Guía de Práctica de Laboratorio: Funciones II en C++

Objetivo General:

- Familiarizar al estudiante con el uso avanzado de funciones en C++, incluyendo la creación de proyectos estructurados y la implementación de archivos de cabecera (.h).

Parte 1: Ejercicios de Funciones

Ejercicio 1: Simulación de Cajero Automático

Objetivo: Familiarizar a los estudiantes con la creación de un menú de opciones utilizando funciones `void` en C++.

1.1: Creación del Menú de Opciones

Instrucciones:

1. **Crear el Menú de Opciones**:
   - Define una función `void` llamada `menu` que muestra un menú de opciones.
   - La función `menu` debe permitir al usuario seleccionar una opción, pero por ahora, solo mostrará mensajes indicando la opción seleccionada.
2. **Incluir la Función en el `main`**:
   - Invoca la función `menu` desde `main`.

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

```cpp
// Función para mostrar el menú y ejecutar la opción seleccionada
void menu() {
    int opcion;
    do {
        cout << "\n--- Cajero Automático ---" << endl;
        cout << "1. Depositar 100" << endl;
        cout << "2. Retirar 100" << endl;
        cout << "3. Mostrar Saldo" << endl;
        cout << "4. Salir" << endl;
        cout << "Seleccione una opción: ";
        cin >> opcion;

        switch (opcion) {
            case 1:
                //depositar();
                break;
            case 2:
                //retirar();
                break;
            case 3:
                //mostrarSaldo();
                break;
            case 4:
                cout << "Saliendo..." << endl;
                break;
            default:
                cout << "Opción no válida. Intente de nuevo." << endl;
        }
    } while (opcion != 4);
}
```

```
int main() {
    menu();
    return 0;
}
```

Parte 1.2: Implementación de las Funciones y Uso de la Variable Global

Objetivo: Implementar las funciones de depositar, retirar y mostrar saldo utilizando una variable global para el saldo.

Instrucciones:

1. **Definir una Variable Global para el Saldo**:
   - Define una variable global `int saldo` inicializada a 0.
2. **Crear las Funciones `void`**:
   - `void depositar()`: Aumenta el saldo en 100 unidades.
   - `void retirar()`: Disminuye el saldo en 100 unidades si hay suficiente saldo.
   - `void mostrarSaldo()`: Muestra el saldo actual.
3. **Modificar la Función `menu`**:
   - Actualiza la función `menu` (descomentando) para invocar las funciones `depositar`, `retirar` y `mostrarSaldo` en las opciones correspondientes.

```
// Variable global para el saldo
int saldo = 0;

// Función para depositar dinero
void depositar() {
    saldo += 100;
    cout << "Se han depositado 100 unidades. Nuevo saldo: " << saldo << endl;
}

// Función para retirar dinero
void retirar() {
    if (saldo >= 100) {
        saldo -= 100;
        cout << "Se han retirado 100 unidades. Nuevo saldo: " << saldo << endl;
    } else {
        cout << "Saldo insuficiente." << endl;
    }
}

// Función para mostrar el saldo
void mostrarSaldo() {
    cout << "Saldo actual: " << saldo << endl;
}
```

Ejercicio2: Paso por Valor y por Referencia

- **Objetivo:** Mostrar la diferencia entre pasar parámetros por valor y por referencia, y cuándo se usaría cada uno.
- **Instrucciones:**
  - Define dos funciones: una que pase un parámetro por valor y otra por referencia.
  - Modifica el valor del parámetro dentro de cada función y muestra el resultado.

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

void porValor(int a) {
    a = 10;
    cout << "Dentro de porValor, a = " << a << endl;
}

void porReferencia(int &a) {
    a = 10;
    cout << "Dentro de porReferencia, a = " << a << endl;
}

int main() {
    int num = 5;
    cout << "Antes de pasar por valor, num = " << num << endl;
    porValor(num);
    cout << "Luego de pasar por valor, num = " << num << endl;

    cout << "Antes de pasar por referencia, num = " << num << endl;
    porReferencia(num);
    cout << "Luego de pasar por referencia, num = " << num << endl;

    return 0;
}
```

- Paso por valor para evitar modificar la variable original, y paso por referencia para permitir modificaciones.

Ejercicio 3: Uso de cmath

Objetivo: Usar varias funciones matemáticas de la biblioteca `cmath`.

Instrucciones:

1. Solicita un número al usuario.
2. Realiza varias operaciones matemáticas con el número: raíz cuadrada, valor absoluto, potencia, redondeo, seno, coseno y tangente.

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

int main() {
    double num;
    cout << "Ingrese un numero: ";
    cin >> num;

    cout << "Valor absoluto: " << fabs(num) << endl;
    cout << "Raíz cuadrada: " << sqrt(num) << endl;
    cout << "Potencia (n^2): " << pow(num, 2) << endl;
    cout << "Redondeo al entero más cercano: " << round(num) << endl;
    cout << "Redondeo hacia arriba: " << ceil(num) << endl;
    cout << "Redondeo hacia abajo: " << floor(num) << endl;
    cout << "Seno: " << sin(num) << endl;
    cout << "Coseno: " << cos(num) << endl;
    cout << "Tangente: " << tan(num) << endl;

    return 0;
}
```

Ejercicio 4: Generación de Números Aleatorios con cstdlib

Objetivo: Generar números aleatorios usando la biblioteca `cstdlib`.

Instrucciones:

1. Define un juego de "Número Mágico" donde el usuario adivina un número entre 1 y 100.
2. Usa `rand` y `srand` para generar números aleatorios.

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
    srand(time(0)); // Inicializa el generador de números aleatorios
    int numeroMagico = rand() % 100 + 1;
    int intento;

    cout << "Adivina el numero mágico (entre 1 y 100): ";
    while (true) {
        cin >> intento;
        if (intento > numeroMagico) {
            cout << "Muy alto. Intenta de nuevo: ";
        } else if (intento < numeroMagico) {
            cout << "Muy bajo. Intenta de nuevo: ";
        } else {
            cout << "Correcto El numero mágico es " << numeroMagico << endl;
            break;
        }
    }
    return 0;
}
```

Parte 2: Creación de un Proyecto en Dev C++ y Archivo de Cabecera

Objetivo: Crear un proyecto en Dev C++ y utilizar archivos de cabecera para modularizar el código.

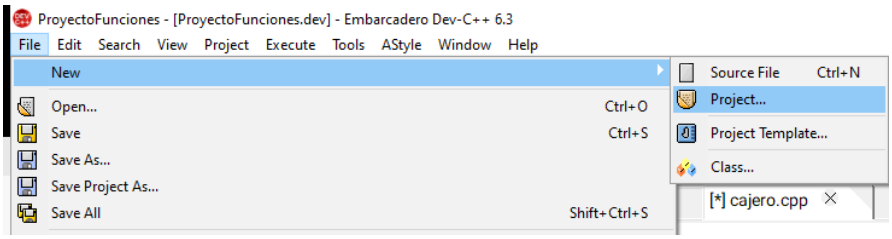
2.1 Creación de un Proyecto en Dev C++

Iniciar Dev C++:

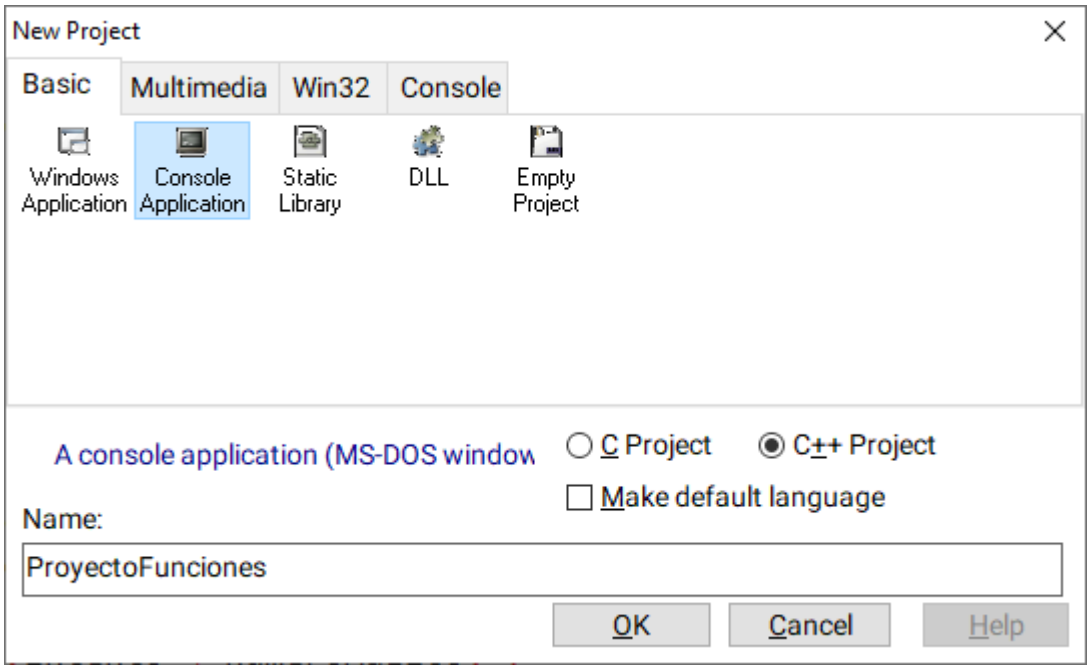
1. Abre el entorno de desarrollo Dev C++ desde tu escritorio o menú de aplicaciones.

Crear un nuevo proyecto:

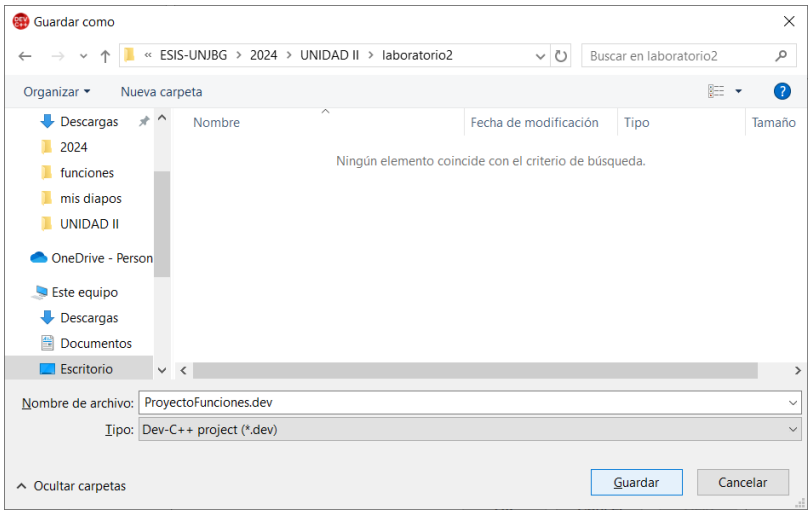
1. Ve a File > New > Project.



2. Selecciona Proyecto de consola.
3. Elige C++ Project.
4. Asigna un nombre al proyecto, por ejemplo, ProyectoFunciones.



5. Selecciona una ubicación para guardar el proyecto y haz clic en Aceptar.



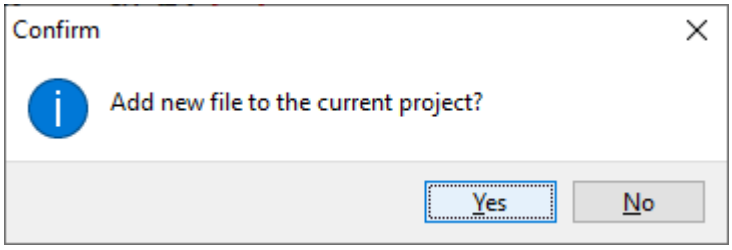
Configurar el proyecto:

1. Dev C++ generará automáticamente una estructura de carpetas y archivos para tu proyecto. Asegúrate de que el proyecto se ha creado correctamente verificando que ves los archivos y carpetas en el explorador de proyectos de Dev C++.

Parte 2: Creación de Archivos de Código

Crear el archivo de encabezado (funciones.h):

1. Ve a File > New > Source File



Responda Yes

2. Escribe el siguiente código en el nuevo archivo:

```
#ifndef FUNCIONES_H
#define FUNCIONES_H

int suma(int a, int b);
int resta(int a, int b);

#endif
```

3. Guarda el archivo como funciones.h en la carpeta del proyecto.

Crear el archivo de implementación (funciones.cpp):

1. Ve a File > New> Source File
2. Escribe el siguiente código en el nuevo archivo:

```
#include "funciones.h"

int suma(int a, int b) {
    return a + b;
}

int resta(int a, int b) {
    return a - b;
}
```

3. Guarda el archivo como funciones.cpp en la carpeta del proyecto.

Crea o modifica el archivo principal (main.cpp):

1. Ve a Archivo > Nuevo > Archivo de fuente.
2. Escribe el siguiente código en el nuevo archivo:

```cpp
#include <iostream>
#include "funciones.h"

using namespace std;

int main() {
    int a = 10, b = 5;
    cout << "Suma: " << suma(a, b) << endl;
    cout << "Resta: " << resta(a, b) << endl;
    return 0;
}
```

3. Guarda el archivo como main.cpp en la carpeta del proyecto.
4. Compila el proyecto

Preguntas de Comprensión:

1. ¿Qué es una variable global y cómo se utiliza en el programa del cajero automático?
2. ¿Cuál es la diferencia entre pasar un parámetro por valor y por referencia?
3. ¿Qué función matemática usaría para redondear un número al entero más cercano y cómo se utiliza?
4. ¿Cuál es la ventaja de utilizar archivos de cabecera (.h) en proyectos de C++?

EJERCICIOS DE FUNCIONES II

1. **Cálculo del Área de un Círculo:** Implementa un programa que calcule el área de un círculo dado su radio. Utiliza la constante `M_PI` de la librería `cmath` para obtener el valor de pi. Solicita al usuario el radio del círculo y utiliza la función `pow` de `cmath` para calcular el cuadrado del radio. Muestra el resultado del área calculada.

2. **Generador de Números Aleatorios:** Crea un programa que genere y muestre una serie de números aleatorios entre 0 y un número máximo especificado por el usuario. El programa solicitará al usuario la cantidad de números aleatorios que desea generar y el valor máximo permitido. Utiliza la función `rand` de la librería `cstdlib` para generar los números aleatorios.

3. **Cálculo de la Hipotenusa:** Crea un programa que calcule la hipotenusa de un triángulo rectángulo utilizando la fórmula de Pitágoras. Solicita al usuario los valores de los catetos y utiliza la función `sqrt` de `cmath` para calcular la raíz cuadrada de la suma de los cuadrados de los catetos. Muestra el resultado de la hipotenusa calculada.

4. **Cálculo de Raíces de una Ecuación Cuadrática:** Crea un programa que calcule las raíces de una ecuación cuadrática utilizando la fórmula general. Utiliza la función `sqrt` de la librería `cmath` para calcular la raíz cuadrada. Solicita al usuario los coeficientes de la ecuación (a, b, c) y muestra las raíces obtenidas. Considera los casos en los que las raíces son reales o complejas.

5. **Función calcularRaices:** Amplía el ejercicio anterior definiendo una función `void` llamada `calcularRaices` que reciba los coeficientes de la ecuación (a, b, c) por valor y las variables donde se almacenarán las raíces por referencia. Además, incluye una variable por referencia que indique si las raíces son reales o complejas. La función debe calcular las raíces utilizando la fórmula general y actualizar las variables correspondientes.

6. **Juego de Piedra, Papel y Tijeras:** Desarrolla un programa que simule el juego de piedra, papel y tijeras contra la computadora. El usuario elegirá su jugada (piedra, papel o tijeras) y la computadora generará su jugada de forma aleatoria. El programa determinará el ganador de cada ronda según las reglas del juego. El juego continuará hasta que el usuario o la computadora gane 3 rondas. Muestra el resultado final indicando el ganador del juego.