C++ IV

C/C++

- Variables Punteros
  - Operador * y &
  - Operadores new/delete
  - Aritmética de punteros
  - Segfault
  - Punteros múltiples
  - Punteros a funciones
- **Orientación a objetos**
  - **Objetos**
  - **Clases**
    - **Puntero this**
  - **Métodos**
    - **Constructor**
    - **Polimorfismo**
  - **Atributos**
  - **Encapsulación**
  - **Instanciación**
  - Herencia
    - Clases Abstractas

- Templates
- Sobrecarga de operadores

Orientación a objetos

La orientación a objetos es un paradigma de programación que se centra en la creación de objetos y la manipulación de esos objetos para realizar una tarea o resolver un problema. Se basa en el concepto de "clases", que son plantillas para crear objetos, y en la "herencia", que permite crear nuevas clases a partir de clases existentes y así reutilizar el código.

La orientación a objetos también incluye otros conceptos clave como la encapsulación, que se refiere a la ocultación de la complejidad interna de un objeto y la exposición solo de una interfaz sencilla para interactuar con él; el polimorfismo, que permite que diferentes objetos respondan de manera diferente a un mismo mensaje o método; y la abstracción, que se refiere a la capacidad de simplificar y generalizar conceptos complejos para que puedan ser entendidos y utilizados más fácilmente en la programación.

En resumen, la orientación a objetos es una forma de organizar y estructurar el código de una aplicación o programa, basada en la creación y manipulación de objetos que interactúan entre sí para resolver un problema o realizar una tarea específica.

Orientación a objetos

- Objetos
  - Es una variable creada a partir de una clase (instancia)
  - Es una variable que permite acceder a los datos y las funciones(métodos) de la clase
- Clase:
  - Es una plantilla que describe el objeto
  - Contiene variables que son llamadas atributos
  - Contiene funciones que son llamadas métodos
- Encapsulación
  - Permite cambiar la visibilidad atributos y métodos para con las palabras claves public, protected y private
- Atributos
  - Variables de la clase
- Métodos
  - Funciones de la clase
- Constructor
  - Es un método especial con el que se inicializa la clase.
  - No retorna valor
  - Los parámetros con con los que se instancian los objetos
- Destructor
  - Es un método especial para finalizar “limpiar” la clase.
  - No retorna valor
  - Se llama automáticamente

Clases

- Se declaran con la palabra class
- El bloque de código de la clase se declara entre {} y se finaliza con ;
- El constructor se tiene que llamar igual que la clase y recibe los parámetros para iniciarla.
- El destructor se tiene que llamar igual que la clase pero debe tener ~ antes del nombre en la declaración.
- Existe un puntero especial llamado “this” que permite llamar los atributos y métodos de la clase, no siempre se tiene que llamar explícitamente.

```cpp
#include<iostream>
#include<string>

using namespace std;

string mensaje="mensaje global";

class Saludo
{
    private:
    string mensaje;

    public:
    Saludo(string mensaje)
    {
        this->mensaje = mensaje;
        cout<<"Contructor llamado"<<endl;
    }

    ~Saludo(){
        cout<<"Destructor llamado"<<endl;
    }

    void saludar()
    {
        cout<<mensaje<<endl;
    }
};

int main()
{
    Saludo s("hola clases");
    s.saludar();
    //cout<<s.mensaje<<endl;
    return 0;

}
```

Ejemplo partícula

- Operador de asignación (=) en dos objetos: En algunos casos, puede que no produzca un error en tiempo de compilación, pero puede causar un comportamiento inesperado en tiempo de ejecución. Esto es especialmente cierto si la clase contiene punteros o recursos de memoria dinámica, ya que puede llevar a fugas de memoria o a problemas de acceso a memoria.

  Por lo tanto, es importante que las clases que se utilizan en operaciones de asignación sobrecarguen el operador de asignación (=) para definir su comportamiento de manera adecuada.

- Cuando el objeto es puntero para acceder a sus atributos y métodos se y el operador -> en vez de .
- Se puede usar la palabra clave **auto** en la instanciación para definir la variable.

https://github.com/jmejiagu/BigData-UdeARootBasicCurse-20231/blob/main/codes/Unidad1/particula.cxx

Cabeceras y Código fuente

El código de c++ se puede dividir en dos partes:

- Cabeceras: prototipos de las clases y funciones
- Código fuente: donde están las definiciones

Veamos un ejemplo usando la clase partícula:

Particle.h Particle.cxx ParticleMain.cxx

g++ ParticleMain.cxx Particle.cxx -I.