

# Memoria Dinámica

Ing. Armandina J. Leal Flores

# Memoria Estática

- ▶ Es la memoria que se asigna a una variable. Esta memoria le pertenece a la variable hasta que se termina la ejecución del programa.
- ▶ Ejemplo: La memoria que se le asigna a las variables globales y las variables locales declaradas como estáticas (*variables estáticas*).

# Variable automática

- ▶ Es una variable local (que no fue declarada como estática) a la que se le asigna memoria (**se crea**) cuando se llega a su declaración y se le quita la memoria (**se destruye**) cuando se termina la ejecución del bloque en el cual se declaró.

# Variable dinámica

- ▶ Variable a la que se le asigna memoria durante la ejecución del programa.
- ▶ En el programa se especifica el momento en el que se debe crear y/o destruir según sea necesario.
- ▶ Se emplean los operadores: **new** y **delete**.

# New y Delete

- ▶ Cuando el programa necesita una variable adicional debe realizar un **new**.
- ▶ Cuando el programa ya no necesita la variable que solicitó con el new se debe realizar un **delete**.

# New

- ▶ Permite la creación de una variable.
- ▶ Sintaxis:

```
new tipoDeDato;
```

*//variable simple*

```
new tipoDeDato [expresión];
```

*//variable arreglo*

# New

- ▶ Ejemplo:

```
int *ipApunt;  
ipApunt = new int;
```

Crea una variable de tipo **int** y almacena su dirección en la variable **ipApunt**.

- ▶ Ejemplo:

```
char *cpLetrero;  
cpLetrero = new char[10];
```

Crea una variable de tipo arreglo y almacena la dirección base en la variable **cpLetrero**.

# New

- ▶ **Cuidado....** Si al realizar la operación new, no hay espacio de memoria disponible, se asigna al apuntador el valor de NULL.

```
int *ipApunt;  
ipApunt = new int;  
if (ipApunt == NULL)  
    cout << "No hay espacio de memoria";
```

# Heap

- ▶ Área de memoria reservada que se emplea para atender las solicitudes realizadas con los operadores new y delete.

# Delete

- ▶ Permite la destrucción de una variable dinámica (libera la memoria en la que se encontraba) en cualquier tiempo de la ejecución del programa.
- ▶ Sintaxis:  
`delete ipApuntador;` *//destruir una variable simple*  
`delete [ ] ipApuntador;` *//destruir un arreglo*

# Delete

---

Ejemplo:

---

```
int *ipApunt;
```

```
ipApunt = new int;
```

```
*ipApunt = 10;
```

```
delete ipApunt;
```

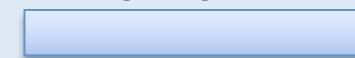
---

*ipApunt*

*ipApunt*

*ipApunt*

*ipApunt*



# Delete

- ▶ Si se aplica el operador **delete** a un apuntador que contiene NULL no sucede nada.
- ▶ Un **delete** se debe aplicar solamente a apuntadores que contienen una dirección asignada a través de un new.

# Memory leak (filtración de memoria)

- ▶ Pérdida de espacio de memoria disponible que ocurre cuando los espacios dinámicos que fueron asignados nunca son liberados.

# ApuntaDores a Objetos

# Objetos Dinámicos

```
#include <iostream>
using namespace std;
class Fraccion
{
public:
    Fraccion ()
    { iNumerador = 0; iDenominador = 1; }
    Fraccion ( int iNumerador, int iDenominador)
    {   this -> iNumerador = iNumerador;
        this -> iDenominador = iDenominador;
    }
    void despliega()
    { cout << iNumerador << "/" << iDenominador << endl; }
private:
    int iNumerador, iDenominador;
};
```

## Constructores

```
#include <iostream>
#include "Fraccion.h"
int main()
{
    Fraccion *frapF, *frapK;
    frapF = new Fraccion();
    frapK = new Fraccion(4,5);
    (*frapF).despliega();
    frapK -> despliega();
    delete frapF;
    delete frapK;
    return 0;
}
```

# Objetos con atributos dinámicos

```
#include <iostream>
using namespace std;
class Ejemplo
{
public:
    Ejemplo () { ipX = new int; *ipX = 0; }
    Ejemplo (int iValor)
    { ipX = new int; *ipX = iValor; }
    ~Ejemplo () { delete ipX; }

    void despliega() { cout << *ipX << endl; }

private:
    int *ipX;
};
```

**Constructores**

**Destructor**

```
#include <iostream>
#include "Ejemplo.h"

int main()
{
    Ejemplo ejeE1;
    Ejemplo ejeE2(5);

    ejeE1.despliega();
    ejeE2.despliega();

    return 0;
}
```

# Decálogo para programar con Memoria Dinámica

# Decálogo

1. Por cada vez que se haya ejecutado un NEW, deberá ejecutarse un DELETE antes de terminar la ejecución del programa.
2. Un DELETE actúa liberando el espacio de memoria apuntado, independientemente de que existan más apuntadores al mismo espacio.
3. Un apuntador local a un módulo, se destruye al terminar la ejecución del mismo, sin importar a qué espacio de memoria referenciaba.
4. “BASURA” es diferente a “NADA”.
5. Hacer un DELETE, con un apuntador que no hace referencia a un espacio de memoria dinámica, provocará fallas en ejecución.

# Decálogo

6. Al asignar un valor a un apuntador con la operación NEW, el apuntador perderá su valor anterior, independientemente de a qué esté apuntando.
7. Para utilizar un apuntador, no siempre se tiene que realizar un NEW.
8. Los valores de los apuntadores se pueden comparar para verificar si apuntan a donde mismo.
9. A un dato referenciado por un apuntador se le pueden aplicar todas las operaciones válidas para el tipo de dato.
10. Los arreglos en C++ realmente son apuntadores.