PUCP

Programación 2

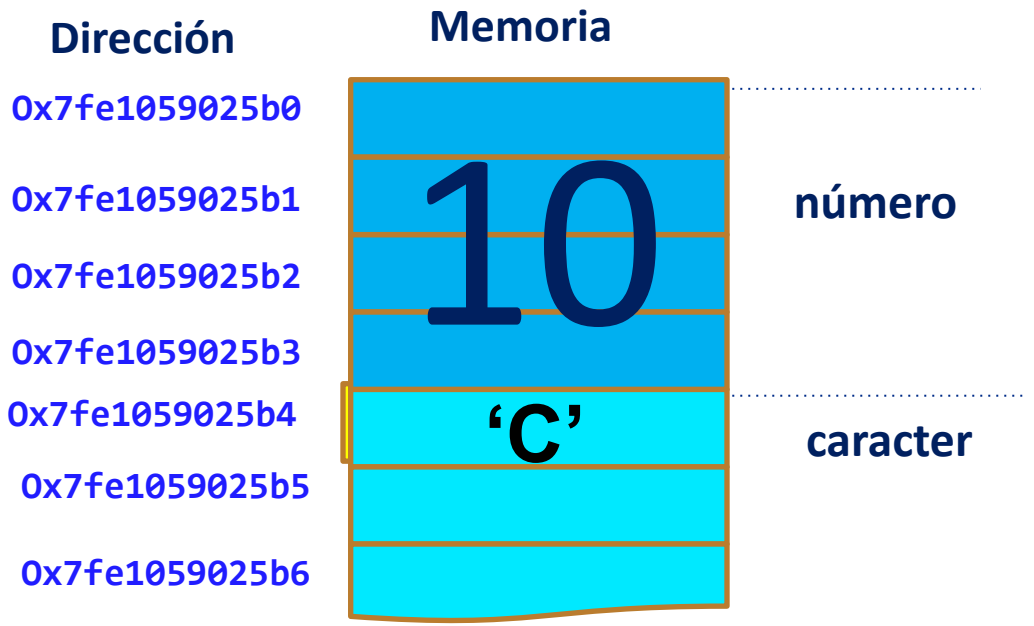
Dr. Heider Sanchez

Manejando memoria directamente

Representación de una variable

```
int numero = 10;
char caracter = 'C';
```

- Los **tipos de datos:** tamaño específico y conjunto de reglas para cada tipo.
- Ejemplo: **`char`** es un byte
- El tipo **`int`**: 4 bytes en 32 bits, 8 bytes en 64 bits.



¿Cuál es la dirección de memoria de la variable número?

¿Qué es un puntero? y ¿Cómo se define?

- **Un puntero, es un tipo de componente que apunta a otro tipo.**
- **Se utilizan para acceder indirectamente a otros objetos.**
- **El puntero es un objeto que puede ser asignado y copiado.**

Se define así:

```
int  *p;
p = &ival;
```

```
int  ival= 42;
int  *p = &ival;  // p contiene la dirección de ival; se dice que p apunta a ival
int *q =nullptr;
q = &ival;
```

La segunda instrucción define **p** como un puntero a un **int** e inicializa a **p** apuntando a un objeto int llamado **ival**

¿Cómo se define?

El tipo del puntero y el objeto al cual apunta deben coincidir.

```
double dval;
double *pd = &dval;   //   pd se inicializa con la dirección de un double

int *pi = pd;  //  error  pi  y pd difieren en el tipo
pi = &dval;  //  error, se asigna la dirección de un double a un puntero a un entero.
```

El valor de un puntero puede:

1. Apuntar a un objeto ó
2. Puede ser ***nullptr***, que indica que el puntero no ha sido ligado a ninguna variable/objeto.

Usando un puntero para acceder a un objeto

Cuando un puntero apunta a un objeto, se puede utilizar el ”dereference operator” (the * operator) para acceder al objeto.

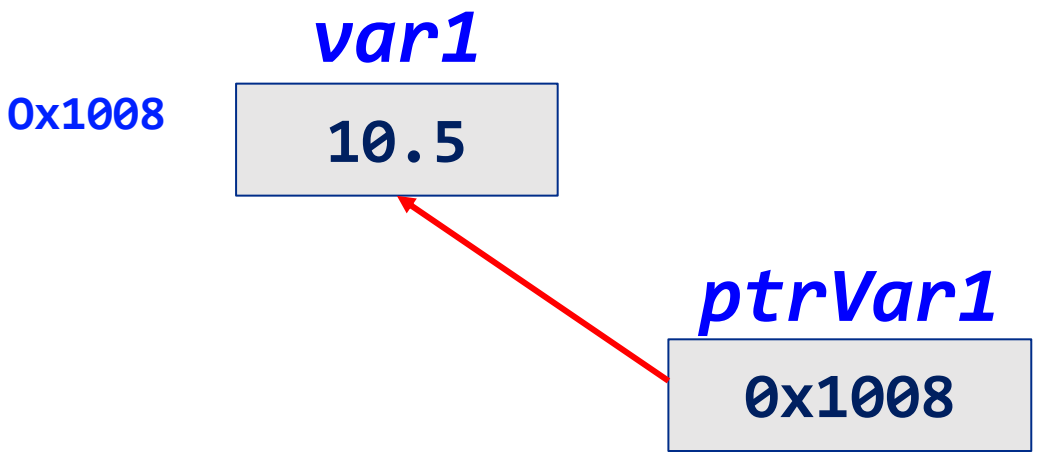
```
int ival = 42;
int *p = &ival;  //  p contiene la dirección de ival;  p es un puntero a ival
cout <<  p << *p << &ival << &p;
```

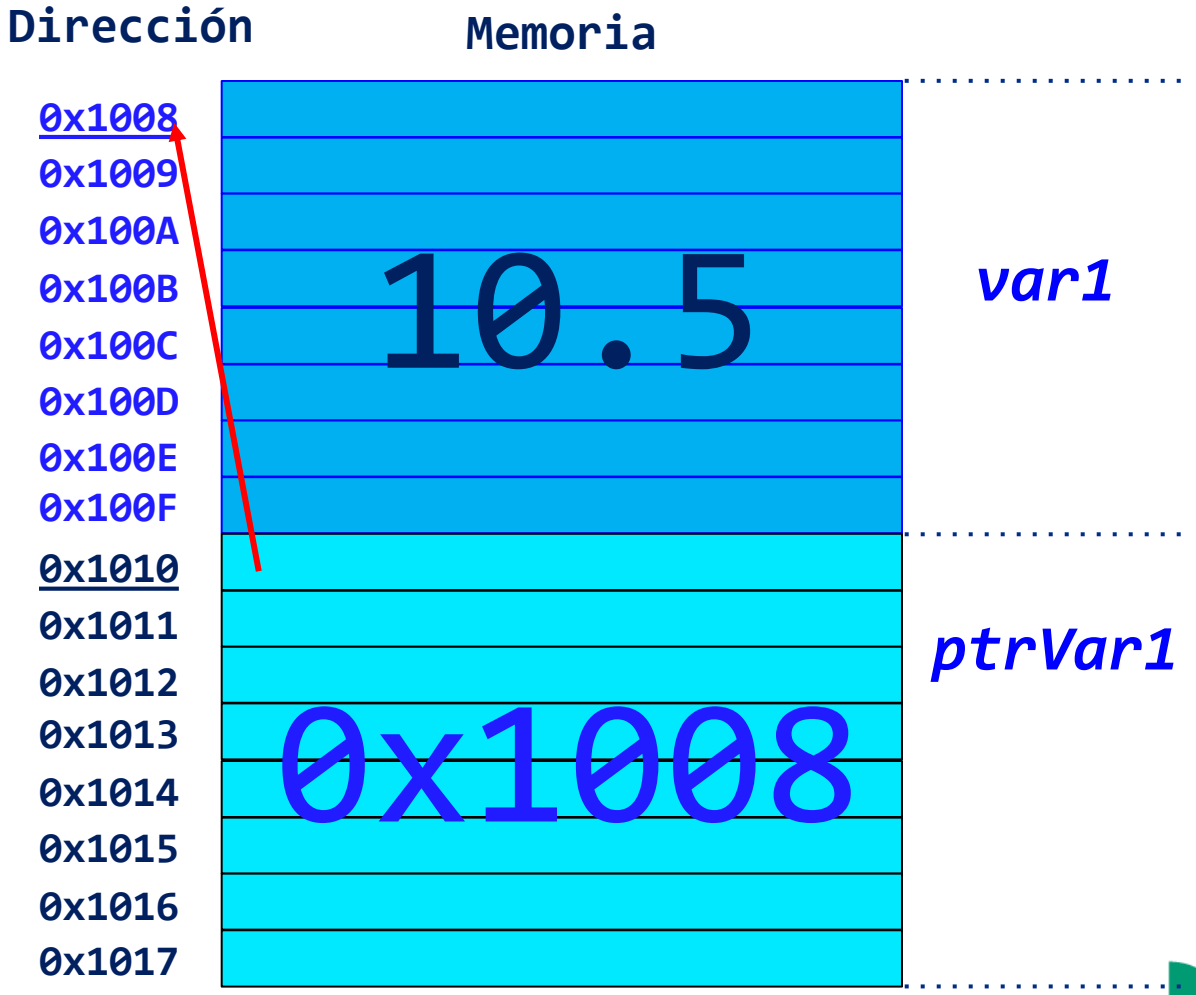
Desreferenciar un puntero se accede al objeto que es apuntado por el puntero.

```
*p = 0;     // * se accede al objeto, se asigna un nuevo valor a ival a través de p
  //  Cuando se asigna a *p, estamos asignando el valor 0  al objeto al cual p apunta

cout << *p;   // se imprime 0
```

Veamos lo que podría ocurrir en la memoria

```
double   var1 = 10.5;
double*  ptrVar1 = &var1;
```



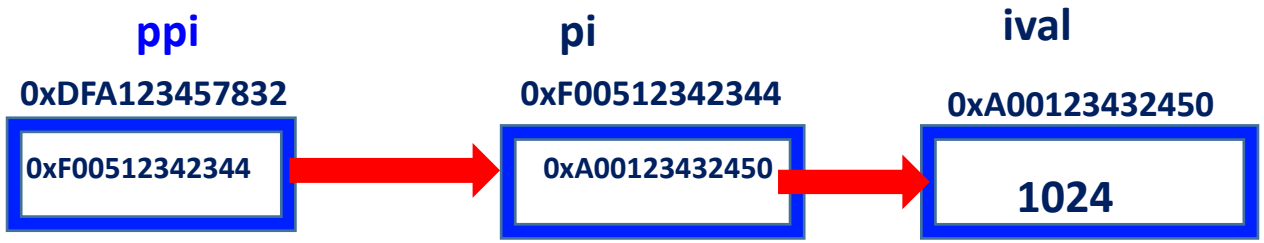


Un puntero en x64 utiliza 8 bytes.

Punteros a punteros:

Un puntero es un objeto en memoria, entonces como cualquier objeto tiene una dirección. Por lo tanto se puede asignar la dirección de un puntero en otro puntero.

```
int ival = 1024;
int *pi ;    //  pi es un puntero a un int
pi  = &ival;
int **ppi = &pi;   //  ppi es un puntero a un puntero de un int
```




```
cout << "Los valores de ival  \n";
cout << "Valor directo          :  " <<  ival  << "\n";
cout << "Valor indirecto        : " <<  *pi   << "\n";
cout << "Doble valor indirecto  : "<<   **ppi;
```

Referencia: Es un alias

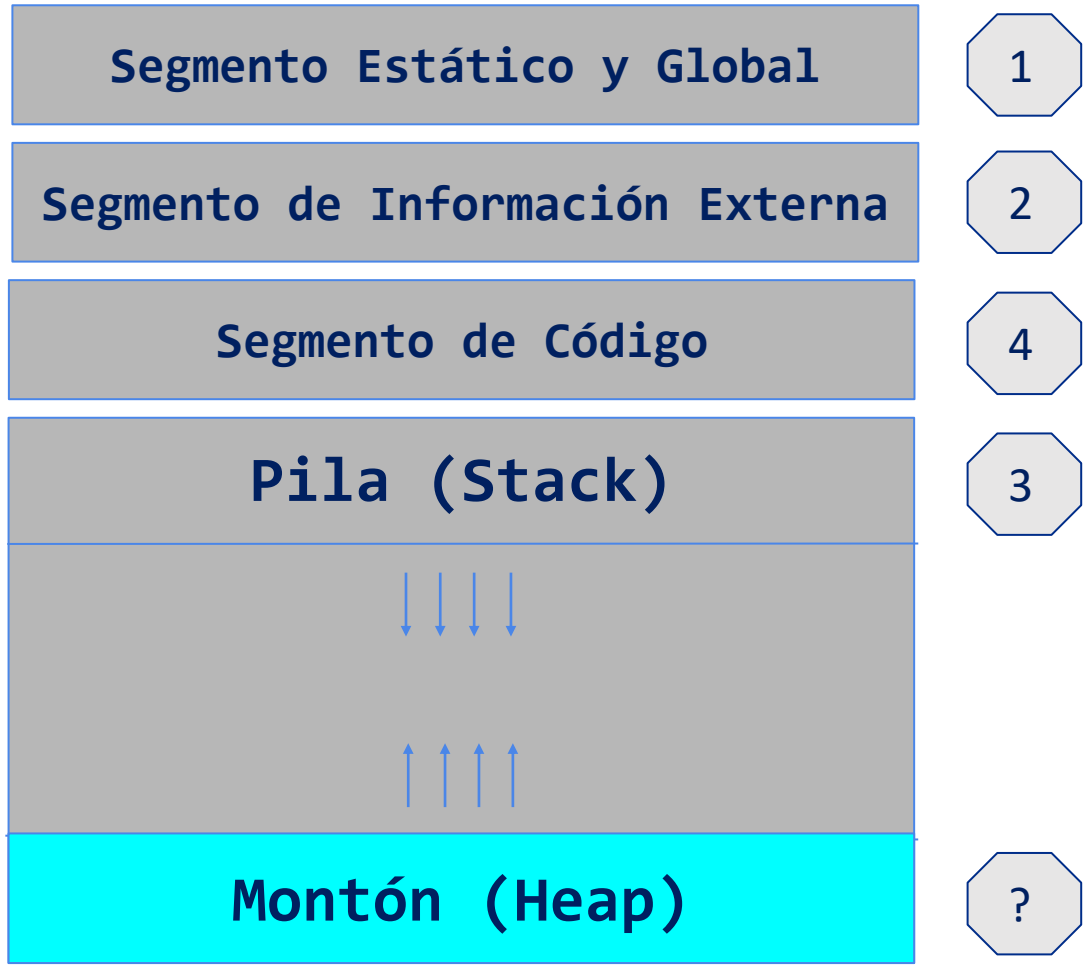
```
int ival= 1024;
int &refVal = ival;  // refVal refers to ival    (es otro nombre de ival)
refVal = 2;   // asigna 2 al objeto al que se refiere refVal, es decir ival = 2;
int &refVal2;    // error: una referencia debe ser inicializada
```

Referencia a Punteros:

Una referencia no es un objeto. Por lo tanto no puede haber un puntero a una referencia.
Sin embargo, como un puntero es un objeto, se puede definir una referencia a un puntero.

```
int i= 42;
int *p;          // p es un puntero a un int
int * &r=p;   //  r es una referencia al puntero p
r = &i;          //  r refiere al puntero, asignando &i a r hace que p apunte a i
*r = 0;          //  desreferenciado r da acceso al objeto i, que es el objeto apuntado
                 // por p y cambia i con el valor cero.
```

Programa de C++ en la memoria primaria



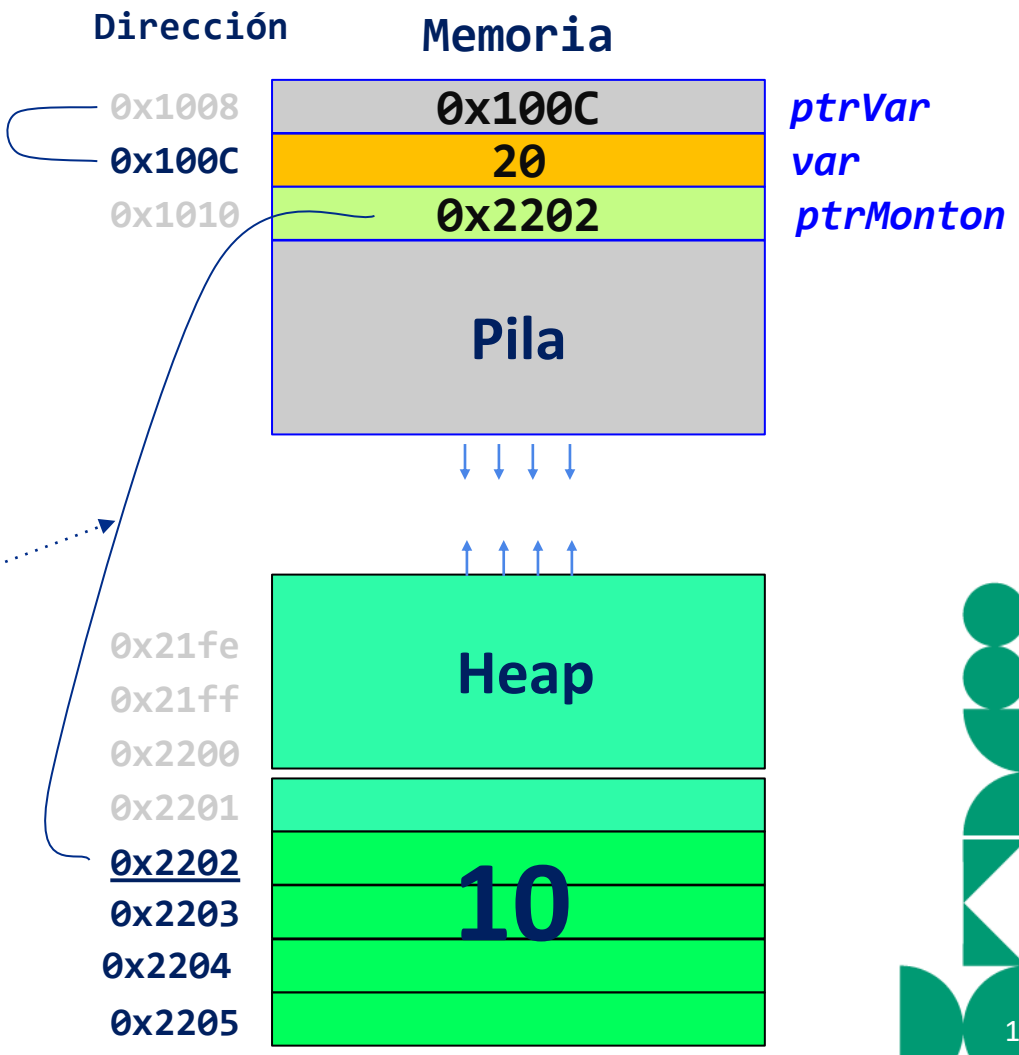

```
#include <iostream>
using namespace std;

int varGlobal = 20;                          1

int main(int argc, char * argv[])            2
{
    int varLocal = 10;                       3
int* ptrVarLocal = &varLocal;

    cout << varLocal << "\n";
    return 0;
}
```

4

Al Heap solo se puede acceder a través del uso de punteros.

Acceso al Heap

```
int* ptrMonton = nullptr;
int* ptrVar = nullptr;
....
int var = 20;
prtVar = &var;
....
ptrMonton = new int;
*ptrMonton = 10;
....
delete ptrMonton;
```



Paso por Valor

Memoria principal
del computador

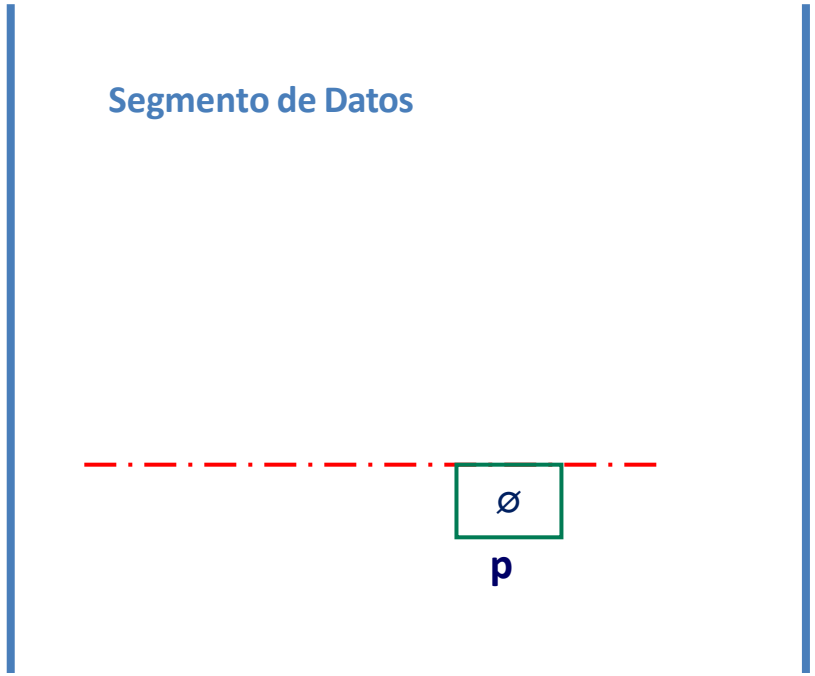
Segmento de Datos

```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

Memoria principal
del computador

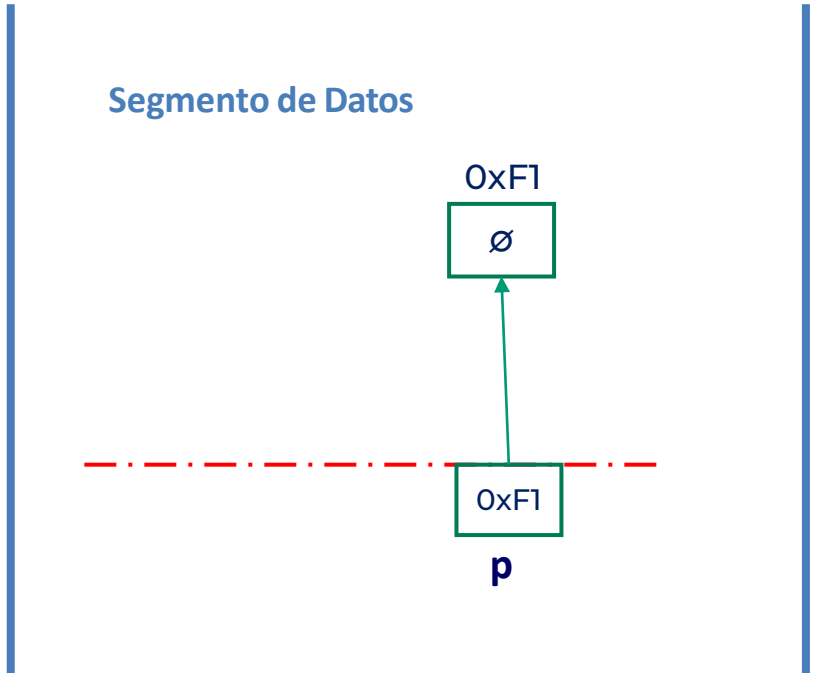



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

Memoria principal
del computador

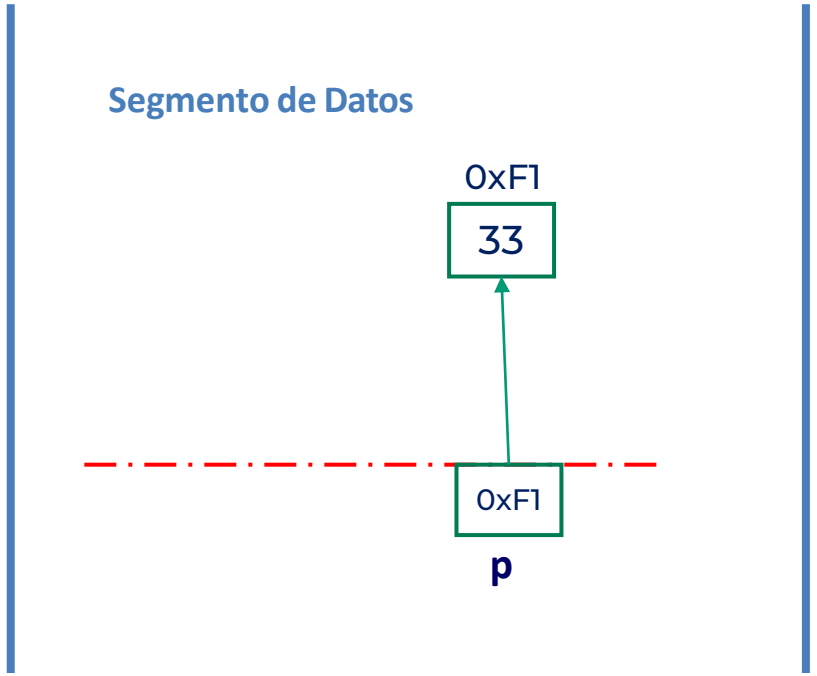



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

Memoria principal
del computador

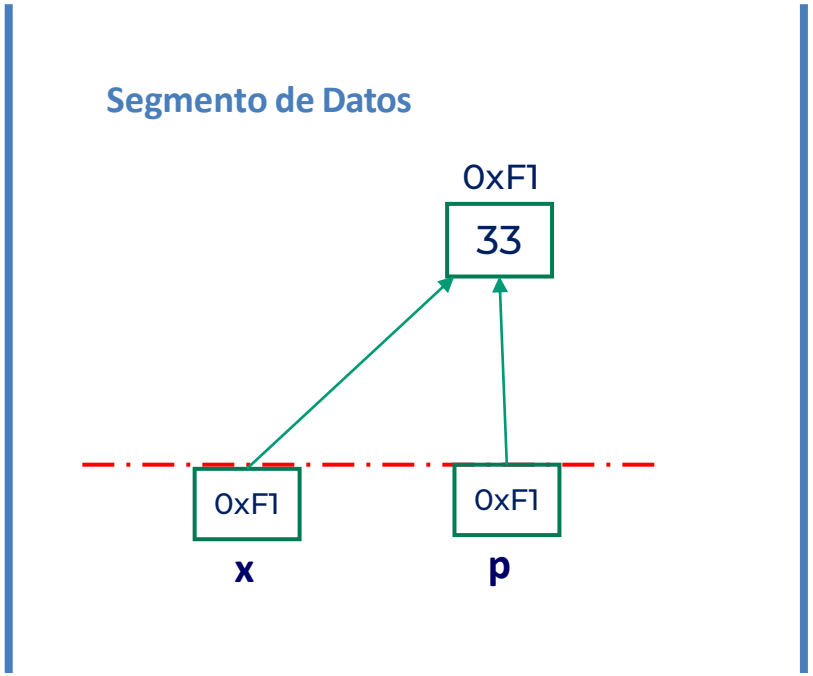



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

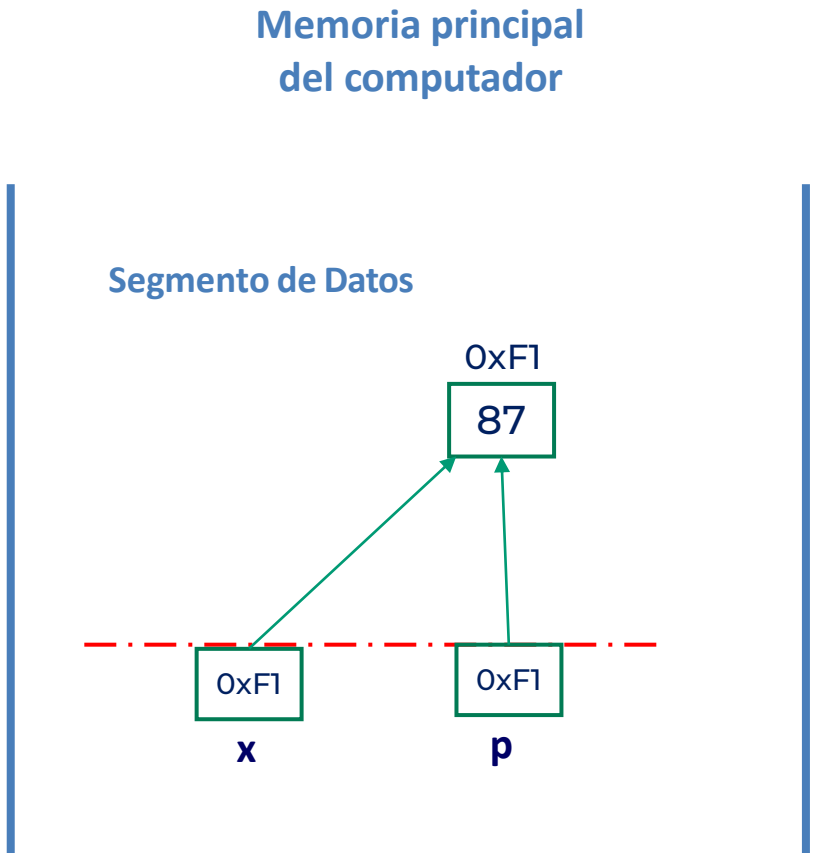
Memoria principal
del computador




```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

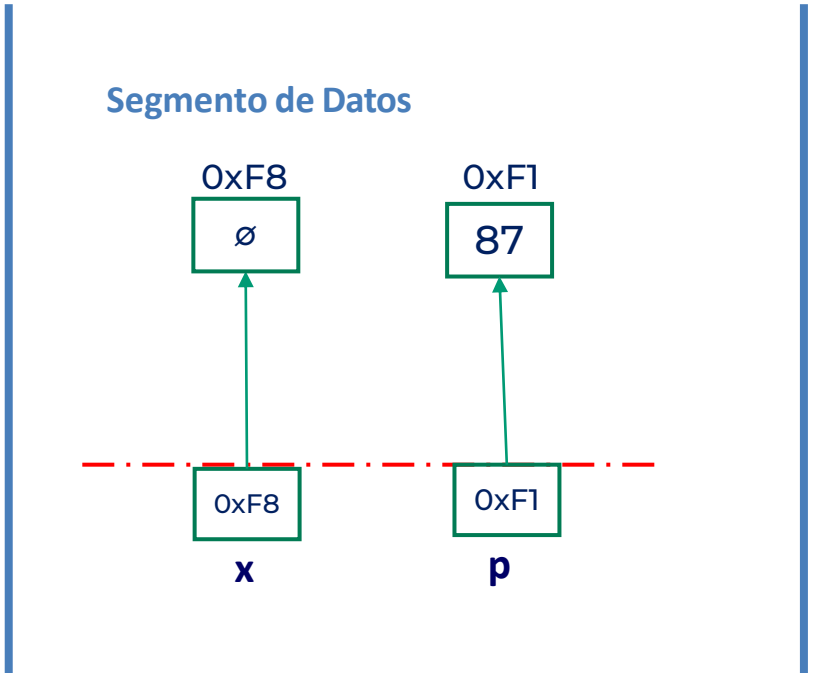



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

Memoria principal
del computador



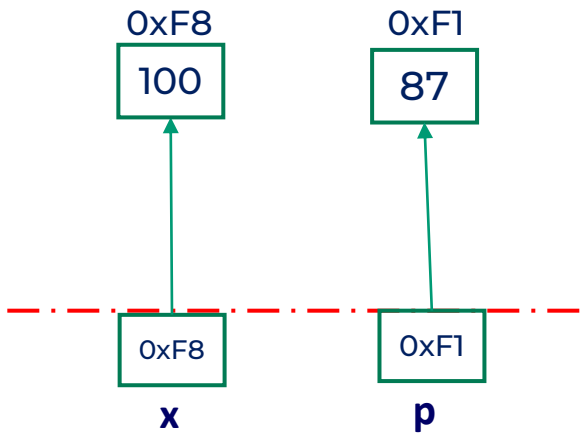

```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

Memoria principal
del computador

Segmento de Datos

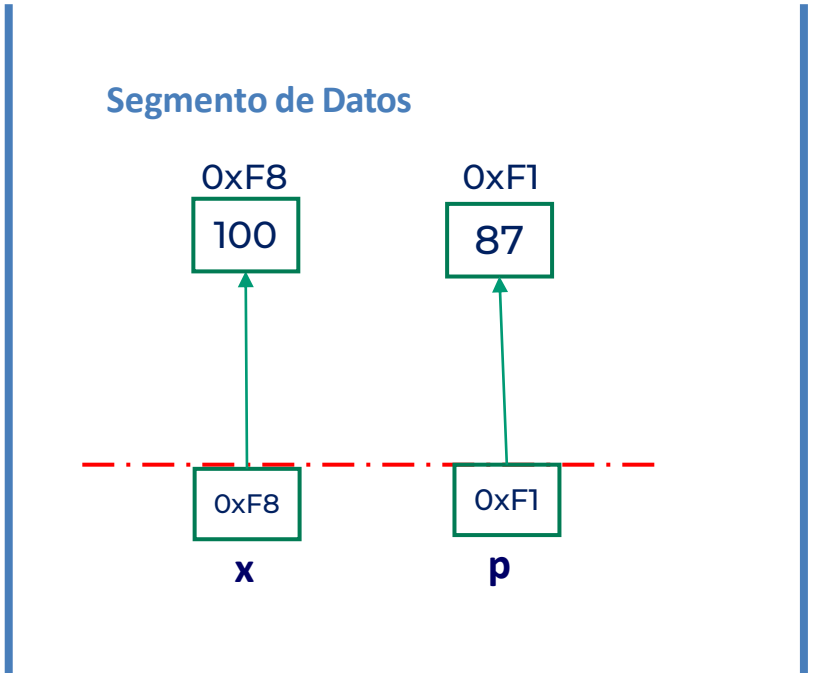



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Valor

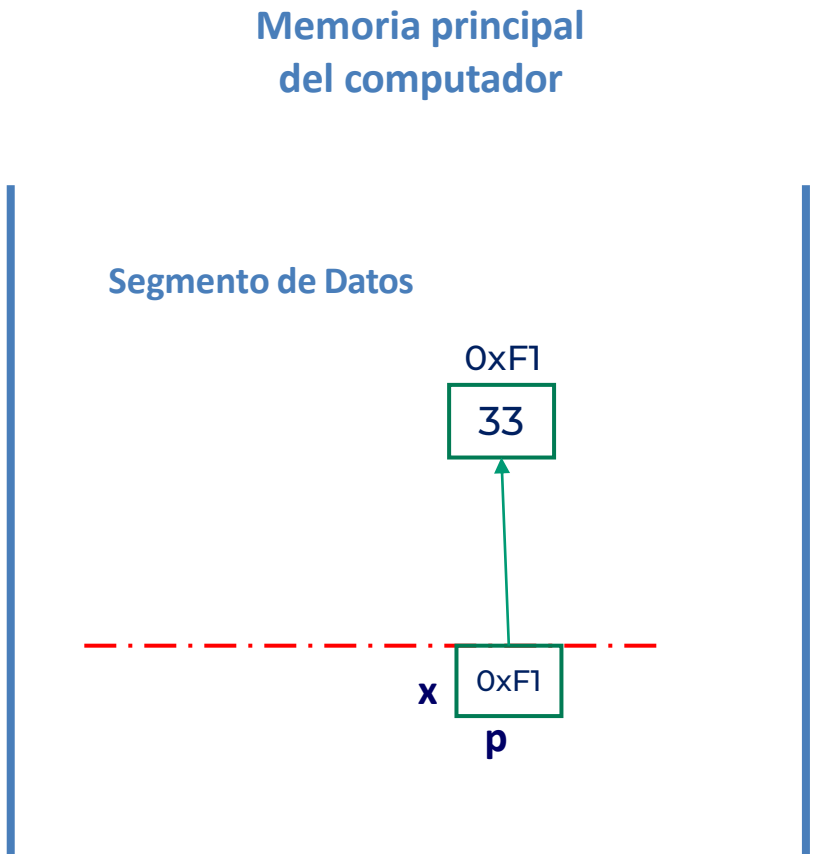
Memoria principal
del computador




```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Referencia

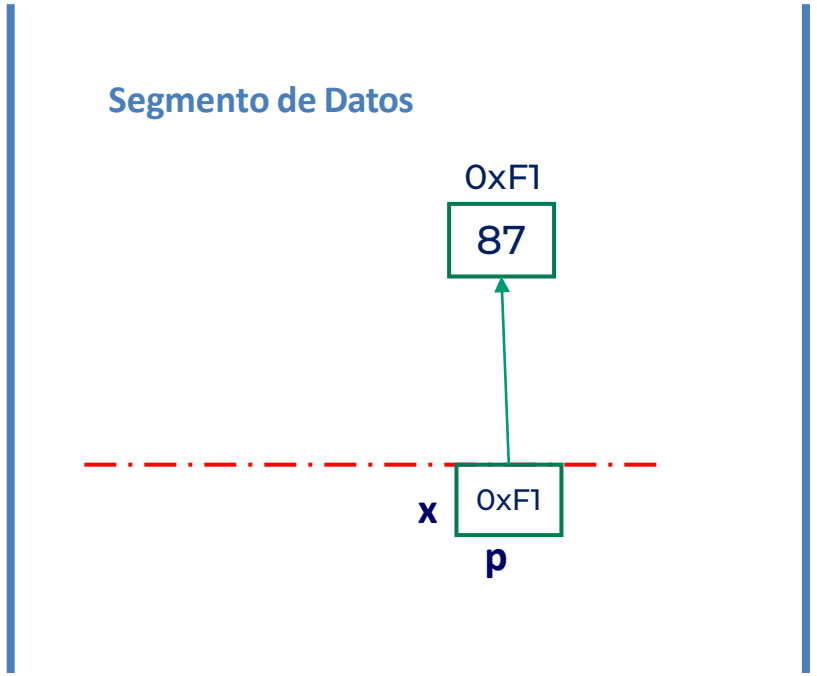



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *&x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Referencia

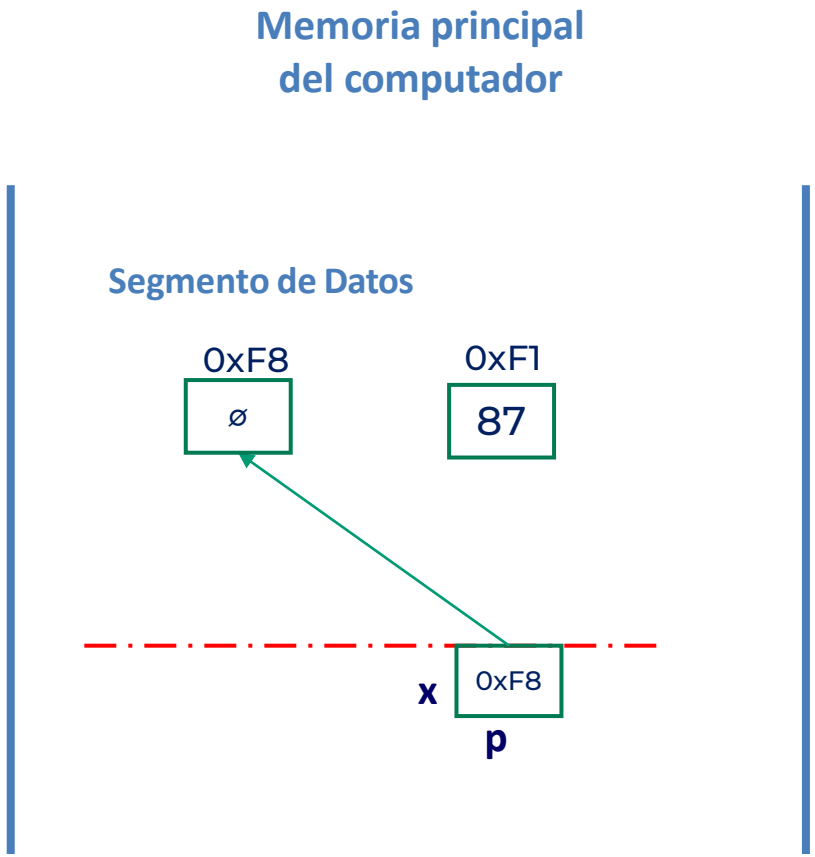
Memoria principal
del computador




```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *&x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Referencia

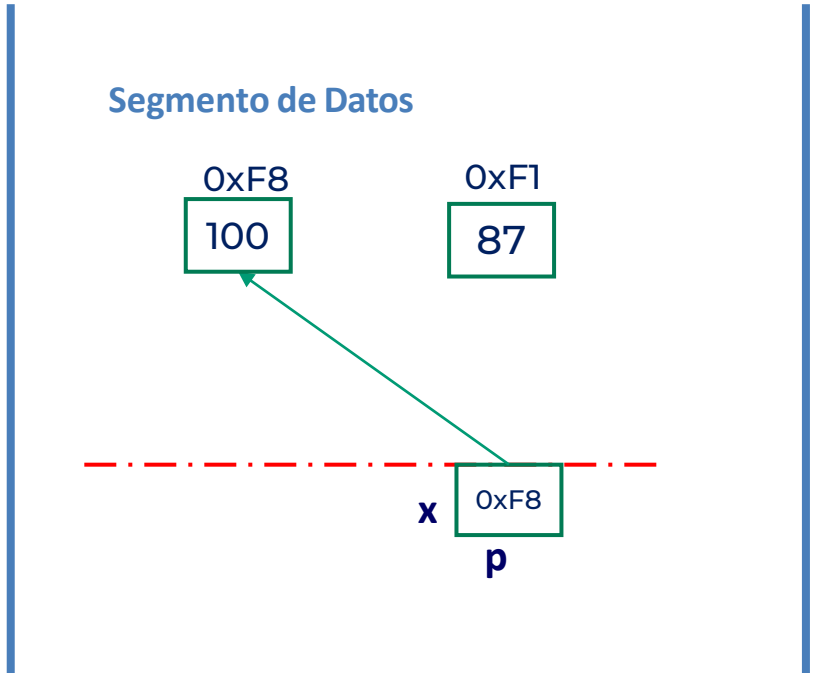



```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *&x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Referencia

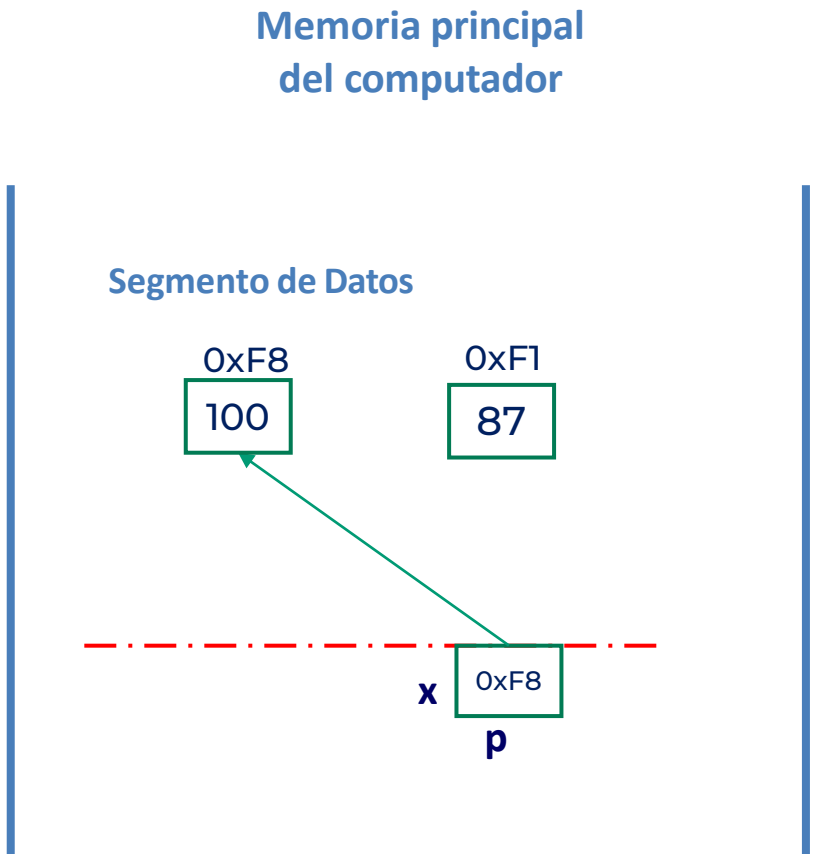
Memoria principal
del computador




```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *&x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

Paso por Referencia




```
int main()
{
    int* p;
    p = new int;
    *p = 33;
    porValor(p);
    cout<<"P="<<p<<endl;//??
}
```

```
void porValor(int *&x)
{
    *x = 87;
    x = new int;
    *x = 100;
}
```

¡Gracias!

PUCP