Lenguaje de programación C++

Objetos

Objetos

En la teoría definirán, o verán múltiples definiciones de lo que son los objetos.

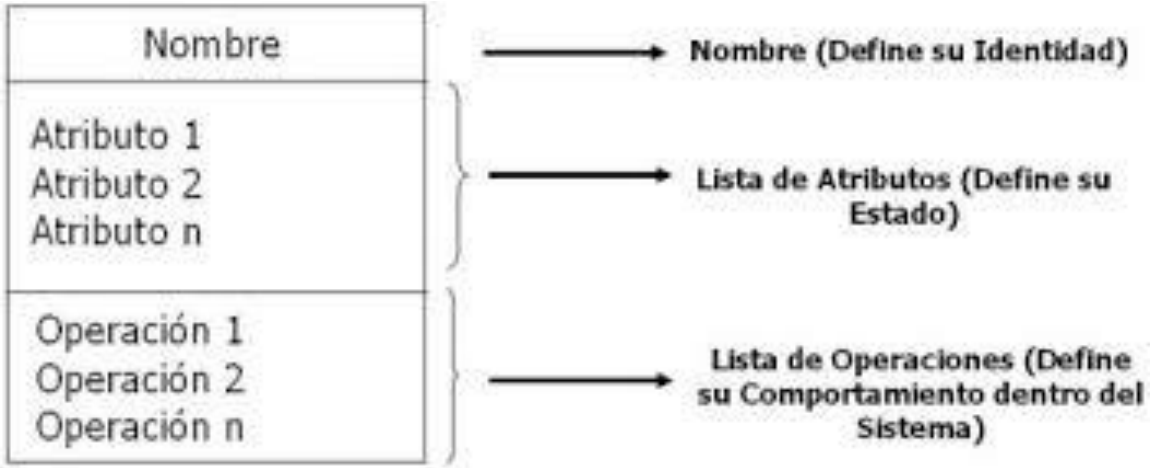
Tenemos que tener en cuenta que los lenguajes orientados a objetos son anteriores (algunos de ellos por supuesto) a la formulación o generalización del paradigma, por lo tanto, no siempre van a cumplir con la totalidad de las caracteristicas.

Objetos

- Representan a un objeto físico o virtual del mundo real
- Cada objeto tiene *“vida”* propia, independiente de otro aunque sea del mismo tipo de objeto.
- Un objeto puede estar compuesto de otro objeto.
- Poseen características, *atributos* (estos conforman el *estado* del objeto)
- Dicho estado es propio del objeto que se analiza, no implica que el resto de los objetos del mismo tipo lo comparta. Ejemplo: que el auto A este encendido no implica que todos los autos lo estén.
- Tienen métodos para manipular a los atributos y al comportamiento del objeto. a ellos llamamos “interfaz” del objeto.

La Clase

Es la descripción del objeto. Su diseño.



Clases y Objetos

- Una Clase será usada como un tipo de dato más.

- Los objetos son instancias de una clase. Pueden existir múltiples objetos por clases y cada uno tendrá un estado diferente

Ejemplo: Tenemos la clase Persona, y dos instancias de dicha clase, la Persona Pablo y María.

Clases y Objetos

Una clase se define de la siguiente forma:

```
class Nombre{
public:
    variables //atributos
    ...
    funciones //métodos
    ...
private:
    variables
    ...
    funciones
    ...
}
```

Veremos con mejor detalle la diferencia entre métodos públicos y privados.

Esto permite proteger a los datos miembros y dar una interfaz más correcta y segura.

Clases y Objetos

Encapsulamiento:

- Los atributos y los métodos miembros pueden ser públicos o privados.
    - Los públicos pueden ser accedidos desde cualquier punto del programa
    - Los privados solo pueden accederse desde métodos miembros de la clase.
- Por defecto, la declaración de permiso es privada

Es recomendable que la clase se mantenga simple, o sea, solo tenga los atributos necesarios representativos del objeto y los métodos cumplan funciones definidas y únicas.

Clases y Objetos

Veamos el siguiente ejemplo:

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

class prueba{
 int a;

};

int main (){
 prueba X;
 X.a = 0;
 return 0;
}
```

Al compilar nos dice los siguiente:

```
$ g++ prueba.cpp
prueba.cpp: En la función ‘int main()’:
prueba.cpp:11:4: error: ‘int prueba::a’ is private
within this context
   11 |  X.a = 0;
      |    ^
prueba.cpp:5:9: nota: declared private here
    5 |     int a;
      |         ^
```

Clases y Objetos

El scope de variables es similar a C.

Un atributo de la clase puede ser accedido por cualquier método de la misma clase y se mantendrá mientras dure la “vida” del objeto.

Si la variable es declarada dentro de la función sólo es vista por dicha función.

En C++ no se justifica el usar funciones globales, excepto la declaración del *main*. El resto de las funciones deben pertenecer a un objeto correspondiente.

Puntero **this**

Sirve para autoreferenciar a la clase dentro de los métodos miembros. Ejemplo:

```
class Costo{
    int valor;
public:
    void setValor(int valor){
        this->valor= valor;
    }
    Costo& incremento(){
        valor++;
        return *this;
    }
};
```

```
int main() {
    Cost c;
    c.setValor(20);
    c.incremeto();
}
```

Para diferenciar a la variable input de la miembro se utiliza this.

Para retornarse a sí mismo, el objeto también utiliza this.

Métodos

Si el método es simple puede ser definido dentro de la clase (inline)

Lo recomendable es hacerlo fuera de ella. Para esto se debe usar el operador de ámbito `::`

```
class Costo{
    int valor;
public:
    void setValor(int);
    Costo& incremento();
};
```

```
Costo& Costo::incremento(){
    valor++;
    return *this;
}

void Costo::setValor(int valor){
    this->valor= valor;
}
```

Estado, Eventos y Métodos

Brevemente podemos definir como:

- Estado: Son los valores que poseen los atributos de un objeto en un momento determinado
- Evento: Es la acción o situación en que un objeto cambia de estado.
- Método: es una función que realiza el cambio de estado de un objeto.

Dos eventos y por lo tanto métodos fundamentales en C++ son:

- Evento de creación -> constructor
- Evento de destrucción -> destructor

Constructores y Destructores

- Función del Constructor:
  - Es llamado automáticamente al instanciarse el objeto
  - Aloca memoria
  - Inicializa variables
  - Sirve para hacer lo que queremos apenas inicia el objeto.

- Función del Destructor
  - Es llamado automáticamente cuando el objeto deja de existir
  - Liberar la memoria que se haya reservado
  - Cualquier otra acción que se desee en el momento de que el objeto sea eliminado.

Constructores y Destructores

Ejemplo de constructor

```
class Punto {
// private (default)
    int m_x; // X coordinate
    int m_y; // Y coordinate
public:
    Punto(){
        m_x = 0;
        m_y = 0;
        cout << "Point object initialized" <<
endl;
    }
    Punto(int, intÇ); //otro contructor
// resto de las funciones
};
```

```
Punto::Punto(int x, int y){
    m_x = x;
    m_y = y;
}

int main() {
    Punto p1;
    Punto *pPto;
    pPto = new Punto(2, 4);
    delete pPto;

    return 0;
}
```

Constructores y destructores

Ejemplo de destructor

```
class Poligono {
    int NoPuntos;
    Punto* aPuntos;
public:
    Poligono() : NoPuntos(0), aPuntos(NULL) {}
    Poligono(int);
// Destructor:
    ~Poligono();

};
```

```
Poligono::~Poligono() {
    if (aPoints != NULL)
        delete[]aPoints;
}
```

```
/*
Poligono() : NoPuntos(0), aPuntos(NULL) {}
es equivalente a
Poligono(){NoPuntos = 0; aPuntos = NULL;}
*/
```

Constructores y Destructores

Hay que tener en cuenta dos situaciones en caso del constructor y destructor

Cuando el objeto es local de una función y no es un puntero, el constructor y destructor se llaman automáticamente. Cuando el objeto está alocado, el destructor se debe invocar.

```
int main (){
  Objeto A; // se llama al constructor por default
  Objeto B(1, 5); // se llama al constructor específico

  Objeto* C; // no invoca a ningún constructor
  C = new Objeto() // se instancia el objeto

  delete(c); // se invoca al destructor
  /* se llama automáticamente los destructores de A y B*/
  return 0;
}
```

Constructores y Destructores

En caso que no se definan constructores y destructores explícitamente, C++ los hará por defecto y solo reservarán el espacio de memoria que correspondan para guardar al objeto (o liberar la memoria en caso del destructor)

Esto no es lo recomendable. La correcta inicialización de variables, la reserva dinámica de memoria, o la liberación de la misma en caso que sea necesario es el modo de trabajo ordenado y evita comportamientos erróneos