

UNIVERSIDAD DE LOS LLANOS
CURSO: FUNDAMENTOS DE PROGRAMACIÓN
PROFESORA: ING. PIEDAD CHICA SOSA
LABORATORIO DE FUNCIONES, POO, GUI EN C++

FUNCIONES

Básicamente una función puede realizar las mismas acciones que un programa:

- aceptar datos
- realizar unos cálculos determinados y, finalmente,
- devolver resultados

Las funciones son **invocadas desde otras funciones**, con una excepción: la función **global** `main()`, que tienen todos los programas en C++. Permite al compilador conocer donde está el punto inicial de un programa. Por razones obvias, `main()` no puede ser invocada desde otras funciones.

Podemos distinguir 3 características de las funciones en C++:

- la **definición**
- la **declaración**
- la **llamada**

Definición de una función

Es el código que realiza las tareas para las que la función ha sido prevista.

La primera línea de la definición recibe el nombre de **encabezamiento** y el resto, un bloque encerrado entre llaves, es el **cuerpo** de la función.

La definición de una función se debe realizar en alguno de los ficheros que forman parte del programa.

La sintaxis habitual de una definición de función es la siguiente

```
tipo nombre_funcion(tipo_1 arg_1, ..., tipo_n arg_n)
{
    sentencias;
    return expresion; // optativo
}
```


Ejemplo:

```
double calcula_media(double num1, double num2)
{
    double media;
    media = (num1 + num2)/2.;
    return media;
}
```

El tipo de una función

Una función devuelve, como mucho, **un único valor** a través de la sentencia `return`. Este valor tendrá el mismo **tipo** que el de la función.

La palabra reservada `return` permite realizar un **salto incondicional**, de la misma forma que `break`. En este caso, devuelve desde el interior de la función actual el control del programa a la función que la llamó.

Una función puede tener varios puntos en los que se devuelve un valor con `return` y, lógicamente, finalizará en el primero que se ejecute.

Al igual que con los `break`, la legibilidad del código debe ser la decisión que determine que haya un único `return` (preferible) o varios.

Declaración de una función

Al igual que ocurre con otros identificadores, en C++ no podemos llamar a una función en una sentencia sin que esté **declarada** previamente.

Una declaración explícita de una función, también denominada **prototipo** de la función, tiene la expresión general:

```
tipo nombre_funcion(tipo_1, tipo_2, ..., tipo_n);
```

En el ejemplo anterior el prototipo de la función es:

```
double calcula_media(double, double);
```


LLamada a la función

La llamada a una función se hace especificando su nombre y, entre paréntesis, las expresiones cuyos valores se van a enviar como argumentos de la función.

Estos **parámetros** pueden ser identificadores o cualquier otra expresión válida.

La llamada a una función se escribe de forma general como sigue:

```
salida = nombre_funcion(arg_1, arg_2, ..., arg_n);
```

En nuestro ejemplo podría ser:

```
double resultado = calcula_media(numero1, numero2);
```

Ejemplo 1: Edita, Compila y ejecuta el código

```
// Cálculo de la media de dos números
#include <iostream>
using namespace std;

double calcula_media(double, double); // Declaración

int main()
{
    double numero1, numero2;
    cout << "Introduzca el primer número: ";
    cin >> numero1;
    cout << "Introduzca el segundo número: ";
    cin >> numero2;

    double resultado = calcula_media(numero1, numero2); // Llamada
    cout << "La media de " << numero1 << " y " << numero2
        << " es " << resultado << ".\n";
}

// Definición
double calcula_media(double num1, double num2)
{
    double media;
    media = (num1+num2)/2.;
    return media;
}
```

Funciones `void`

En ocasiones una función no necesita devolver ningún valor y, por tanto, no es obligatorio usar `return`.

Para ello, debemos definir la función con un tipo especial llamado `void`.

Ejemplo 2: Edita, Compila y ejecuta el código

```
#include <iostream>
using namespace std;

void imprime_cadena(string cadena)
{
    cout << cadena;
    // return; // Opcional
}

int main()
{
    string cadena = "Hola";
    imprime_cadena(cadena);
    imprime_cadena(" mundo.\n");
}
```

Funciones sin argumentos

Una función puede no necesitar una lista de parámetros.

Ejemplo 3: Edita, Compila y ejecuta el código

```
#include <iostream>
using namespace std;

//Opcional void imprime_mensaje_inicial(void)
void imprime_mensaje_inicial()
{
    cout << "Este programa bla bla bla...\n";
}

int main()
{
    imprime_mensaje_inicial();
}
```


POO

El paradigma de programación orientada a objetos es una metodología especial de programación de software en la que un programa informático se diseña como la interrelación entre un conjunto de instancias conocidas como objetos. Dichos objetos son entidades que tienen un estado que está descrito por sus variables internas que se conocen como atributos o propiedades; y que tienen comportamientos que están plasmados en funciones o métodos que se utilizan para manipular las variables que describen el estado del objeto.

Ejemplo 4:

```
#include <iostream>

#include <cstdlib>

using namespace std;

class operaciones{

public:

int primero;

int segundo;

int resultado;

void sumar(){

resultado=primero+segundo;

cout<<resultado<<endl;

}

void restar(){

resultado=primero-segundo;

cout<< resultado<<endl;

}

operaciones(int,int);

};
```

```
operaciones::operaciones(int dato1, int dato2){  
    primero=dato1;  
    segundo=dato2;  
}
```

```
int main(){  
    operaciones objeto1(30,10);  
    objeto1.sumar();  
    objeto1.restar();  
    system("pause");  
    return 0;  
}
```

GUI

Se utiliza la librería Windows.h para trabajar el modo grafico e interfaces graficas de usuario en C++

El siguiente código crea una ventana vacia:

```
#include <windows.h>
```

```
const char g_szClassName[] = "myWindowClass";
```

```
// Step 4: the Window Procedure
```

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)  
{  
    switch(msg)
```

```

{
    case WM_CLOSE:
        DestroyWindow(hwnd);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
}
return 0;
}

```

```

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc;
    HWND hwnd;
    MSG Msg;

    //Step 1: Registering the Window Class
    wc.cbSize    = sizeof(WNDCLASSEX);
    wc.style     = 0;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon     = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor   = LoadCursor(NULL, IDC_ARROW);

```



```

wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);

wc.lpszMenuName = NULL;

wc.lpszClassName = g_szClassName;

wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);


if(!RegisterClassEx(&wc))
{
    MessageBox(NULL, "Window Registration Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}


// Step 2: Creating the Window
hwnd = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    g_szClassName,
    "The title of my window",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, 240, 120,
    NULL, NULL, hInstance, NULL);


if(hwnd == NULL)
{
    MessageBox(NULL, "Window Creation Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}


ShowWindow(hwnd, nCmdShow);

```



```
UpdateWindow(hwnd);
```

```
// Step 3: The Message Loop
```

```
while(GetMessage(&Msg, NULL, 0, 0) > 0)
```

```
{
```

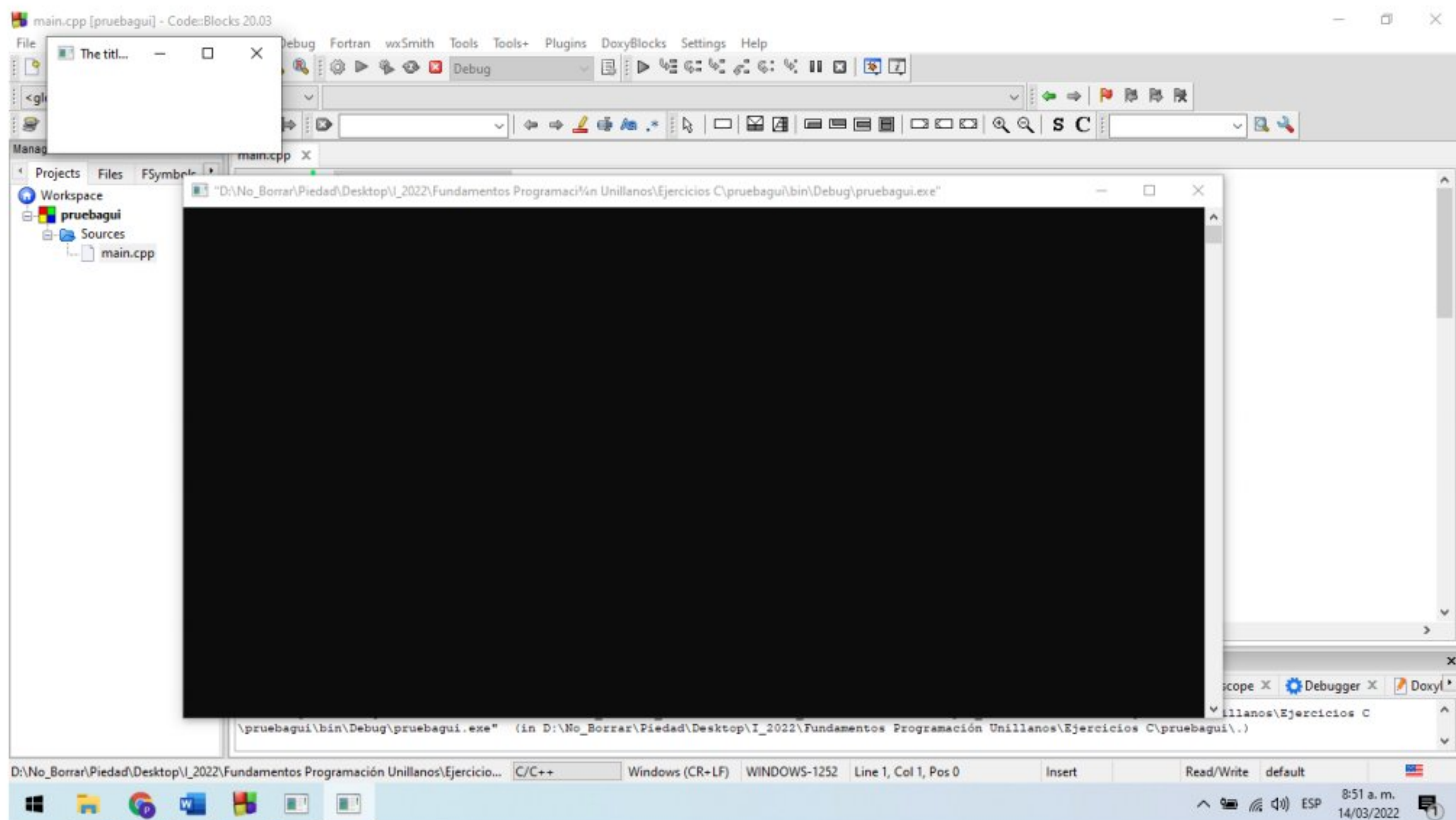
```
    TranslateMessage(&Msg);
```

```
    DispatchMessage(&Msg);
```

```
}
```

```
return Msg.wParam;
```

```
}
```



El siguiente Ejemplo nos muestra un mensaje en pantalla y las características del código C++ en una aplicación en Windows.

```
#include <windows.h>
```

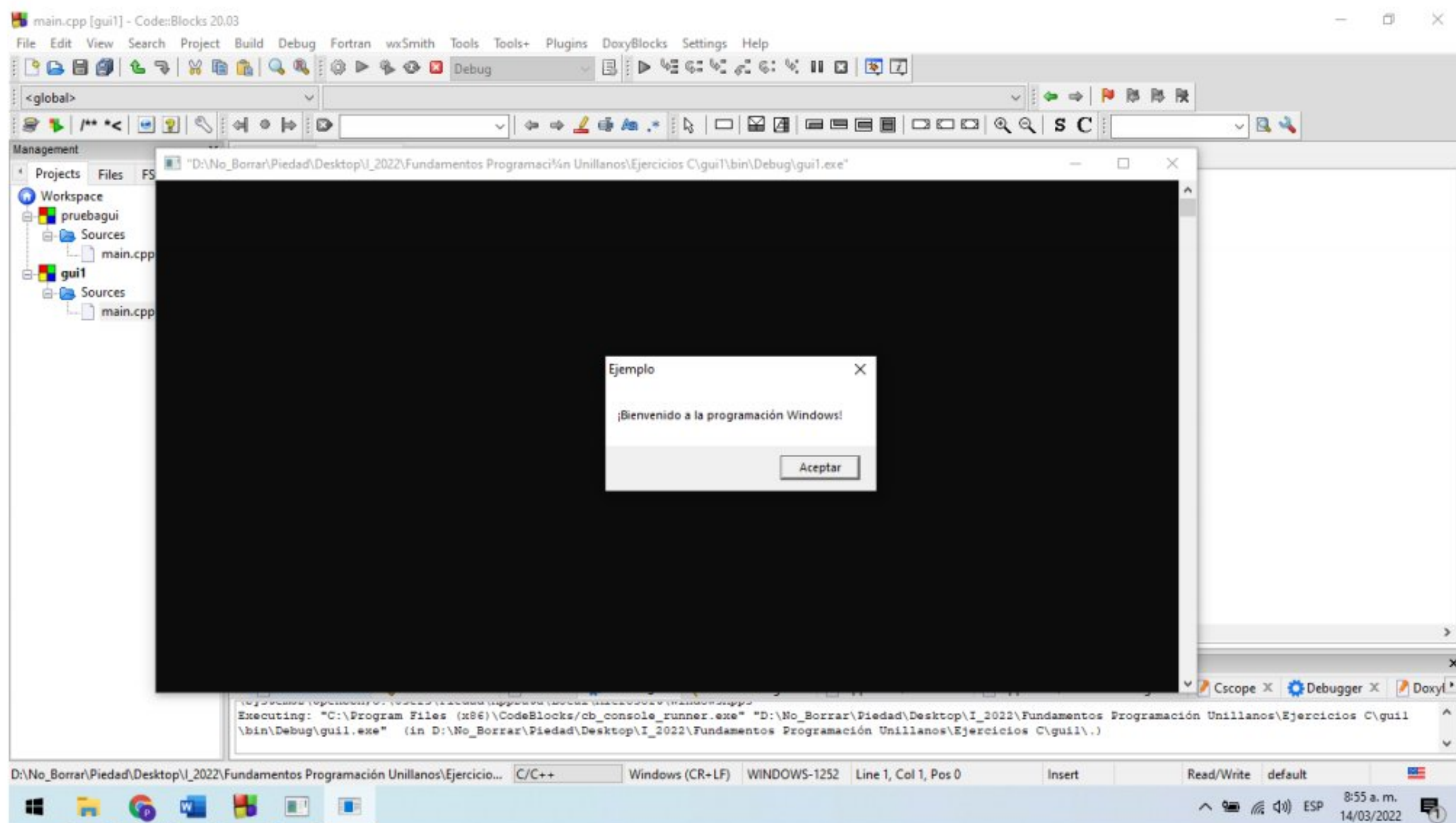
```
int WINAPI WinMain(HINSTANCE p1, HINSTANCE p2, LPSTR p3, int p4)
```

```
{
```

```
    MessageBox(NULL, "¡Bienvenido a la programación Windows!", "Ejemplo", MB_OK);
```


```
    return 0;
```

```
}
```



La función `MessageBox()` tiene 4 parámetros de los cuales ahora solo interesan los últimos 3: el segundo y tercer parámetro son punteros hacia cadenas de texto (LPCWSTR); el segundo indica el contenido del mensaje a mostrar en pantalla, y el tercero es un string que se mostrará en la barra de título del mensaje. El cuarto parámetro es un número entero sin signo (`unsigned int` o `UINT`) en el cual indicará el estilo del cuadro de mensaje.

Hasta ahora al cuarto parámetro le hemos indicado el valor MB_OK (que sabemos vale cero) porque ese número indica que quiere mostrar el botón Aceptar. Si al cuarto parámetro se le indica otro número, es posible mostrar un ícono de error o de pregunta, además de mostrar otros botones diferentes de "Aceptar". Esto no se hace al azar, ya he explicado que contamos con constantes numéricas para indicar qué botones y qué ícono mostraremos en pantalla. Aquí va una pequeña lista de constantes numéricas ya declaradas en windows.h y que puede combinar para mostrar diferentes mensajes:

Definiciones		 Copiar código
// BOTONES		
#define MB_OK 0	//botón "Aceptar"	
#define MB_OKCANCEL 1	//botones "Aceptar" y "Cancelar"	
#define MB_ABORTRETRYIGNORE 2	//botones "Anular", "Reintentar", "Omitir"	
#define MB_YESNOCANCEL 3	//botones "Sí", "No" y "Cancelar"	
#define MB_YESNO 4	//botones "Sí" y "No"	
// ICONOS		
#define MB_ICONINFORMATION 64	//ícono de información	
#define MB_ICONEXCLAMATION 0x30	//ícono de alerta, signo de exclamación	
#define MB_ICONERROR 16	//ícono de error, alto, tacha roja	
#define MB_ICONQUESTION 32	//ícono de pregunta	

Ejemplo:

```
#include <windows.h>
```

```
int WINAPI WinMain(HINSTANCE hInstancia, HINSTANCE hInstanciaPrev,  
                  LPSTR lpszCmd, int nEstadoVentana)  
{  
    MessageBox(0,"¿Guardar cambios?", "Título", MB_YESNO | MB_ICONQUESTION);  
    return 0;  
}
```

