Introducción C++

SSEE 2022 - Pío IX

Programación Orientada a Objetos (POO)

- **Abstracción:** descripción de un modelo
- **Encapsulamiento:** definición del comportamiento interno y externo
- **Polimorfismo:** sobrecarga de acciones (métodos/funciones)
- **Herencia:** adquisición de características de algo ya definido

Definición de una clase

```
class Clase
{
    public:
        Clase();
        ~Clase();

        int   metodo1(char var1);
        short metodo2(void);

    private:
        int   variable1;
        float variable2;
        char  variable3[50];
};
```

- Una clase es un descriptor de un conjunto de objetos con estructura, comportamiento y relaciones similares.
- Es una abstracción de la realidad.
- Funciona como molde para objetos que puedan ser descriptos por ella.
- Al instanciar una **clase** se crea un **objeto**. Pueden crearse muchos objetos a partir de una clase y cada objeto tendrá sus propios datos.

Ejemplo de una clase

```
class Auto
{
    public:
        Auto();
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

Las etiquetas public y private son especificadores de acceso a miembros.

Public: cualquier variable o método definido en este sector será accesible dentro y fuera de la clase.

Private: las variables o métodos definidos dentro de este especificador sólo serán accesibles dentro de la clase. Suele usarse para variables que no queremos que sean modificadas por fuera.

Ejemplo de una clase

```
class Auto
{
    public:
        Auto();
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

Constructor de la clase:

- El constructor inicializa una instancia de la clase y tiene el mismo nombre que la clase.
- Pueden existir varios constructores sobrecargados que reciban diferentes parámetros de inicialización.

Destructor de la clase:

- El destructor se ejecutará cuando se destruya una instancia de la clase.
- No es necesario llamar al destructor, este se ejecutará automáticamente.

Ejemplo de una clase

```
class Auto
{
    public:
        Auto();
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

Métodos de la clase:

- **Modificador (set):** es una función que modifica algún parámetro del objeto.
- **Consultor (get):** retorna un valor desde su objeto pero no lo modifica. Nos permite acceder indirectamente a sus miembros privados para leerlo.

Ejemplo de una clase

```
class Auto
{
    public:
        Auto();
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

Variables o características de la clase:

- Definirán al objeto y lo harán distinto a cualquier otro objeto que se instancie de la misma clase.

Creación de una biblioteca

Aseguro una única inclusión del archivo auto.h, la primera vez que se vea se creará el define AUTO_H y si se instanciase de nuevo el ifndef sería falso.

```
#ifndef AUTO_H
#define AUTO_H

class Auto
{
    public:
        Auto();
        Auto(float kilometraje_inicial);
        Auto(bool luces_inicial, bool estado_inicial);
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion_);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};

#endif // AUTO_H
```

```
#include "clase.h"

Auto::Auto()
{
    kilometraje = 0;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}

Auto::Auto(float kilometraje_inicial)
{
    kilometraje = kilometraje_inicial;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}

Auto::Auto(bool luces_inicial, bool estado_inicial)
{
    kilometraje = 0;
    luces = luces_inicial;
    estado = estado_inicial;
    temperatura = 50;
    aceleracion = 0;
}
```

Incluyo el archivo .h o header en el .cpp o source, para luego realizar todas las definiciones de las funciones declaradas.

Archivos de la clase

Auto.h

```
class Auto
{
    public:
        Auto();
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion_);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

Auto.cpp

```
Auto::Auto()
{
    kilometraje = 0;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}

Auto::~Auto()
{

}

void  Auto::Prender(void)
{
    estado = 1;
}
void  Auto::Acelerar(float aceleracion_)
{
    aceleracion = aceleracion_;
}
void  Auto::AccionarLuces(bool estado)
{
    luces = estado;
}

float Auto::ObtenerKilometraje(void)
{
    return kilometraje;
}

int   Auto::ObtenerTemperatura(void)
{
    return temperatura;
}
```

Sobrecarga de funciones

- Como mencionamos al principio, una de las características de la POO es el polimorfismo. Este nos permitirá definir funciones con el mismo nombre, pero que recibirán distintos parámetros en función de mis necesidades en el programa.

Sobrecarga de constructores

```
class Auto
{
    public:
        Auto();
        Auto(float kilometraje_inicial);
        Auto(bool luces_inicial, bool estado_inicial);
        ~Auto();

        void  Prender(void);
        void  Acelerar(float aceleracion_);
        void  AccionarLuces(bool estado);

        float ObtenerKilometraje(void);
        int   ObtenerTemperatura(void);

    private:
        float kilometraje;
        int   temperatura;
        bool  estado;
        bool  luces;
        float aceleracion;
};
```

```
Auto::Auto()
{
    kilometraje = 0;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}

Auto::Auto(float kilometraje_inicial)
{
    kilometraje = kilometraje_inicial;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}

Auto::Auto(bool luces_inicial, bool estado_inicial)
{
    kilometraje = 0;
    luces = luces_inicial;
    estado = estado_inicial;
    temperatura = 50;
    aceleracion = 0;
}
```

Sobrecarga de constructores e instanciación

```cpp
int main()
{
    Auto auto1;
    Auto auto2(115503.58);
    Auto auto3(true, true);

    return 0;
}
```

Instancia al objeto mediante el constructor por defecto que no recibe ningún parámetro.

Instancia mediante el constructor de kilometraje inicial.

Instancia mediante el constructor de luces y estado inicial.

```cpp
Auto::Auto()
{
    kilometraje = 0;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}
```

```cpp
Auto::Auto(float kilometraje_inicial)
{
    kilometraje = kilometraje_inicial;
    luces = 0;
    estado = 0;
    temperatura = 50;
    aceleracion = 0;
}
```

```cpp
Auto::Auto(bool luces_inicial, bool estado_inicial)
{
    kilometraje = 0;
    luces = luces_inicial;
    estado = estado_inicial;
    temperatura = 50;
    aceleracion = 0;
}
```

Acceso a los miembros del objeto

```
int main()
{
    Auto auto1;
    Auto auto2(115503.58);
    Auto auto3(true, true);

    auto1.Prender();
    auto2.Acelerar(2.35);
    int temperatura3 = auto3.ObtenerTemperatura();

    return 0;
}
```

Ingreso y muestra de datos por consola en C++

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

int main()
{
    int valor1;
    char string1[50];
    float valor2;

    /*
    cout -> printf sin necesidad de especificar formato
    cin -> scanf sin necesidad de especificar formato
    */

    cout << "Ingrese un valor entero: ";
    cin >> valor1;
    cin.get(); // Atrapo el enter que queda en el flujo de entrada
    cout << valor1 << endl; // endl es el equivalente al \n

    cout << "Ingrese un string: ";
    cin.getline(string1, sizeof(string1));
    cout << string1 << endl;

    cout << "Ingrese un valor decimal: ";
    cin >> valor2;
    cout << valor2 << endl;

    return 0;
}
```

Defino que voy a utilizar el namespace std el cual tiene asociadas consigo un conjunto de funciones, en este caso usaremos cin y cout del std.

Impresión: cout <<, con la posibilidad de concatenar <<.

Obtención: cin >>, con la posibilidad de concatenar >>.

Ejercicio

- Diseñar una clase **Alumno** que posea los siguientes miembros y métodos:
  - **Miembros:**
    - Notas (máximo 20)
    - Cantidad de presentes y ausentes
  - **Métodos:**
    - void CargarNota(unsigned char nota)
    - void CargarAsistencia(bool presencia)
    - float ObtenerPromedio()
    - unsigned char ObtenerPresentes()
    - unsigned char ObtenerAusentes()
- Crear también un programa que cargue notas y asistencia, y luego imprima por pantalla lo devuelto por los métodos consultores.