



# Programación I

## Unidad 03

### Punteros y memoria dinámica



# Unidad 3 – Memoria Dinámica

## Temario

- Asignación de memoria dinámica
- New - Delete
- NULL



# Memoria de la computadora

Existen 3 tipos de memoria:

## **Memoria estática**

Los objetos son creados al comenzar el programa y destruidos sólo al finalizar el mismo. Mantienen la misma localización en memoria durante todo el transcurso del programa. (Ej. Variables globales)

## **Memoria automática**

Los objetos son creados al entrar en el bloque en que están declarados, y se destruyen al salir del bloque. Se trata de un proceso dinámico pero manejado de modo automático por el compilador. (Ejemplo. Funciones)

## **Memoria dinámica**

En este caso tanto la creación como destrucción de los objetos está en manos del programador.



# Memoria dinámica

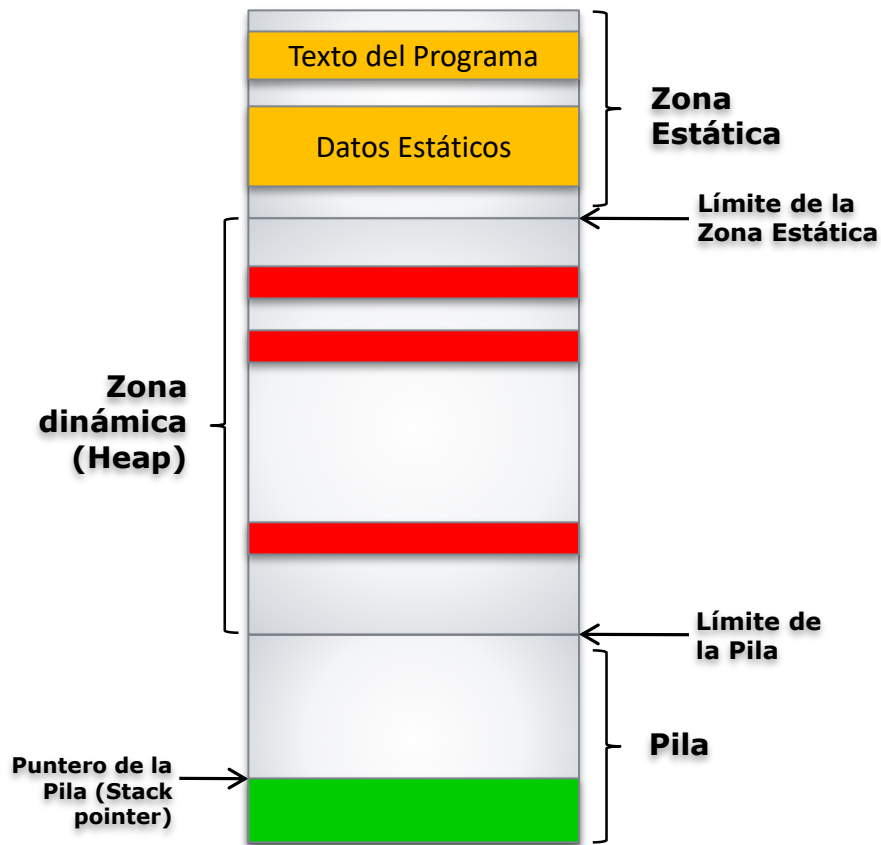
“La memoria dinámica es un espacio de almacenamiento que se solicita *en tiempo de ejecución*. De esa manera, a medida que el proceso va necesitando espacio para más líneas, va solicitando más memoria al sistema operativo para guardarlas” <sup>(1)</sup>.

El medio para manejar la memoria que otorga el sistema operativo, es el **puntero**.

(1) [http://laurel.datsi.fi.upm.es/~rpons/personal/trabajos/curso\\_c/node113.html](http://laurel.datsi.fi.upm.es/~rpons/personal/trabajos/curso_c/node113.html)



# Memoria dinámica



Cuando el sistema operativo carga un programa para ejecutarlo y lo convierte en proceso, le asigna cuatro partes lógicas en memoria principal: texto, datos (estáticos), pila y una zona libre (o *heap*) .

Esta zona libre es la que va a contener los datos dinámicos, la cual, a su vez, en cada instante de la ejecución tendrá partes asignadas a los mismos y partes libres que *fragmentarán* esta zona, siendo posible que se agote si no se liberan las partes utilizadas ya inservibles.

La pila también varía su tamaño dinámicamente, pero la gestiona el sistema operativo, no el programador.



# En resumen...

## Memoria Estática

Es asignada por el compilador al inicio del programa y liberado al término del mismo.

Ejemplos:

```
int x;  
char d;
```

## Memoria Dinámica

Debe ser asignada y liberada por el programador luego de haber cumplido su función.



# ¿Por qué usar punteros?

Para poder emplear variables dinámicas es necesario emplear un tipo de dato que permita referenciar nuevas posiciones de memoria que no han sido declaradas a priori y que se van a crear y destruir en tiempo de ejecución.



# Asignación de memoria dinámica





# Asignación de memoria dinámica

Para manejar la memoria dinámicamente podemos utilizar los siguientes comandos:

- **New – Delete**
- **Malloc - Free**

Definidos, ambos, en la librería `<stdio.h>`



# New y Delete

Solo en C++

stdio.h



# New

**New** permite reservar memoria dinámicamente y asignarla a una variable de tipo puntero.

## Sintaxis

```
TipoDato *nombrePuntero;  
nombrePuntero = new TipoDato;
```

## Ejemplo 1

```
int *variableA;  
variableA = new int;
```

## Ejemplo 2

```
int *variableA = new int;
```



# Delete

`delete` permite liberar la memoria reservada con `new`.

**SIEMPRE** que exista un `new` debe existir un `delete` correspondiente.

Una vez liberada la memoria ya no se podrá acceder a ella por lo que es necesario que se libere cuando ya no será utilizada.

## Sintaxis

```
delete nombrePuntero;
```

## Ejemplo

```
delete variableA;
```



# Cómo funciona New – (1/7)

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
Pila	...	
	...	
	...	
	...	
	...	
	...	

- Supongamos que la memoria se encuentra como en el dibujo.
- Analizaremos únicamente la **Zona estática y el Heap...**
- El color blanco representa espacios de memoria libre y las de color representan espacios de memoria ocupados.



# Cómo funciona New – (2/7)

Y queremos realizar lo siguiente:

1. Definir un puntero a float llamado ptrFloat.

```
float * ptrFloat;
```

2. Asignar dinámicamente el espacio de memoria donde se almacenará el dato float que apuntará el puntero.

```
ptrFloat = new float;
```

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	



# Cómo funciona New – (3/7)

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	

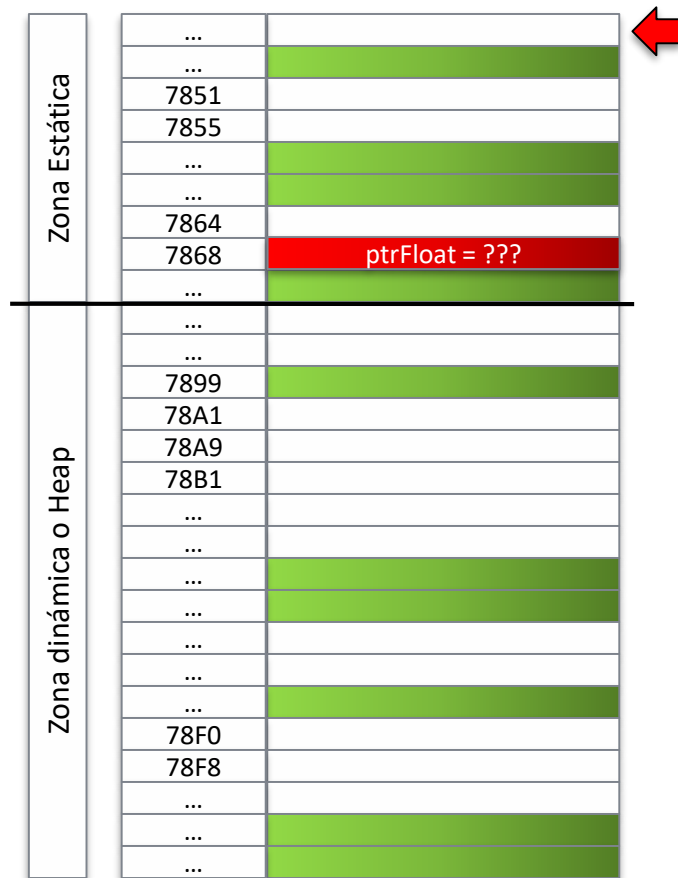
1. Definir un puntero a float llamado ptrFloat.

```
float * ptrFloat;
```

En este caso la variable **ptrFloat** es una **variable estática**, por lo tanto ocupará una posición de memoria dentro de la **zona estática**.



# Cómo funciona New – (4/7)



```
float *ptrFloat;
```

Mediante un algoritmo de selección se busca un espacio en memoria para alojar al puntero.

Imaginemos que el espacio de **color rojo** es el asignado para alojar a la variable **ptrfloat**.





# Cómo funciona New – (5/7)

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	ptrFloat = ???
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	

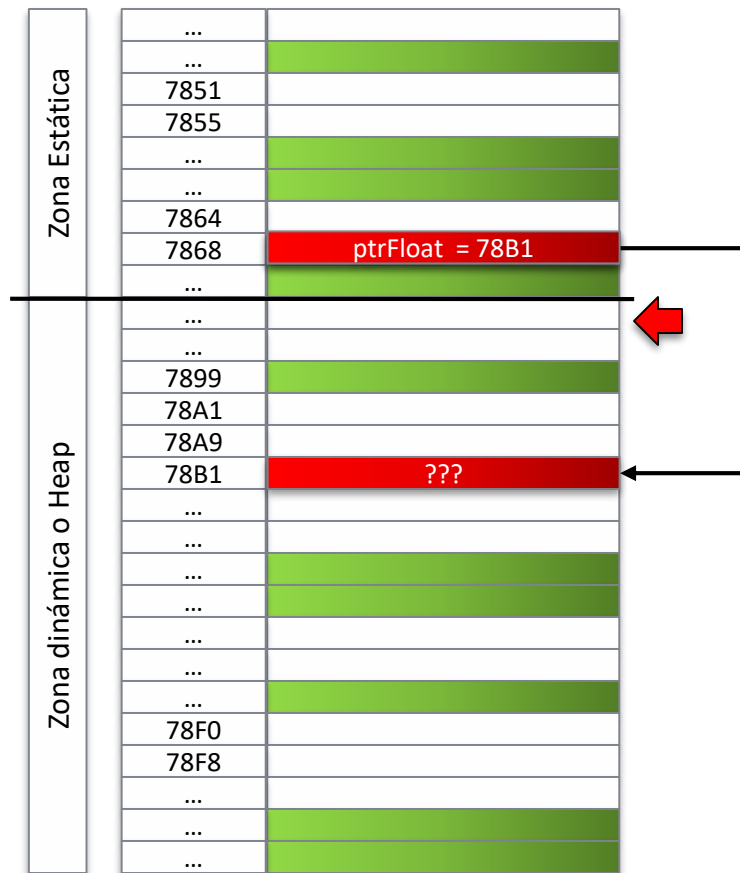
2. Asignar dinámicamente el espacio de memoria donde se almacenará el dato float que apuntará el puntero.

**ptrFloat = new float;**

En este caso al ser una asignación **dinámica** se le asignará un espacio dentro del **Heap**.



# Cómo funciona New – (6/7)



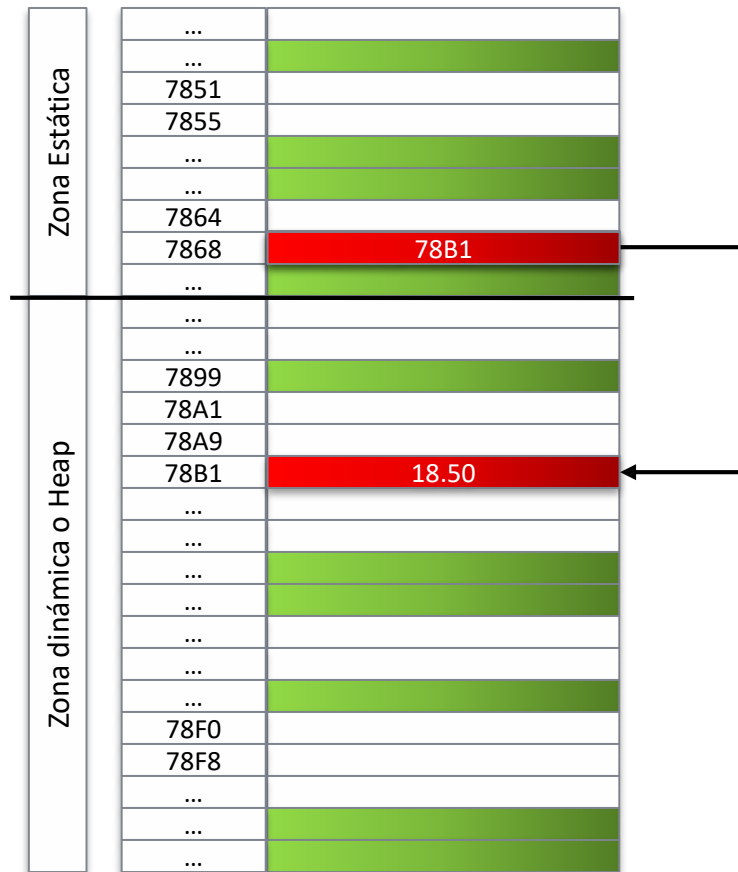
Mediante un algoritmo de selección se busca un espacio de memoria suficiente para alojar un dato según el tipo al que apunta el puntero.

En este caso se ha definido que el espacio de memoria **78B1** es el adecuado para guardar el dato.

El valor **78B1** se almacena en la variable **ptrFloat** conectando así al puntero con el espacio en memoria.



# Cómo funciona New – (7/7)

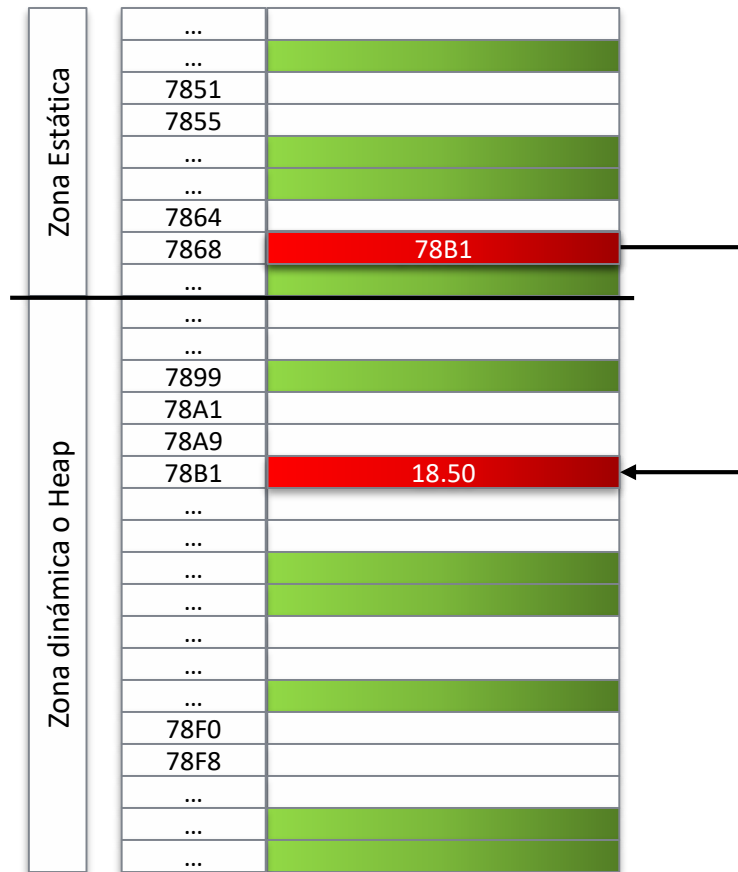


Por último, si deseamos asignarle un valor al dato apuntado tendríamos que hacerlo de la siguiente forma:

`*ptrFloat = 18.50`



# Cómo funciona Delete – (1/2)



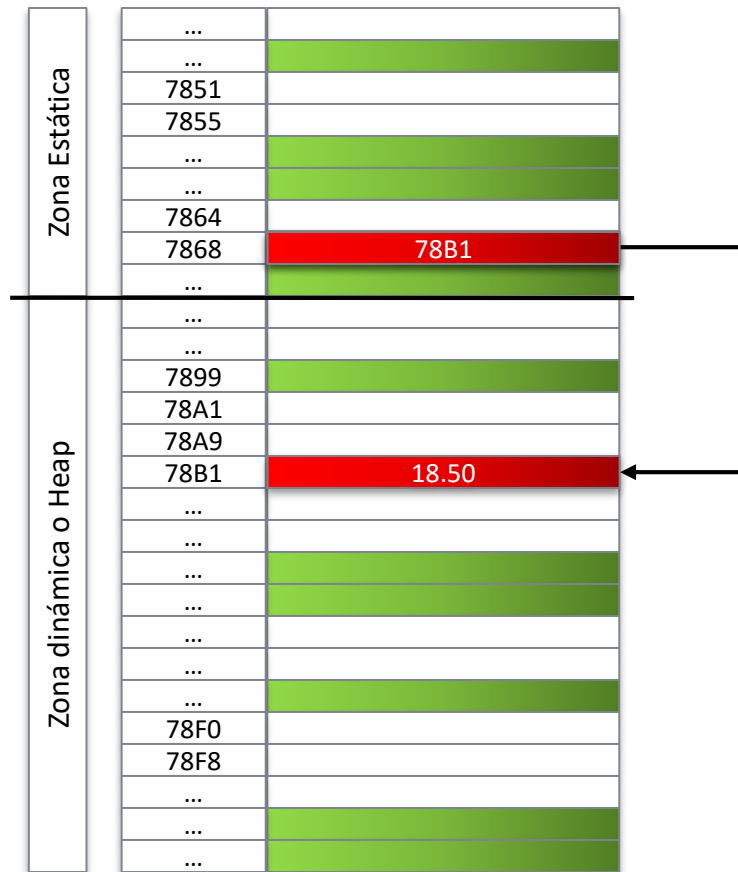
Del ejemplo anterior se tenía lo siguiente:

Mediante el comando New se había asignado dinámicamente el espacio de memoria al puntero llamado **ptrFloat**.

Cuando el puntero ya no sea usado durante la ejecución del programa, tendremos que liberar la memoria que fue asignada.



# Cómo funciona Delete – (2/2)



Para liberar la memoria reservada utilizaremos el siguiente comando:

```
delete ptrFloat;
```

Este comando libera la memoria reservada, y rompe el enlace entre el puntero y la memoria que guarda el dato almacenado.

Se dice que el puntero deja de “apuntar” a la dirección de memoria.



# Ejemplos

New y Delete



## Ejemplo 1

Asignarle memoria dinámica a un puntero a un dato de tipo int, luego asignarle el valor de 5 e imprimirlo por pantalla.

```
int main()
{
    int *A;
    A = new int;
    *A = 5;
    cout<<"El dato es " << *A << "\n";

    delete A;
    return 0;
}
```



## Ejemplo 2

Asignarle memoria dinámica a dos punteros float, asignarle un valor, sumarlos y presentar el resultado por pantalla.

```
int main()
{
    float *B, *C;
    B = new float;
    C = new float;
    *B = 16.4;
    *C = 3.2;
    cout<<"La suma de "<< *B <<" y "<< *C <<" es "<< *B + *C <<"\n";

    delete B;
    delete C;
    return 0;
}
```





## Ejemplo 3

Realice un programa que permita leer dos números enteros desde el teclado, e imprima la suma, diferencia y el producto de ellos.

Resuelva el ejercicio declarando dos punteros a enteros.

```
int main()
{
    int *A = new int;
    int *B = new int;

    cout<<"Ingrese el primer numero: ";
    cin>> *A;

    cout<<"Ingrese el segundo numero: ";
    cin>> *B;

    cout<<"\n\nRESULTADOS\n-----\n";

    cout<<"Suma: " << *A + *B << "\n";
    cout<<"Diferencia: " << *A - *B << "\n";
    cout<<"Producto: " << *A * *B << "\n";

    delete A;
    delete B;
    return 0;
}
```



# NULL

- Definición
- La importancia de NULL al utilizar New
- La importancia de NULL al momento de declarar
- La importancia de NULL al utilizar Delete



# NULL - Definición

Es una constante que se aplica, especialmente, a los punteros para indicar que estos no apuntan a ningún valor.

## Sintaxis

```
TipoDato *nombrePuntero  
nombrePuntero = NULL
```

## Ejemplo 1

```
int *pA;  
pA = NULL;
```

## Ejemplo 2

```
int *pA = NULL;
```

**NOTA:** No olvide de asignarle un espacio de memoria al puntero antes de utilizarlo.



# Importancia de NULL

Al momento de utilizar New



# La importancia de NULL al utilizar New

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
Pila	...	
	...	
	...	
	...	
	...	
	...	

- Imaginemos que la memoria se encuentra como en el dibujo.
- Analizaremos únicamente la **Zona estática y el Heap...**
- El color blanco representa espacios de memoria libre y las de color representan espacios de memoria ocupados.



# La importancia de NULL al utilizar New

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	

Y queremos realizar lo siguiente:

1. Definir un puntero a float llamado ptrFloat.

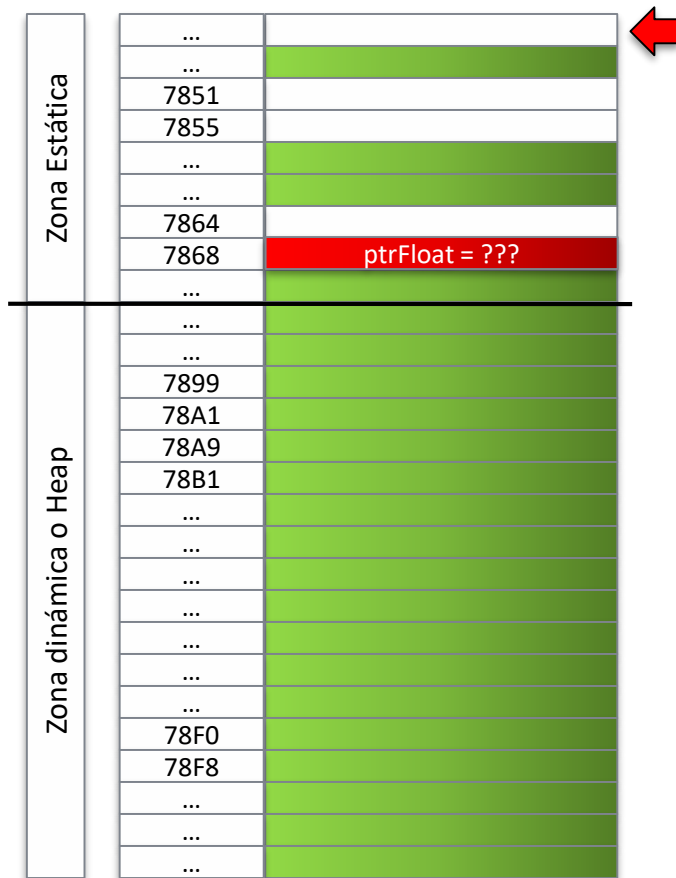
```
float * ptrFloat;
```

2. Asignar dinámicamente el espacio de memoria donde se almacenará el dato float que apuntará el puntero.

```
ptrFloat = new float;
```



# La importancia de NULL al utilizar New o Malloc



```
float *ptrFloat;
```

Mediante un algoritmo de selección se busca un espacio en memoria para alojar al puntero.

Imaginemos que el espacio de **color rojo** es el asignado para alojar a la variable **ptrfloat**.



# La importancia de NULL al utilizar New

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	ptrFloat = ???
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	

2. Asignar dinámicamente el espacio de memoria donde se almacenará el dato float que apuntará el puntero.

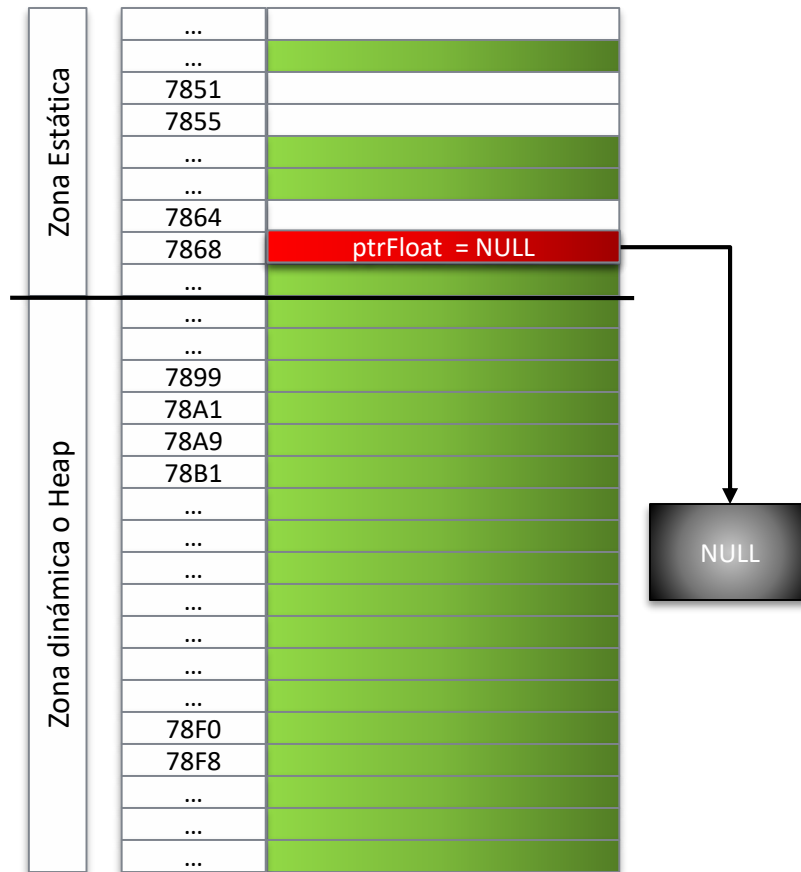
**ptrFloat = new float;**

En este caso al ser una asignación **dinámica** se le asignará un espacio dentro del **Heap**.





# La importancia de NULL al utilizar New



Mediante un algoritmo de selección se busca un espacio de memoria suficiente para alojar un dato según el tipo al que apunta el puntero.

En este caso no se ha podido encontrar espacio suficiente en la memoria dinámica por lo que se le asigna el valor de **NULL** a la variable **ptrFloat**.

Esto indica que la variable **ptrFloat** no apunta a ningún lugar por lo que se considera que no tiene espacio reservado.

**NOTA:** El funcionamiento de **NULL** es igual para **New** que para **Malloc**



# Importancia de NULL

Al momento de declarar punteros



# La importancia de NULL al momento de declarar

- Cuando declaramos variables, es recomendable inicializarlas para eliminar posibles errores.
- Por ejemplo cuando declaramos `int A;` es posible que el valor de `A` sea 256 o cualquier otra cosa ya que no necesariamente comienza con un valor de 0.
- Para evitar estos errores por lo general después de declarada la variable la inicializamos con un valor como por ejemplo `A = 10;`



# La importancia de NULL al momento de declarar

- Lo mismo sucede con los punteros.
- Al declarar `int *pA;` el valor de `pA` puede ser cualquier cosa, inclusive puede ser una dirección de memoria que no nos corresponde por lo que si intentamos acceder a dicha memoria obtendremos un error.
- Es por esta razón que es importante **inicializar** el valor del puntero y por esa razón colocamos inmediatamente el valor que nos otorga **New** o **Malloc** que puede ser una dirección de memoria o NULL.
  - `pA = new int;`
  - `pA = (int*) malloc(sizeof(int));`



# La importancia de NULL al momento de declarar

- Si por alguna razón no se reservará la memoria para el puntero de forma inmediata o nunca (si es un puntero para manipular variables) deberíamos colocarle inmediatamente la dirección de memoria a donde queremos que apunte.
- Si no deseamos que apunte a ningún lado por el momento debemos colocarle el valor de NULL.

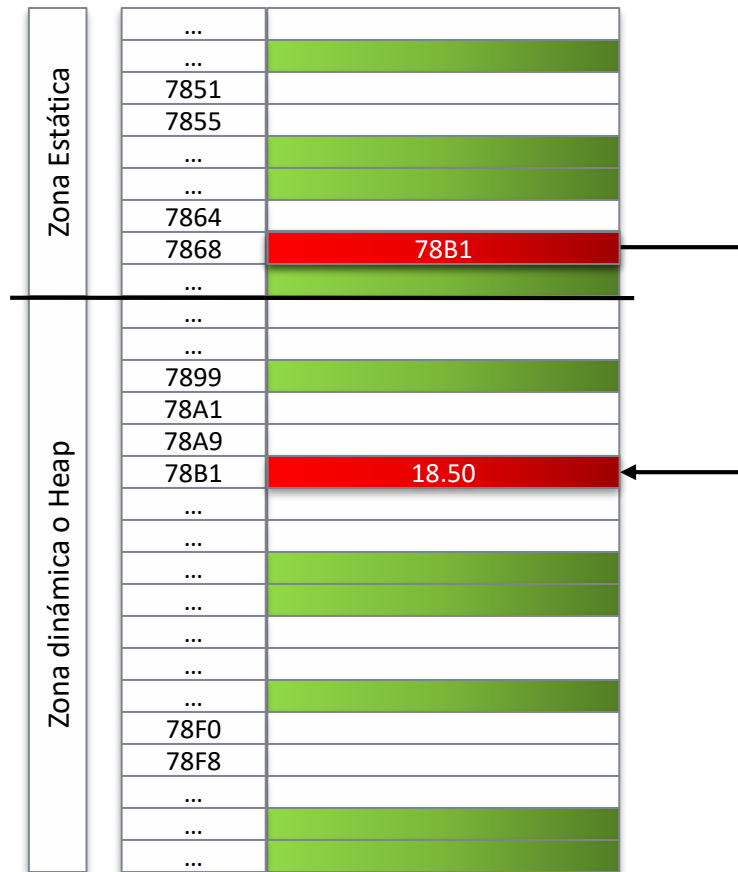


# Importancia de NULL

Al momento de liberar la memoria usando delete



# La importancia de NULL al utilizar delete



Del ejemplo anterior se tenía lo siguiente:

Mediante el comando New se había asignado dinámicamente el espacio de memoria al puntero llamado **ptrFloat**.

Cuando el puntero ya no sea usado durante la ejecución del programa, liberábamos la memoria utilizando del comando delete.



# La importancia de NULL al utilizar delete

Zona Estática	...	
	...	
	7851	
	7855	
	...	
	...	
	7864	
	7868	78B1
	...	
	...	
Zona dinámica o Heap	...	
	...	
	7899	
	78A1	
	78A9	
	78B1	
	...	
	...	
	...	
	...	
	...	
	...	
	...	
	78F0	
	78F8	
	...	
	...	
	...	

- El problema surge que ahora la variable ptrFloat ya no apunta al espacio de memoria y tampoco existe memoria reservada para esta variable.
- Sin embargo la variable ptrFloat aún contiene la dirección de memoria 78B1 por lo que si alguien realiza la siguiente operación:

```
if (ptrFloat != NULL)
    printf("ptrFloat tiene datos");
```
- Se imprimirá que aún tiene datos para dicha variable cuando esto ya no es cierto.
- Es por esta razón que es recomendable colocar NULL al puntero después de un delete o free.





# Ejemplos

NULL



## Ejemplo 1

Declarar una variable entera de forma dinámica y en caso de que se pueda reservar el espacio necesario, solicitar un numero y mostrar el valor incrementado en 5. En caso de no poder reservar el espacio necesario mostrar un mensaje de error.

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

int main()
{
    int *A;
    A = new int;
    if (A == NULL)
        cout<<"No se pudo asignar la memoria...";
    else
    {
        cout<<"Ingrese un valor entero: ";
        cin>>*A;
        *A = *A + 5;
        cout<<"El valor entero incrementado en 5 es :"<< *A;
        delete A;
    }
    return 0;
}
```



## Ejemplo 2

Demostrar la importancia de NULL.

- Mostrar que al declarar un puntero apunta a una dirección cualquiera.
- Mostrar que al utilizar New o Malloc el puntero apunta a otra dirección de memoria.
- Mostrar que al liberar la memoria mediante delete o free, el puntero sigue manteniendo la misma dirección de memoria.

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

int main()
{
    int *ptr;
    cout<<"ptr apunta a la direccion: "<< ptr <<"\n";

    ptr = new int;
    cout<<"ptr despues de new apunta a la direccion: "<< ptr <<"\n";

    delete ptr;
    cout<<"ptr despues de delete apunta a la direccion: "<< ptr <<"\n";

    if (ptr == NULL)
        cout<<"Ya se libero la memoria";
    else
        cout<<"No se ha liberado nada";

    return 0;
}
```