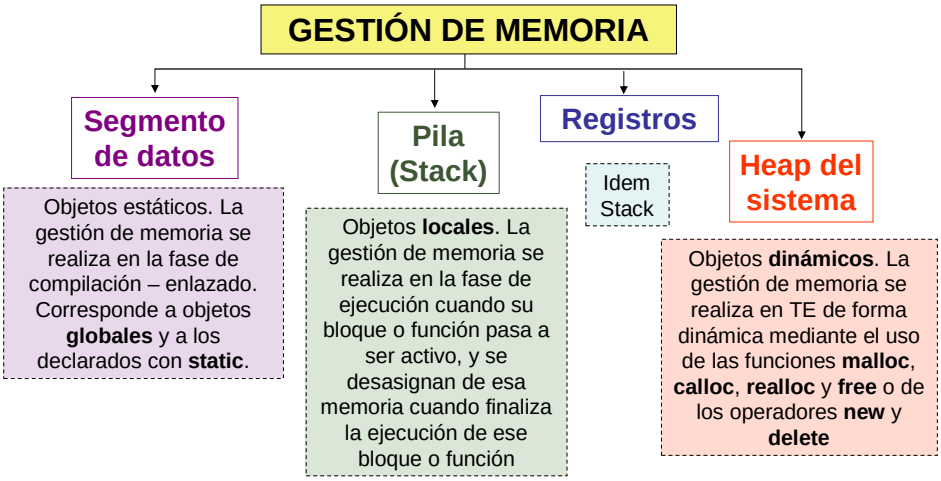
PARADIGMAS DE PROGRAMACIÓN
Licenciatura en Informática
Programador Universitario

UNIDAD III
PARADIGMAS DE PROGRAMACIÓN
Conceptos varios
(miembros const, static, gestión de memoria y puntero this)

Gestión de Memoria



2

Gestión dinámica de Memoria

Operador new

El operador **new** crea en forma automática un objeto del tamaño apropiado, si existe un constructor disponible, lo llama y retorna la dirección de la memoria asignada al objeto ó NULL si no hay espacio suficiente

Sintaxis

```
<objeto-puntero> = new <tipo-objeto>[<inicializador>];
```

```
int *pi = new int;  //asigna un entero de 2 bytes
char *pc = new char[80];//asigna un vector de 80 caract.
```

Inicializador

Se puede **inicializar el objeto asignado**.
Si no se inicializa el objeto se crea con valor indefinido

```
float *p = new float(10.4);//asigna e inicializa *p = 10.4
```

new vs malloc

```
float *p = (float *) malloc(sizeof(float));// C
float *p = new float(7);                   // C++
```

3

Gestión dinámica de Memoria

Operador delete

La memoria asignada desde el heap por el operador **new** es liberada por el operador **delete**.
El resultado tiene tipo **void**

Sintaxis

```
delete <objeto-puntero>
delete [] <objeto-puntero>; // caso de arrays
```

```
int *pt = new int (4);  // *pt == 4
delete pt;    //libera la memoria gestionada por new
```

Importante `delete [] nombreArrayObjetos;`

Libera la memoria asignada con new a un array de objetos de clase. El operador [] indica que debe invocar el **destructor** de la clase para cada uno de los objetos del arreglo

```
class X {/*...*/};
X *px = new X[10];  // asigna un vector de 10 objetos de X
delete[] px;
/*libera la memoria asignada al vector de la clase X e invoca al
destructor de la clase X para destruir los objetos creados*/
```

Puntero this

Puntero this → Cada O mantiene un apuntador a sí mismo, llamado **puntero this**

- **this** es una variable local de tipo puntero a la clase, disponible en el cuerpo de cualquier función miembro no estática.
- Cualquier función miembro tiene acceso al puntero this del O para el cual está siendo invocada a fin de poder manipular correctamente los datos de ese O

  `C++ usa el puntero this, que se pasa como argumento implícito a la función y especifica la dirección de inicio de ese O`
- Formas de uso
  - **Implícita:** para referenciar miembros datos y funciones miembros de un O, se referencia a través del nombre
  - **Explícita** utilizando `this ->miembro`

5

Puntero this

```
void Pila::push(int x)
{
    if(tope+1 < MAX)
    {   tope++;
        arreglo[tope] = x;
    }
}
```

Referencia Implícita

```
void Pila::push(int x)
{
    if(this->tope+1 < this->MAX)
    {   this->tope++;
        arreglo[this->tope] = x;
    }
}
```

Referencia Explícita

```
Pila& Pila::operator=(Pila& P)
{
    if(this!= &P) //Controlo que NO se esté asignando
    {             //el objeto a si mismo  ( P = P; )

      delete[] this->arreglo; //Libero la memoria del arreglo del
      this->MAX = P.MAX;      //Obj. implícito y seteo sus datos miembros
      this->tope =P.tope;     //con los valores del Obj. asignado
      this->arreglo = new int[this->MAX];
      for(int i=0; i<=this->tope; i++)
        this->arreglo[i]=P.arreglo[i];
    }
    return *this;   //Retorno el Obj. Implícito
}
```

Miembros de clase estáticos

Miembros de Instancia → **Copia local** de los datos miembros de la clase que posee cada O

Miembros Estáticos → Denominados **miembros de clase** representan información que es aplicable a TODA la clase

- Los datos y las funciones miembro se pueden declarar estáticos
- Se utiliza el especificador de almacenamiento **static**
- Si un dato miembro se define como estático sólo existe una copia de él para todos los O de la clase
- Si cualquier O altera el contenido de un dato miembro estático, este cambio queda reflejado para todos los O de la clase
- Un miembro estático puede ser accedido sin referenciar un O de la clase

7

Miembros de clase estáticos

Miembro Estático

- Los miembros de clase estáticos **existen** aún cuando no existan objetos de dicha clase
- La declaración de un dato miembro estático no es una definición, por lo que su uso exige una definición posterior

```
int CuentaObjeto::objCreados = 0;
int CuentaObjeto::objDestruidos = 0;
```

- **Una función miembro puede ser declarada static si no tiene acceso a miembros de clase no estáticos**
- Los datos miembros estáticos pueden ser públicos, privados o protegidos

8

Miembros de clase estáticos

Miembro Estático

Los **datos miembros estáticos públicos** son accesibles a través de cualquier O de la clase o a través del nombre de la clase, usando el operador de resolución de alcance binario

```
nombreClase objeto;
objeto.miembroDatoEstatico;
objeto.funcionMiembroEstatica();

nombreClase::miembroDatoEstatico;
nombreClase::funcionMiembroEstatica();
```

Para acceder a los **datos miembros estáticos privados o protegidos** se debe usar las funciones miembros públicas de la clase

9

Miembros de clase estáticos

IMPORTANTE
Las funciones miembro estáticas
NO ESTAN LIGADAS A OBJETOS DE LA CLASE
∴ no disponen del puntero this

```
class clase{
    int uno, dos;
    static int tres;
 public:
    int suma(){ return uno + dos + tres;       }

    static int calcular(int valor){
          int aux;
          aux = tres * valor;
          //aux = this->uno + dos + tres; //ERROR
          //aux = this->suma() + tres; //ERROR
          return aux;
   }
};
```

10

Miembros de clase estáticos

IMPORTANTE
Las funciones miembro estáticas
NO ESTAN LIGADAS A OBJETOS DE LA CLASE
∴ no disponen del puntero this

```
class clase{
    int uno, dos;
    static int tres;
 public:
    int suma(){ return uno + dos + tres;       }

    static int calcular2(clase obj){
          int aux;
          aux = obj.uno + obj.dos + tres; //OK
          aux = obj.suma() + tres; //OK
          return aux;
    }
};
```

11

Objetos y funciones miembros const

const → Palabra reservada que indica que un O no es modificable y que cualquier intento de modificarlo sería un error

```
const Hora mediodia(12,0,0);
```

Un O const no puede ser modificado por asignación, por lo que deberá ser inicializado mediante un inicializador de miembro que proporciona al constructor valores iniciales correspondientes al O

```
class nuevaClase {
  private:
     int count;
     const int incremento; //dato miembro const
  public:
     nuevaClase(int c, int i): incremento(i)
     {
      //incremento = i;//Error no esta permitida la asignación a una cte
      count = c;
     }
};
```

← Inicializador de miembro

12

Objetos y funciones miembros const

`const`

Una función miembro se define como **const** si tanto en su declaración como en su definición se inserta la palabra reservada const después de la lista de parámetros de la función:

```
int getValue( ) const; //Declaración dentro de la clase
int Clase::getValue( ) const { . . . } //Definición
```

Solamente las funciones const pueden operar sobre O const, aunque, obviamente, no pueden modificarlos

Para constructores y destructores de O const no se requiere la palabra const

13