are parameters the OS passes into the application when the application starts.

MessageBox function.

`Creating a Windows message box is achieved by using the MessageBox() function.`

MessageBox function.



Are you sure?

Do you really want to continue?

To replicate the message box seen above we would type the following;

```
MessageBox(NULL,"Do you really want to continue?","Are you sure?",MB_ICONQUESTION);
```

MessageBox function.

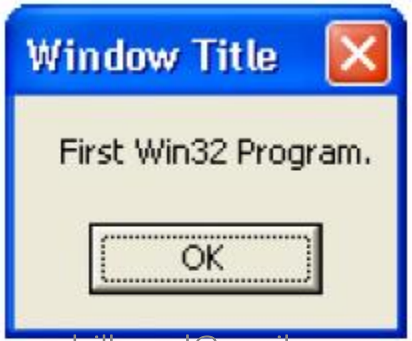
The message box function is defined as follows;

```
int MessageBox(
   HWND hWnd,                // handle to owner window
   LPCTSTR lpText,           // text in message box
   LPCTSTR lpCaption,        // message box title
   UINT uType                // message box style
);
```

message box style

The fourth parameter (uType) is an unsigned integer value that denotes a style flag. Here is an abridged list of possible style flags (these are predefined values of type unsigned int):

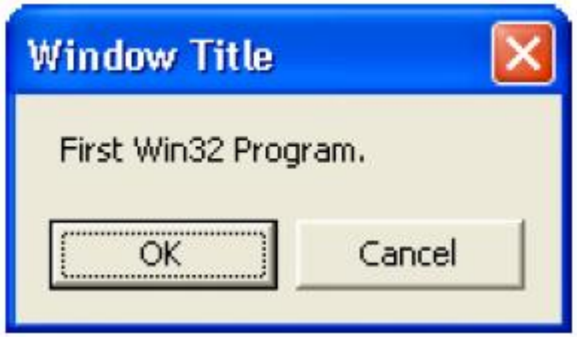
MB_OK: Instructs the message box to display an OK button.



First Win32 Program.

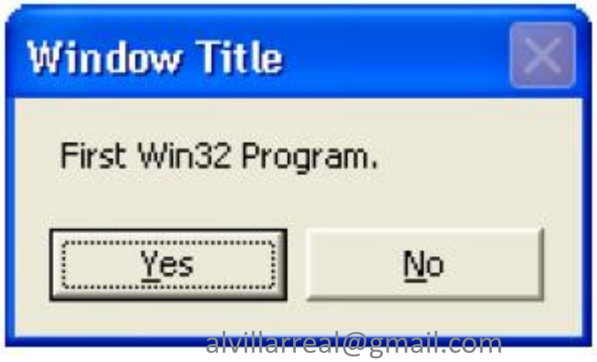
message box style

MB_OKCANCEL: Instructs the message box to display an OK and CANCEL button.



First Win32 Program.

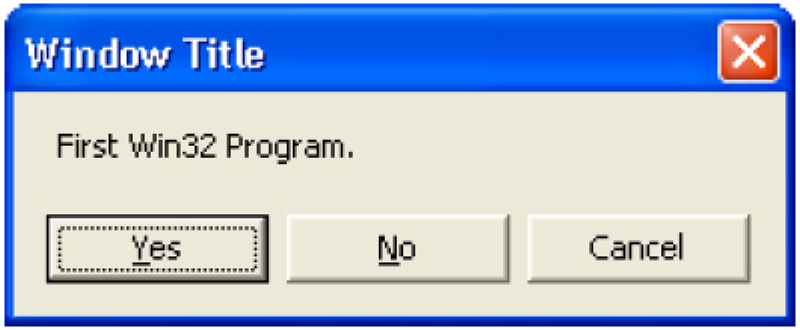
MB_YESNO: Instructs the message box to display a YES and NO button.



First Win32 Program.

message box style

MB_YESNOCANCEL: Instructs the message box to display a YES, NO, and CANCEL button.



First Win32 Program.

message box style

- Finally, the message box’s return value depends on which button the user pressed; here is an abridged list of return values (see the Win32 documentation for more details):
  - IDOK: The user pressed the OK button.
  - IDCANCEL: The user pressed the CANCEL button.
  - IDYES: The user pressed the YES button.
  - IDNO: The user pressed the NO button.
- You can test which value was returned using an “if” statement and thus determine which button the user selected and then take appropriate action.

message box style

- We could have used any of the following types:

- MB_ICONQUESTION
  MB_ICONWARNING
  MB_ICONINFORMATION
  MB_ICONERROR

message box style

The other main elements of interest are the types input buttons available to the user (ie, Yes, No, Cancel, etc..)

MB_ABORTRETRYIGNORE	Abort, Retry, and Ignore
MB_CANCELTRYCONTINUE	Cancel, Try Again, and Continue
MB_HELP	Help
MB_OK	OK
MB_OKCANCEL	OK and Cancel
MB_RETRYCANCEL	Retry and Cancel
MB_YESNO	Yes and No
MB_YESNOCANCEL	Yes, No, and Cancel

message box style

The message box function will return the value of the button that was pressed. The return values are actually integers, but it is best to use the 'defines' as listed below for readability and fault finding.

IDABORT	Abort button was pressed
IDCANCEL	Cancel button was pressed
IDCONTINUE	Continue button was pressed
IDIGNORE	Ignore button was pressed
IDNO	No button was pressed
IDOK	OK button was pressed
IDRETRY	Retry button was pressed
IDTRYAGAIN	Try Again button was pressed
IDYES	Yes button was pressed

```
#include <windows.h>
INT WINAPI wWinMain(HINSTANCE hInst,
                    HINSTANCE hPrevInst,
                    LPWSTR lpCmdLine,
                    INT nShowCmd)
{
int nResult = MessageBox(NULL,
                "An example of Cancel,Retry,Continue",
                "Hello Message Box!",
                MB_ICONERROR | MB_ABORTRETRYIGNORE);
switch (nResult)
{
   case IDABORT:
   // 'Abort' was pressed
   break;
   case IDRETRY:
   // 'Retry' was pressed
   break;
   case IDIGNORE:
   // 'Ignore' was pressed
   break;
   } return 0;}
```