Algoritmos y Estructuras de Datos
Ingeniería en Informática, Curso 2º

SEMINARIO DE C++
Sesión 2

Contenidos:

1. Definición de clases
2. Implementación de los métodos
3. Constructores y destructores
4. Objetos dinámicos
5. Algunas notas

Ejercicios

1. Definición de clases

- La **programación orientada a objetos** se basa en el uso de clases y objetos.
  - **Modelo imperativo:** un programa es una sucesión de instrucciones que se ejecutan secuencial o iterativamente.
  - **Modelo orientado a objetos:** un programa está formado por un conjunto de objetos que se comunican entre sí.

- **Naturaleza dual de las clases:**
  - Una clase es un **tipo abstracto de datos** (mecanismo para crear nuevos tipos definidos por el usuario). Mecanismo similar a las estructuras, pero incluyendo tanto los datos y como las funciones del tipo.
  - Una clase es un **módulo** que proporciona **encapsulación** (agrupa funciones y datos) y **ocultamiento** de la implementación (parte `private` y parte `public`).

- Un **objeto** es una variable cuyo tipo es una clase.

```
class persona                    // persona es una clase
{
   char nombre[80];
   long dni, telefono;
   int edad;

   void leer (FILE *f);
   void escribir (void);
};

persona p1;                      // p1 es un objeto de clase persona
p1.leer(f1);
p1.edad++;
p1.escribir();
```

- **Nomenclatura:**
  - **Miembros de una clase:** atributos y operaciones de la clase (datos y funciones miembro). Ej.: `nombre`, `dni` y `edad` son datos miembros de la clase `persona`.
  - **Métodos de una clase:** funciones definidas dentro de esa clase. Ej.: `leer()` y `escribir()` son métodos de la clase `persona`.
  - **Mensaje:** invocación de un método sobre un objeto. Ej.: `p1.leer(f1);` y `p1.escribir();` son mensajes aplicados sobre el objeto `p1`.
  - **Objeto receptor:** el objeto receptor de un mensaje es el objeto sobre el que se aplica. Ej.: en `p1.leer(f1)`, el objeto receptor es `p1`.

- **Declaración de una clase. Sintaxis.**

```
class nombre {
  ...            // Miembros de la clase
};
```

- **Miembros públicos y privados.**
  - **Miembros públicos. `public:`** Son accesibles para el usuario de la clase.
  - **Miembros privados. `private:`** No son visibles para el usuario de la clase.

```
class nombre {
  private:
     ...       // Miembros privados
  public:
     ...       // Miembros públicos
};
```

  - Pueden aparecer varias veces (y en distinto orden) las cláusulas `public:` y `private:`
  - Por defecto, si no se indica nada, los miembros son `private`.

- **Declaración de los miembros** de una clase: Igual que la definición de variables y funciones.
  - No se permite inicialización de los datos miembros.
  - En las funciones miembro, el objeto receptor es un parámetro implícito, no hace falta indicarlo.

- **Ejemplo.**

```
class conjuntoInt {
  private:
    int tamano;
    int datos[MAXIMO];
  public:
    void vaciar ();
    void insertar (int n);
    void suprimir (int n);
    bool miembro (int n);
};

class vacia {};

class listaFloat {
  private:
    listaFloat * siguiente;
    listaFloat * anterior;
  public:
    float actual;
    int longitud ();
    void insertar (float valor);
    listaFloat * avanzar ();
    listaFloat * retroceder ();
};
```

2. Implementación de los métodos

- Igual que la implementación de una función.

- Normalmente, la implementación de los métodos va aparte (fuera) de la definición de la clase. Se utiliza el operador de visibilidad '`::`' para el nombre del método.

```
void conjuntoInt::vaciar ()
{
   ...
}

void conjuntoInt::insertar (int n)
{
   ...
}
```

- Dentro de un método todos los miembros de esa clase (tanto públicos como privados) son accesibles como si fueran variables (o funciones) locales.

```
void conjuntoInt::insertar (int n)
{
  if (tamano<MAXIMO)
    datos[tamano++]= n;
  else
    cerr << "ERROR: el conjunto está lleno.";
}
```

- **Métodos in-line.**
  - También se puede implementar el método dentro de la definición de la clase.
  - Los métodos de este tipo se llaman **in-line**: el compilador, al encontrar un mensaje, sustituye la llamada directamente por el código del método.
  - No es muy aconsejable, a menos que el método sea trivial.

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

class contadorModulo10 {
  private:
    int estado;
  public:
    void iniciar() {estado= 0;}
    void avanzar(int n= 1) {estado= (estado+n)%10;}
    int valor() {return estado;}
};

main ()
{
   contadorModulo10 cnt;
   cnt.iniciar();             // ¿Qué ocurre si quitamos esto?
   cnt.avanzar(12);
   cnt.avanzar();             // ¿Cuánto avanza aquí?
   cnt.avanzar();
   cout << cnt.valor();       // ¿Qué debería imprimir?
}
```

- **Ejemplo.** Clase conjunto de enteros con tamaño máximo limitado.

```
#include <iostream>
#define MAXIMO 100
using namespace std;

////////////// Declaración de la clase conjuntoInt /////////////
class conjuntoInt {
  private:
    int tamano;
    int datos[MAXIMO];
  public:
    void vaciar ();
    void insertar (int n);
    bool miembro (int n);
};

////////////// Implementación de los métodos /////////////
void conjuntoInt::vaciar ()
{
  tamano= 0;
}

void conjuntoInt::insertar (int n)
{
  if (tamano<MAXIMO)
    datos[tamano++]= n;     // Ojo, no se comprueba si ya está el n
  else
    cerr << "ERROR: el conjunto está lleno.";
}

bool conjuntoInt::miembro (int n)
{
   for (int i= 0; i<tamano; i++)
     if (datos[i]==n)
       return true;
   return false;
}

////////////// Programa principal /////////////
int main (void)
{
   conjuntoInt cjt;
   cjt.vaciar();             // ¿Qué pasa si lo quitamos?
   char comando;
   cout << ">> ";
   while ((cin>>comando) && (comando!='q')) {
      int valor;
      if (comando=='i') {
         cin >> valor;
         cjt.insertar(valor);
         cout<<"Insertado el "<<valor;
      }
      else if (comando=='m') {
         cin >> valor;
         cout<<"¿Miembro "<<valor<<"? "<<(cjt.miembro(valor)?"SI":"NO");
      }
      else
         cout << "Comando " << comando << " no válido. ";
      cout << "\n>> ";
   }
   return 0;
}
```

3. Constructores y destructores

- **Constructor**: es una operación de inicialización de un objeto.

- Por defecto, si no se definen constructores, los datos miembros de un objeto no se inicializan.

- El **constructor** de una clase es un método de esa clase, con el mismo nombre que la clase.

```
class conjuntoInt {
     ...
  public:
     conjuntoInt () {tamano= 0;}     // Constructor del tipo
     ...
};
```

  - El constructor no devuelve ningún valor.
  - Se puede aplicar **sobrecarga**: pueden haber distintos constructores, siempre que los parámetros de entrada sean distintos.

```
class conjuntoInt {
     ...
  public:
     conjuntoInt () {tamano= 0;} // Constructor de conjunto vacio
     conjuntoInt (int e1)          // Constructor de cjt. con 1 elem.
             {datos[0]= e1; tamano= 1;}
     conjuntoInt (int e1, int e2) // Constructor de cjt. con 2 elem.
             {datos[0]= e1; datos[1]= e2; tamano= 2;}
     ...
};
```

- **Constructor por defecto:** si existe algún constructor sin parámetros (o con todos los parámetros por defecto) se toma como constructor por defecto.

- **Uso de constructores.**

```
conjuntoInt cjt;     // Se inicializa con el constructor por defecto
conjuntoInt cjt(9);  // Constructor que incluye 1 elemento
conjuntoInt cjt(23,12);  // Constructor que incluye 2 elementos
...
```

- **Destructor:** operación de eliminación de un objeto.
  - El nombre del destructor es '`~NombreClase`'.
  - No devuelve ningún valor, ni recibe ningún parámetro.
  - Es necesario definir un destructor si el objeto ha reservado memoria dinámica o ha abierto algún fichero.

```
class conjuntoInt {
     ...
  public:
     ~conjuntoInt () {cout<<"Nada";} // Destructor, irrelevante aquí
    ...
};
```

4. Objetos en memoria dinámica

- **Igual** que con cualquier otro tipo de datos. Se pueden crear nuevos objetos en memoria dinámica con **`new`**, **`new[]`** y eliminarlos con **`delete`** y **`delete[].`**

`nombreClase *obj1= new nombreClase;`
→ Crear un nuevo objeto usando el constructor por defecto.

`nombreClase *obj2= new nombreClase(p1, p2, ...);`
→ Crear un nuevo objeto usando un constructor específico.

`nombreClase *ad= new nombreClase[tamano];`
→ Crear un array dinámico de objetos con el constructor por defecto.

`delete obj1;` → Borra el objeto dinámico, usando el destructor.
`delete[] ad;` → Borra un array dinámico de objetos, usando el destructor.

- **Ejemplo.** En el ejemplo de la clase `conjuntoInt` incluir el siguiente código en la función `main`. Recordar el operador flecha '`->`': `a->b` equivale a `(*a).b`

```
conjuntoInt *c1= new conjuntoInt;
c1->insertar(19);
c1->insertar(81);
cout << c1->miembro(19);
delete c1;
```

- **Ejemplo.** Cambiar la definición de la clase `conjuntoInt` para que se pueda especificar (y pueda variar) el tamaño máximo del conjunto.

```
class conjuntoInt {
  private:
    int tamano;
    int MAXIMO;    //Tamaño máximo variable. Quitar el #define MAXIMO 100
    int *datos;
  public:
    conjuntoInt (int max= 20);  //Constructor, por defecto tamaño 20
    ~conjuntoInt ();            //Destructor, liberar memoria
    ...                         //Esto no cambia
};

conjuntoInt::conjuntoInt (int max)
{
  tamano= 0;
  MAXIMO= max;
  datos= new int[MAXIMO];
}

conjuntoInt::~conjuntoInt ()
{
  delete[] datos;
}

...            // Lo demás queda igual
```

☞ • **Ejemplo.** Árbol binario de cadenas.

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

////////////// Declaración de la clase arbolBinario ////////////
class arbolBinario {
  private:
    char *raiz;
    arbolBinario *izq, *der;
  public:
    arbolBinario (char *cRaiz, arbolBinario *hijoIzq= NULL,
                  arbolBinario *hijoDer= NULL);
    ~arbolBinario ();
    void preorden ();
    const char *obtenRaiz() {return raiz;};
    arbolBinario *obtenHijoIzq() {return izq;};
    arbolBinario *obtenHijoDer() {return der;};
};

////////////// Implementación de los métodos ////////////
arbolBinario::arbolBinario (char *cRaiz, arbolBinario *hijoIzq,
                            arbolBinario *hijoDer)
{
   raiz= cRaiz;
   izq= hijoIzq;
   der= hijoDer;
}

arbolBinario::~arbolBinario ()     //Destructor
{
   delete izq;
   delete der;
}

void arbolBinario::preorden ()
{
   cout<<raiz;
   if (izq) {
      cout<<", ";
      izq->preorden();
   }
   if (der) {
      cout<<", ";
      der->preorden();
   }
}

////////////// Programa principal ////////////
main()
{
   arbolBinario *a, *b, *c, *d;
   a= new arbolBinario("A");
   b= new arbolBinario("B");
   c= new arbolBinario("C", a, b);
   d= new arbolBinario("D", c, new arbolBinario("1"));
   cout<<"Preorden de c: ";
   c->preorden();
   cout<<"\nPreorden de d: ";
   d->preorden();
   delete d;      //Todos los demás están incluidos aquí dentro
}
```

5. Algunas notas

- **Descomposición modular.** Igual que en C: ficheros cabecera (**.hpp**) y ficheros implementación (**.cpp**).
  - Cada módulo puede contener una o varias clases relacionadas.
  - La definición de las clases va en los ficheros de cabecera.
  - La implementación de los métodos va en los ficheros de implementación.

- **Memoria dinámica.** Llevar cuidado, toda la memoria dinámica reservada debe ser liberada.
  - **Solución:** el encargado de reservar memoria es también encargado de liberarla.
  - No liberar "más de una vez".
  - Los literales de tipo cadena de texto (por ejemplo, `c= "Hola"`, ...) se reservan estáticamente, no hace falta liberar memoria.

- Tanto los datos como las funciones pueden ser públicos o privados. Pero:
  - Normalmente los **datos son privados**: la representación interna del tipo debe estar oculta para el usuario.
  - Las **funciones son públicas** (incluidos los constructores y destructores), excepto las que sean de *uso interno*, que serán privadas.
  - **Clases amigas:** permite a una clase acceder a la representación interna de otra.

```
class conjuntoInt {
   friend class diccionario;   // La clase diccionario puede acceder a
                               // la parte private de conjuntoInt
```

  - Evitar en lo posible las clases amigas. La funcionalidad propia de un tipo debe aparecer en la definición de la clase correspondiente, no fuera.

Ejercicios

1. En el ejemplo de la clase `arbolBinario`, definir una operación `inorden()` que liste los elementos de un árbol en inorden. Introducir el siguiente código en la función `main` y predecir el resultado antes de ejecutar.

```
#define na(A,B,C)  new arbolBinario ((A),(B),(C))
   arbolBinario *a= na("1", na("2", na("3", NULL, NULL),
            na("4", NULL, na("5", NULL, NULL))), na("6", NULL, NULL));
   a->preorden();
   cout<<"\n";
   a->inorden();
   delete a;
```

2. Modificar la definición y la implementación de la clase `conjuntoInt`, para que el tamaño no esté limitado por un máximo. Si al insertar un elemento se ha llegado a la capacidad máxima, se deberá crear otro array dinámico con más elementos (p.ej. el doble de elementos). Modificar también la función `insertar` para que un elemento no se inserte más de una vez. Incluir una operación `mostrar()` para mostrar por pantalla todos los elementos del conjunto.

3. Utilizando el código para reserva, liberación y manipulación de matrices de la primera sesión del seminario de C++, definir e implementar una clase `matriz` de matrices dinámicas de `double`. Las dimensiones de la matriz se deben especificar en el constructor de la clase.