Estructura de datos

Dra. Karla R.

Programa

1. **Estructuras fundamentales de datos**
2. **Pilas y colas**
3. **Listas**
4. **Recursión y Arboles**

Diagnóstico

1. Lenguajes de programación que dominas
2. ¿Qué es una estructura de datos?
3. ¿Qué es un vector?
4. En estructura de datos ¿Qué es una matriz?
5. Indique una definición de pila y cola
6. ¿Qué es una recursión?

Unidad 0

Introducción a Lenguaje C++

Introducción a C++

- Los lenguajes de programación soportan en sus compiladores tipos de datos fundamentales o básicos (predefinidos), tales como int, char y float.
- Lenguajes de programación, como C++, añaden sus propios tipos de datos.
- Un tipo de dato definido por el programador se denomina tipo abstracto de dato, TAD (Abstract Data Type, ADT).
- La modularización de un programa utiliza la noción de tipo abstracto de dato (TAD) siempre que sea posible.
- Los paradigmas más populares soportados por el lenguaje C++ son: programación estructurada y programación orientada a objetos.

El mínimo programa de C++

El mínimo programa de C++ es:

main() { }

- Se define una función (main) que no tiene argumentos y no hace nada.
- Las llaves { } delimitan un bloque en C++, en este caso el cuerpo de la función main.
- Todos los programas deben tener una función main() la cual se ejecuta al comenzar el programa.
- Un programa es una secuencia de sentencias, directivas de compilación y comentarios.
- Las sentencias simples se separan por punto y coma y las compuestas se agrupan en bloques mediante llaves.
- Las directivas son instrucciones que indican al compilador que realice alguna operación antes de compilar nuestro programa, comienzan con el símbolo # y no llevan punto y coma.

Un programa simple

```
/* Este es un programa mínimo en C++, lo único que hace es escribir una frase en la pantalla */
#include <iostream>
using std ;
int main()
{
        cout << "Hola mundo“ << endl; // imprime en la pantalla la frase “hola mundo”
 }
```

Ejercicio 1

Realizar un algoritmo que lea tres números; si el primero es positivo calcule el producto de los tres números, y en otro caso calcule la suma.

```
inicio
1 leer los tres números Numero1, Numero2,
Numero3
2 si el Numero1 es positivo
calcular el producto de los tres números
escribir el producto
3 si el Numero1 es no positivo
calcular la suma de los tres números
escribir la suma
fin
```

Codificación en C++

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
  int Numero1, Numero2, Numero3, Producto, Suma;
  cin >> Numero1 >> Numero2 >>Numero3;
  if(Numero1 > 0)
  {
    Producto = Numero1* Numero2 * Numero3;
    cout <<"El producto de los números es" << Producto;
  }
  else
  {
    Suma = Numero1 + Numero2 + Numero3;
    cout <<" La suma de los números es:" << Suma;
  }
  return 0;
}
```

Ejercicio 2

Diseñar un algoritmo que permita identificar si un número es primo o no

Algoritmo	Pseudocodigo en C++

algoritmo primo
inicio
variables
entero: n, x;
lógico: primo;
leer(n);
x ←2; primo ← verdadero;
mientras primo **y** (x < n) **hacer**
  si n mod x <> 0 **entonces**
  x ← x+1
  sino
  primo ← falso
  fin si
fin mientras
si (primo) **entonces**
  escribe('es primo')
sino
  escribe('no es primo')
fin si
Fin
	#include <cstdlib>
	#include <iostream>
	using namespace std;
	int main(int argc, char *argv[])
	{
	  int n, x;
	  bool primo;
	  cin >> n;
	  x = 2;
	  primo = true;
	  while (primo &&(x < n))
	  if (n % x != 0)
	  x = x+1;
	  else
	  primo = false;
	  if (primo)
	  cout << "es primo";
	  else
	  cout << " no es primo";
	  return 0;
	}

TIPOS ABSTRACTOS DE DATOS

- Un tipo de dato definido por el programador se denomina *tipo abstracto de datos* (**TAD**)
- Por ejemplo, en C++ el tipo Punto, que representa a las coordenadas x e y de un sistema de coordenadas rectangulares, no existe. Sin embargo, es posible implementar el tipo abstracto de datos, considerando los valores que se almacenan en las variables y qué operaciones están disponibles para manipular estas variables.
- En esencia un tipo abstracto de datos es un tipo que consta de datos (estructuras de datos propias) y operaciones que se pueden realizar sobre esos datos.
- Un **TAD** se compone de *estructuras de datos* y los *procedimientos* o *funciones* que manipulan esas estructuras de datos.

Un tipo abstracto de Datos puede definirse mediante la ecuación:

TAD = Representación (Datos) + Operaciones (Funciones, procedimientos)

Las unidades de programación de lenguajes que pueden implementar un **TAD** reciben distintos nombres:

- Modula-2 *módulo*
- Ada *paquete*
- C++ *clase*
- *C#* *clase*
- Java *clase*

Ejemplo

- En el siguiente ejemplo, la clase hora, tiene datos separados de tipo **int** para *horas, minutos y segundos.*
- Un constructor inicializara este dato a 0, y otro lo inicializará a valores fijos,
- Una función miembro deberá visualizar la hora en formato $11:59:59$
- Otra función miembro sumará dos objetos de tipo **hora** pasados como argumentos
- La función principal **main()** creará dos objetos inicializados y otro que no esta inicializado
- Se suman los dos valores inicializados y se deja el resultado en el objeto no inicializado
- Por último se muestra el valor resultante.

Ejemplo

```
#include <cstdlib>
#include <iostream>
using namespace std;
class hora
{
    private:
         int horas, minutos, segundos;
    public:
         hora(){ horas = 0; minutos = 0;
         segundos = 0; }
    hora(int h, int m, int s){horas = h;
         minutos = m; segundos = s;}
    void visualizar();
    void sumar(hora h1, hora h2 );
};
void hora::visualizar()
{
    cout << horas << ":" << minutos <<
":" << segundos << endl;
}
```

```
void hora::sumar(hora h1, hora h2 )
{
   segundos = h2.segundos + h1.segundos;
   minutos = h2.minutos + h1.minutos +
             segundos / 60;
   segundos = segundos % 60;
   horas = h2.horas + h1.horas + minutos
           / 60;
   minutos = minutos % 60;
}
int main(int argc, char *argv[])
{
   hora h1(10,40,50), h2(12,35,40), h;
   h1.visualizar();
   h2.visualizar();
   h.sumar(h1,h2);
   h.visualizar();
   return 0;
}
```

PROGRAMACIÓN ESTRUCTURADA

- Un programa en un lenguaje procedimental es un conjunto de instrucciones o sentencias.
- Cuando los programas se vuelven muy grandes, la lista de instrucciones aumenta considerablemente, de modo tal que el programador tiene muchas dificultades para controlar ese gran número de instrucciones.
- Para resolver este problema los programas se descompusieron en unidades más pequeñas que adoptaron el nombre de *funciones* (*procedimientos*, *subprogramas* o *subrutinas*).
- De este modo un programa orientado a procedimientos se divide en funciones, de modo que cada función tiene un propósito bien definido y resuelve una tarea concreta, y se diseña una interfaz claramente definida (el prototipo o cabecera de la función) para su comunicación con otras funciones.

Datos locales y datos globales

En un programa procedimental, existen dos tipos de datos.

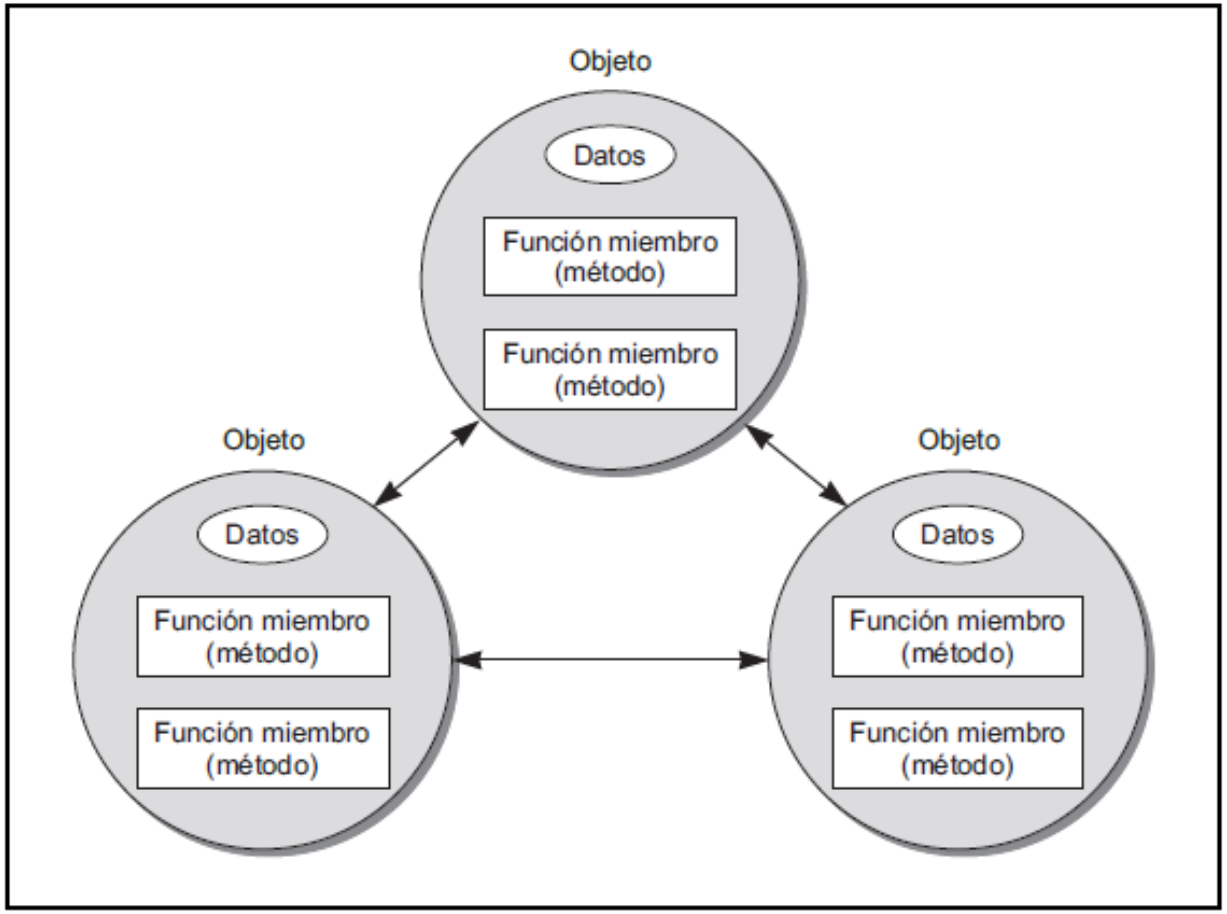
Datos locales que son ocultos en el interior de la función y son utilizados, exclusivamente, por la función.

Datos globales a los cuales se puede acceder desde *cualquier* función del programa. Es decir, dos o más funciones pueden acceder a los mismos datos siempre que éstos sean globales.

PROGRAMACIÓN ORIENTADA A OBJETOS

- La idea fundamental de los lenguajes orientados a objetos es combinar en una única unidad o módulo, tanto los datos como las funciones que operan sobre esos datos. Tal unidad se llama **objeto**.
- Las funciones de un objeto se llaman *funciones miembro* en C++, y son el único medio para acceder a sus datos. Los datos de un objeto, se conocen también como *atributos* o *variables de instancia.* Si se desea leer datos de un objeto, se llama a una función miembro del objeto.
- Se accede a los datos y se devuelve un valor. No se puede acceder a los datos directamente.
- Los datos están ocultos, de modo que están protegidos de alteraciones accidentales.
- Los datos y las funciones se dice que están *encapsulados en una única entidad.*

- Un programa C++ se compone, normalmente, de un número de objetos que se comunican unos con otros mediante la llamada a otras funciones miembro. La llamada a una función miembro de un objeto se denomina *enviar un mensaje* a otro objeto.



En el paradigma orientado a objetos, el programa se organiza como un conjunto finito de objetos, que contiene datos y operaciones (funciones miembro en C++), que llaman a esos datos y que se comunican entre sí mediante mensajes

¿Qué son clases?

- En términos prácticos, una **clase** es un tipo definido por el usuario. Booch denomina a una clase como "un conjunto de objetos que comparten una estructura y comportamiento comunes".
- Una clase contiene la especificación de los datos que describen un objeto junto con la descripción de las acciones que un objeto ha de ejecutar. Estas acciones se conocen como *servicios* o *métodos.*
- Una clase incluye también todos los datos necesarios para describir los objetos creados a partir de la clase. Estos datos se conocen como *atributos, variables* o *variables instancia*.

DECLARACIÓN DE UNA CLASE

- Antes de que un programa pueda crear objetos de cualquier clase, la clase debe ser *declarada* y los métodos definidos (implementados). La declaración de una clase significa dar a la misma un nombre, darle nombre a los elementos que almacenan sus datos y describir los métodos que realizarán las acciones consideradas en los objetos.

Formato

```
class NombreClase
{
        lista_de_miembros
};
```

`NombreClase` Nombre definido por el usuario que identifica a la clase (puede incluir letras, números y subrayados como cualquier identificador).
`lista_de_miembros` Datos y funciones miembros de la clase.

Ejemplo

Definición de una clase llamada *Punto*, que contiene las coordenadas X y Y, de un punto en un plano

```
//archivo Punto.h
class Punto
{
   private:
     int x, y; // coordenadas x, y
   public:
     Punto(int x_, int y_) // constructor
  {
     x = x_;
     y = y_;
  }
  Punto() { x = y = 0;} // constructor sin argumentos
  int leerX() const; // devuelve el valor de x
  int leerY() const; // devuelve el valor de y
  void fijarX(int valorX) // establece el valor de x
  void fijarY(int valorY) // establece el valor de y
};
```

La definición de las funciones miembro se realiza en el archivo `Punto.cpp`:

```
#include "Punto.h"
int Punto::leerX() const
{
   return x;
}
int Punto::leerY() const
{
   return y;
}
void Punto::fijarX(int valorX)
{
   x = valorX;
}
   void Punto::fijarY(int valorY)
{
    y = valorY;
}
```

Objetos

Una vez que una clase ha sido definida, un programa puede contener una *instancia* de la clase, denominada un *objeto de la clase*. Un objeto se crea de forma estática, de igual forma que se define una variable. También de forma dinámica, con el operador new aplicado a un constructor de la clase. Por ejemplo, un objeto de la clase Punto inicializado a las coordenadas (2,1):

```
Punto p1(2, 1); // objeto creado de forma estática
Punto* p2 = new Punto(2, 1); // objeto creado dinámicamente
```

Formato para crear un objeto

```
NombreClase varObj(argumentos_constructor);
```

Formato para crear un objeto dinámico

```
NombreClase* ptrObj;
ptrObj = new NombreClase(argumentos_constructor);
```

Toda clase tiene una o más funciones miembro, denominadas constructores, para inicializar el objeto cuando es creado; tienen el mismo nombre que el de la clase, no tienen tipo de retorno y pueden estar sobrecargados.

El *operador de acceso* a un miembro del objeto, selector punto (.), selecciona un miembro individual de un objeto de la clase. Por ejemplo:

```
Punto p2; // llama al constructor sin argumentos
p2.fijarX(10);
cout << " Coordenada x es " << p2.leerX();
```

El otro *operador de acceso* es el selector *flecha* (->), selecciona un miembro de un objeto desde un puntero a la clase. Por ejemplo:

```
Punto* p;
p = new Punto(2, -5); // crea objeto dinámico
cout << " Coordenada y es " << p -> leerY();
```

Los miembros de la clase

Un principio fundamental en programación orientada a objetos es la *ocultación de la información, que s*ignifica que determinados datos del interior de una clase no se puede acceder por funciones externas a la clase.

El mecanismo principal para ocultar datos es ponerlos en una clase y hacerlos *privados*. A los datos o métodos privados sólo se puede acceder desde dentro de la clase. Por el contrario, los datos o métodos públicos son accesibles desde el exterior de la clase

Para controlar el acceso a los miembros de la clase se utilizan tres diferentes *especificadores de acceso:* public, private y protected. Cada miembro de la clase está precedido del especificador de acceso que le corresponde.

Formato

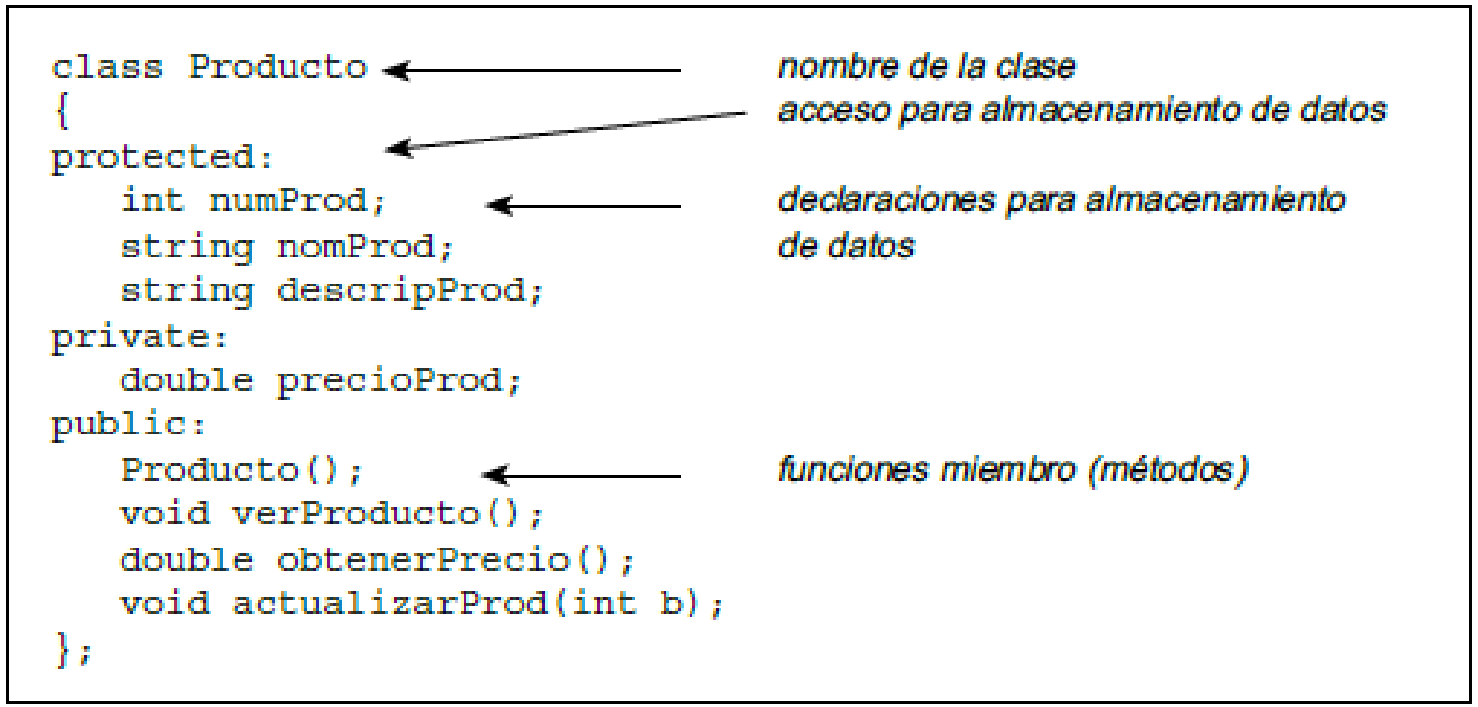
```
class NombreClase
{
   private:
     declaraciones de miembros privados;
   protected:
     declaraciones de miembros protegidos;
   public:
     declaraciones de miembros públicos;
};
```

El ***principio de ocultación*** de la información indica que toda la interacción con un objeto se debe restringir a utilizar una interfaz bien definida que permite que los detalles de implementación de los objetos sean ignorados. Por consiguiente, los datos y métodos públicos forman la interfaz externa del objeto, mientras que los elementos ***privados*** son los aspectos internos del objeto que no necesitan ser accesibles para usar el objeto.

El principio de encapsulamiento significa que las estructuras de datos internas utilizadas en la implementación de una clase no pueden ser accesibles directamente al usuario de la clase

Funciones miembro de una clase

La declaración de una clase incluye la declaración o prototipo de las funciones miembros (métodos). Aunque la implementación se puede incluir dentro del cuerpo de la clase (inline), normalmente se realiza en otro archivo (con extensión .cpp) que constituye la definición de la clase.



CONSTRUCTORES Y DESTRUCTORES

Un *constructor* es una función miembro que se ejecuta automáticamente cuando se crea un objeto de una clase. Sirve para inicializar los miembros de la clase.

El constructor tiene el mismo nombre que la clase. Cuando se define no se puede especificar un valor de retorno, nunca devuelve un valor. Sin embargo, puede tomar cualquier número de argumentos.

Ejemplo:

La clase `Rectangulo` tiene un constructor con cuatro parámetros

```
class Rectangulo
{
   private:
    int izdo, superior;
    int dcha, inferior;
   public:
    Rectangulo(int iz, int sr, int d, int inf) // constructor
    {
      izdo = iz; superior = sr;
      dcha = d; inferior = inf;
    }
   // definiciones de otros métodos miembro
};
```

Al crear un objeto se pasan los valores al constructor, con la misma sintaxis que la de llamada a una función. Por ejemplo:

```
Rectangulo Rect(4, 4, 10, 10);
Rectangulo* ptr = new Rectangulo(25, 25, 75, 75);
```

Se han creado dos instancias de `Rectangulo`, pasando valores concretos al constructor de la clase.

Constructor por defecto, se trata de un constructor que no tiene parámetros, o que se puede llamar sin argumentos, normalmente inicializa los miembros dato de la clase con valores por defecto.

```
class Complejo
{
   private :
      Double x;
      double y;
      public:
   Complejo(double r = 0.0, double i = 1.0)
   {
       x = r;
       y = i;
     }
};
```

Cuando se crea un objeto `Complejo` puede inicializarse a los valores por defecto, o bien a otros valores.

```
Complejo z1; // z1.x == 0.0, z1.y == 1.0
Complejo* pz = new Complejo(-2, 3); // pz -> x == -2, pz -> y == 3
```

Constructores sobrecargados

Los constructores sobrecargados son bastante frecuentes, proporcionan diferentes alternativas de inicializar objetos.

```
class EquipoSonido
{
   private:
     int potencia, voltios;
     string marca;
   public:
     EquipoSonido() // constructor por defecto
     {
       marca = "Sin marca"; potencia = voltios = 0;
     }
     EquipoSonido(string mt)
     {
       marca = mt; potencia = voltios = 0;
     }
     EquipoSonido(string mt, int p, int v)
     {
       marca = mt;
       potencia = p;
       voltios = v;
     }
};
```

A continuación se crean tres objetos:

```
EquipoSonido rt; // constructor por defecto
EquipoSonido ft("POLASIT");
EquipoSonido gt("PARTOLA", 35, 220);
```

Destructor

El destructor es necesario implementarlo cuando el objeto contenga memoria reservada dinámicamente.

Ejemplo: Se declara la Clase Equipo con dos atributos de tipo puntero, un constructor con valores por defecto y un destructor

```
class Equipo
{
   private:
     Jugador* jg;
     int numJug;
     int actual;
   public:
     Equipo(int n = 12)
     {
       jg = new Jugador[n];
       numJug = n; actual = 0;
     }
   ~Equipo() // destructor
   {
     if (jg != 0) delete [] jg;
   }
}
```

Ejercicios

1. Implementar la clase Fecha con miembros dato para el mes, día y año. Cada objeto de esta clase representa una fecha, que almacena el día, mes y año como enteros. Se debe incluir un constructor por defecto, funciones de acceso, la función reiniciar (int d, int m, int a) para reiniciar la fecha de un objeto existente, la función adelantar(int d, int m, int a)para avanzar una fecha existente (día d, mes m, y año a) y la función imprimir(). Escribir una función auxiliar de utilidad, normalizar(), que asegure que los miembros dato están en el rango correcto $1 \leq$ año, $1 \leq$ mes ≤ 12, día $\leq$ días (Mes), siendo días(Mes) otra función que devuelve el número de días de cada mes.
2. Ampliar el programa anterior, de modo que pueda aceptar años bisiestos. **Nota:** un año es bisiesto si es divisible por 400, o si es divisible por 4 pero no por 100.

Unidad 1.

Estructuras de datos fundamentales