

```
int i  
#include<iostream>
```

```
Return 0
```

```
While(i <= j) {  
    }
```

```
if(x > 4) {  
    }
```

```
10100100100
```

```
00001100100
```

```
11101010010
```

```
0101
```

```
0  
1  
1  
1  
1  
0
```

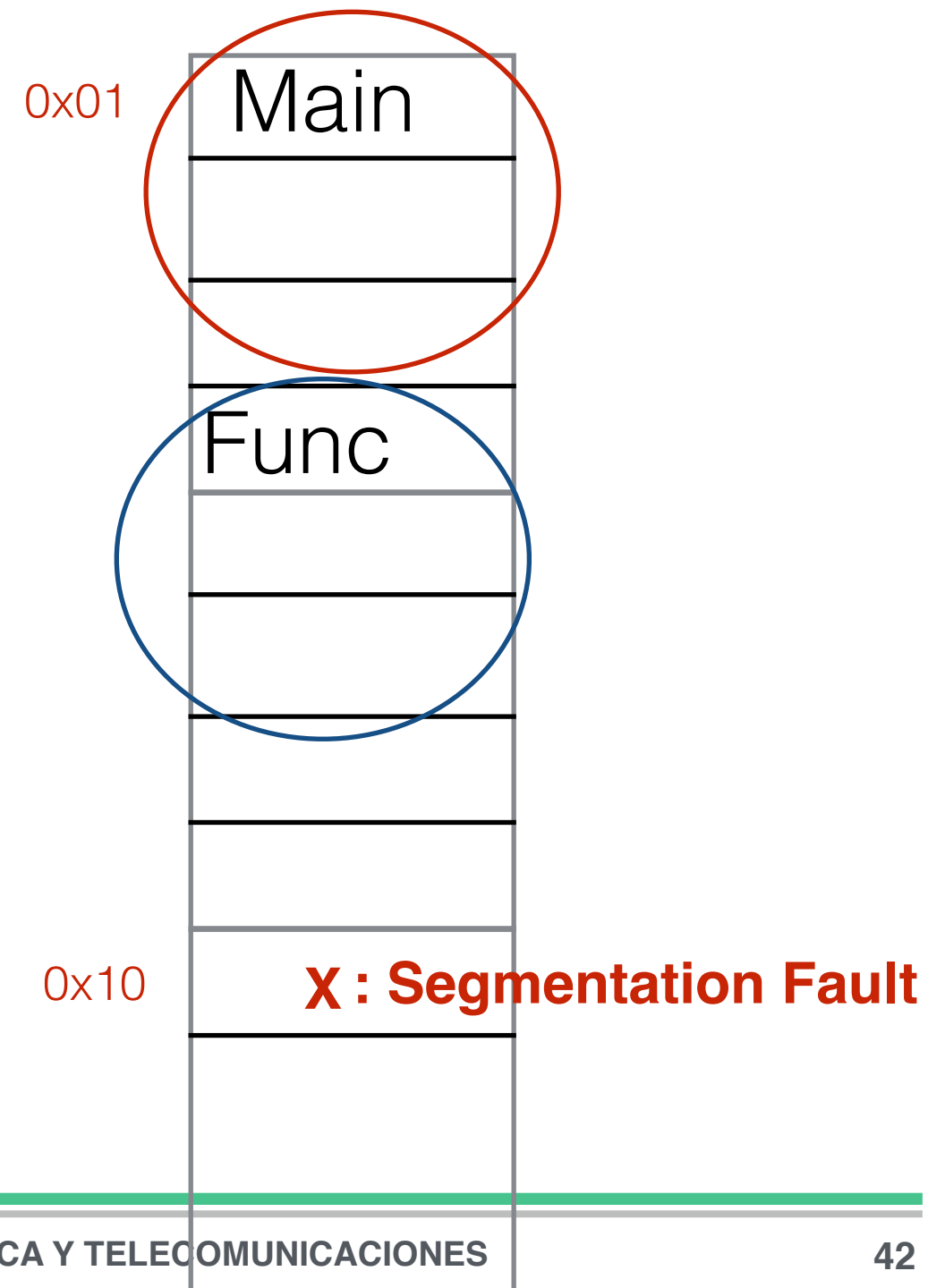
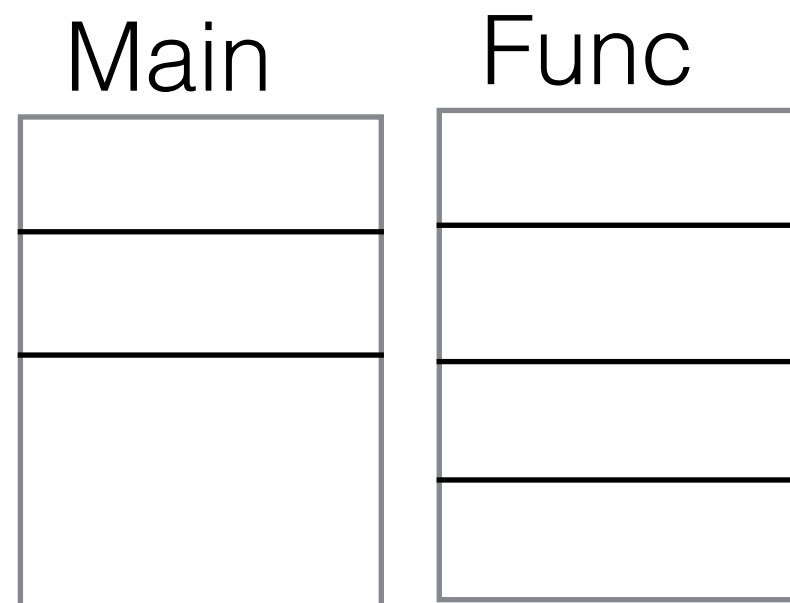
Punteros

Prof. Nicolás Hidalgo
nicolas.hidalgoc@mail.udp.cl



Punteros

- **¿Qué es un puntero?**
 - es una variable que “apunta a” otras variables, una referencia
 - variables se almacenan en memoria
 - un puntero referencia a un dato almacenado en algún lugar de la memoria

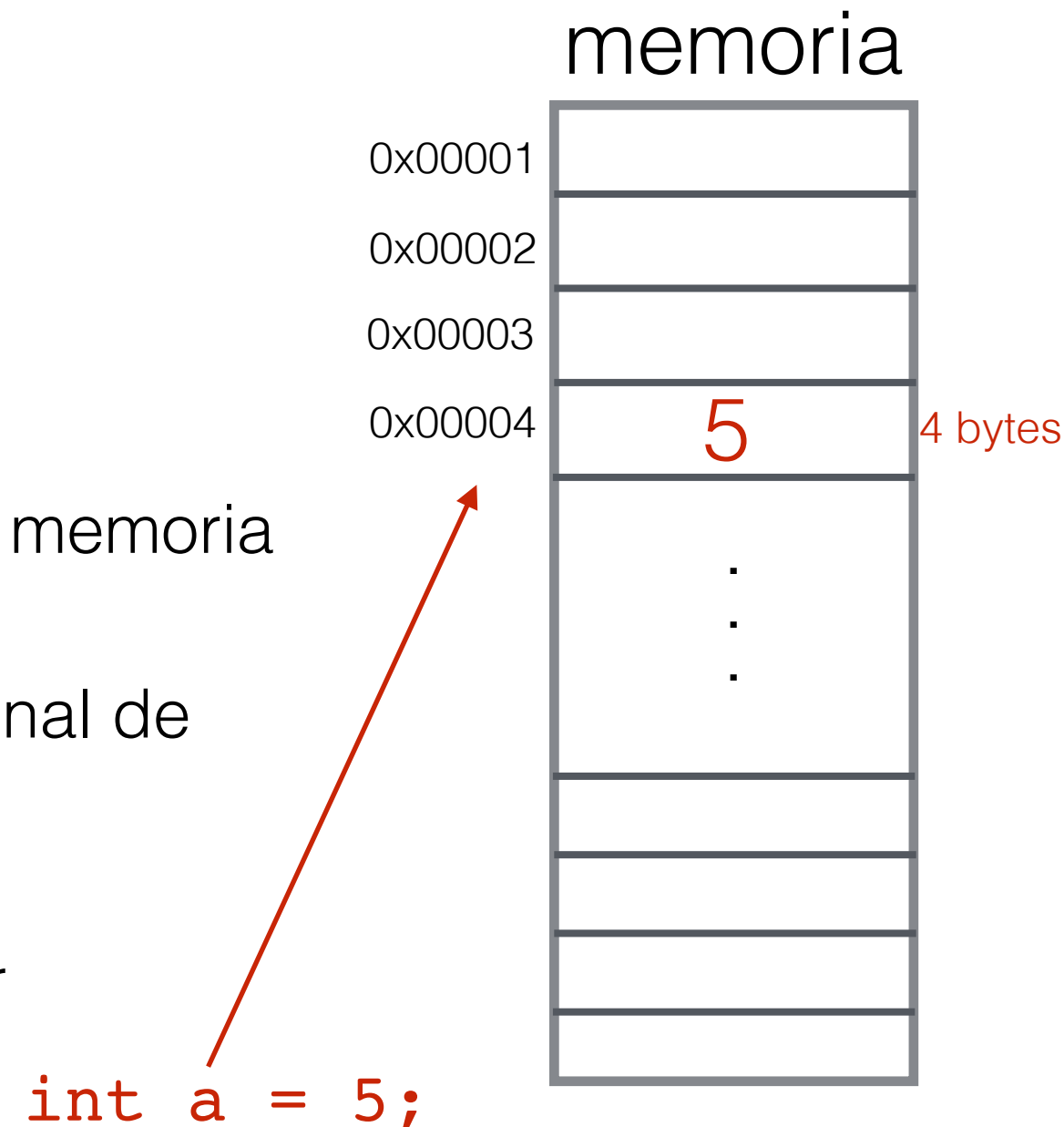


Punteros

- **¿Por qué?**
 - Permiten crear código eficiente y rápido
 - Proporcionan asignación de memoria dinámica
 - Hacen expresiones compactas y concisas
 - Protegen datos pasados como parámetros a una función
 - Proporcionan la capacidad de pasar estructuras de datos mediante un puntero sin ocasionar un exceso de código conocido como “overhead”

Punteros

- ¿Cómo se organiza la memoria en un computador?
- variables en c++ se almacenan en memoria
- memoria es un arreglo unidimensional de espacios de memoria
- cada espacio tiene un identificador



Punteros

- **¿Cómo se definen?**
 - *tipo_dato * nombre;*
 - *Ejemplo: int *ptr; //Esto es un puntero de tipo int llamado ptr*
 - *int *ptr* se lee como “Lo apuntado por ptr es un entero”

Punteros

Main	
0x01	a: 5
0x02	
0x03	b: 0x01
0x04	c:

- **Operadores:**
 - & (ampersand): dirección
 - * (asterisco): lo apuntado

```
#include <iostream>

using namespace std;

int main (){

    int a;
    int *b;
    int *c;

    a = 5;
    b = &a;

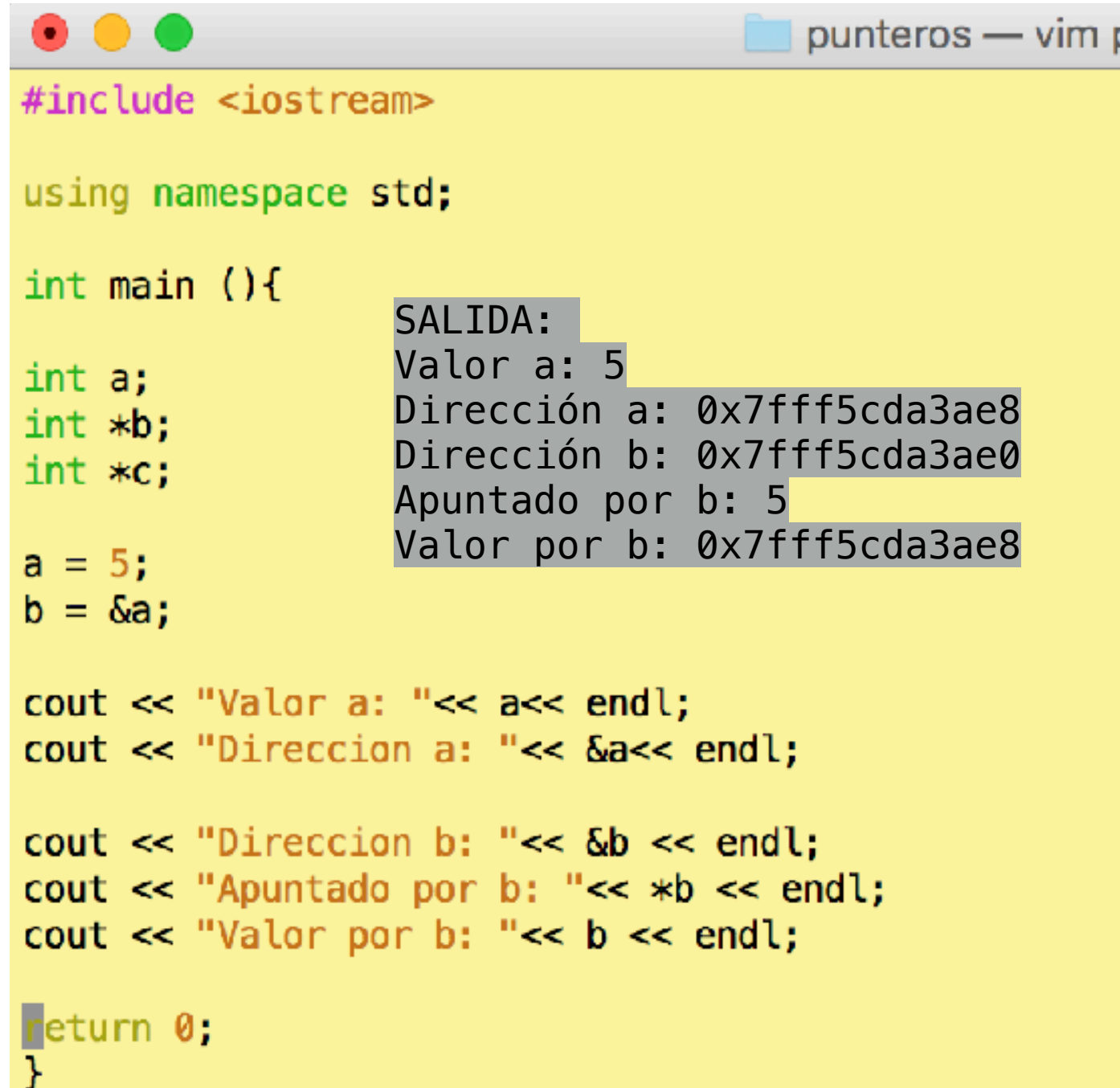
    cout << "Valor a: " << a << endl;
    cout << "Direccion a: " << &a << endl;

    cout << "Direccion b: " << &b << endl;
    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;

    return 0;
}
```

Punteros

- **Operadores:**
 - & (ampersand): dirección
 - * (asterisco): lo apuntado



```
#include <iostream>

using namespace std;

int main (){

    int a;
    int *b;
    int *c;

    a = 5;
    b = &a;

    cout << "Valor a: " << a << endl;
    cout << "Direccion a: " << &a << endl;

    cout << "Direccion b: " << &b << endl;
    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;

    return 0;
}
```

SALIDA:
Valor a: 5
Dirección a: 0x7fff5cda3ae8
Dirección b: 0x7fff5cda3ae0
Apuntado por b: 5
Valor por b: 0x7fff5cda3ae8

Operador *

- El operador * es el operador llamado de **derreferenciación**
- Lo que hace es entregar el valor que está en la dirección de memoria.
 - En otras palabras, * significa “lo apuntado por”.

Operador *

- Al derreferenciar un puntero a entero, se obtiene un entero. El puntero derreferenciado puede ser usado en cualquier contexto en que un entero sea válido:

```
int x, y;  
int *p;
```

```
x = 5;  
p = &x;
```

```
cout << *p;  
y = *p * 10 - 7;  
*p = 9;
```

Operador *

- Al derreferenciar un puntero a entero, se obtiene un entero. El puntero derreferenciado puede ser usado en cualquier contexto en que un entero sea válido:

```
int x, y;  
int *p;
```

```
x = 5;  
p = &x;
```

```
cout << *p;    /* imprime 5 */  
y = *p * 10 - 7;    /* y toma el valor 43 */  
*p = 9;          /* x toma el valor 9 */
```

Operador *

- Ojo, dereferenciar un puntero **NULL** no es Valido!
- **Segmentation Fault!**
- Si existe alguna remota posibilidad de que un puntero pueda tener el valor **NULL**, lo sensato es revisar su valor antes de dereferenciarlo:

```
if (p != NULL)  
    hacer_algo(*p);
```

Punteros

```
int a = 5;
int *ptr = NULL;
```

Dirección	Contenido
0x8130	0x00000005
0x8134	0x00000000

```
ptr = &a;
```

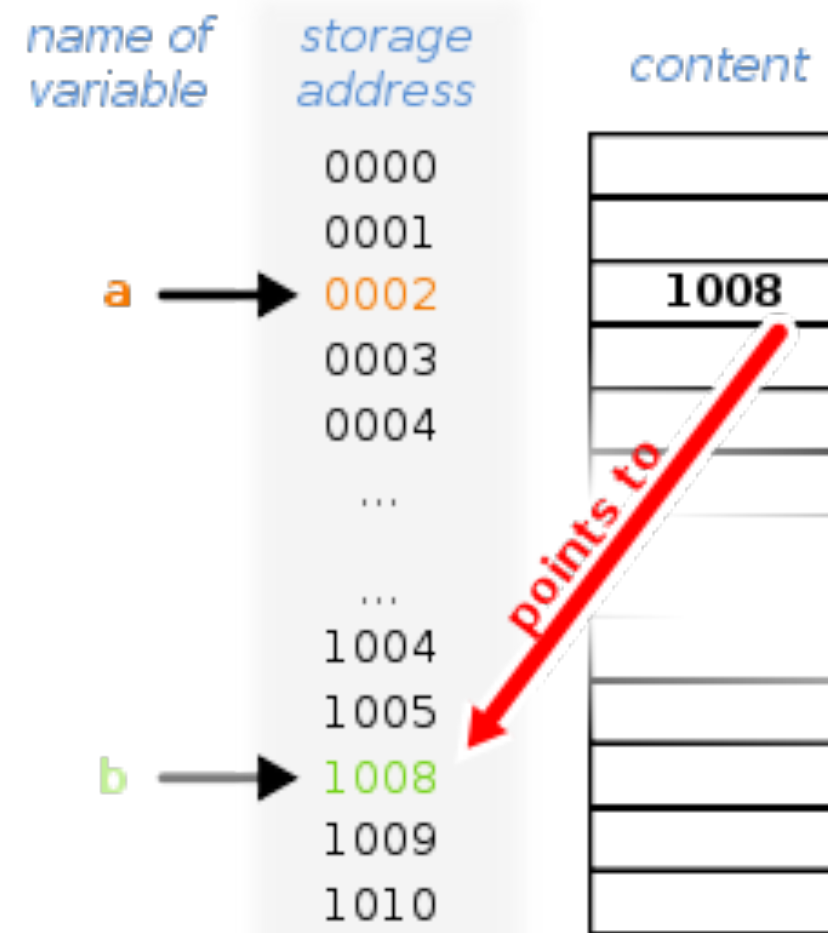
Dirección	Contenido
0x8130	0x00000005
0x8134	0x00008130

```
*ptr = 8;
```

Dirección	Contenido
0x8130	0x00000008
0x8134	0x00008130

Ejemplos punteros

- `int *a, *b;`
- `a = &b`



punteros — vim puntero3.cpp -

```
#include <iostream>

using namespace std;

int main (){

    int a;
    int *b;

    a = 5;
    b = &a;

    cout << "Valor a: " << a << endl;
    cout << "Direccion a: " << &a << endl;
    cout << "-----" << endl;
    cout << "Direccion b: " << &b << endl;
    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;
    cout << "-----" << endl;
    cout << "Despues de sumar...." << endl;

    b = b + 2;

    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;
    cout << "-----" << endl;

    return 0;
}
```

~

¿Qué pasará?

¿Qué pasará?

```
punteros — vim puntero3.cpp
#include <iostream>

using namespace std;

int main (){

    int a;
    int *b;

    a = 5;
    b = &a;

    cout << "Valor a: " << a << endl;
    cout << "Direccion a: " << &a << endl;
    cout << "-----" << endl;
    cout << "Direccion b: " << &b << endl;
    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;
    cout << "-----" << endl;
    cout << "Despues de sumar...." << endl;

    b = b + 2;

    cout << "Apuntado por b: " << *b << endl;
    cout << "Valor por b: " << b << endl;
    cout << "-----" << endl;

    return 0;
}
~
```

Valor a: 5

Dirección a: 0x7fff53014a88

Dirección b: 0x7fff53014a80

Apuntado por b: 5

Valor por b: 0x7fff53014a88

Después de sumar....

Apuntado por b: 213827568

Valor por b: 0x7fff53014a90

¿Válido o inválido?

- Para cada sentencia indique si es **válida** la asignación o **inválida**:

```
int a, b, c;
```

```
float z;
```

```
int *p;
```

```
int *q;
```

```
p = NULL;
```

```
p = &a;
```

```
p = &b;
```

```
p = &z;
```

```
p = 142857;
```

```
q = &b;
```

```
q = p;
```

```
q = NULL;
```

```
q = &p;
```

¿Válido o inválido?

- Para cada sentencia indique si es **valida** la asignación o **invalida**:

```
int a, b, c;
```

```
float z;
```

```
int *p;
```

```
int *q;
```

```
p = NULL; /* valido */
```

```
p = &a; /* valido */
```

```
p = &b; /* valido */
```

```
p = &z; /* invalido (z no es un entero) */
```

```
p = 142857; /* invalido (142857 no es una dirección de memoria) */
```

```
q = &b; /* valido */
```

```
q = p; /* valido */
```

```
q = NULL; /* valido */
```

```
q = &p; /* invalido, no es un puntero a un puntero */
```

Algunos detalles...

- Ambas son sentencias válidas

```
int *x, *y;    /* x e y son punteros */
```

```
int *x, y;     /* x es puntero, y es entero */
```

Ejercicio

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

```
int main() {
    float w, z;
    float *p, *q;

    w = 20;
    p = &z;
    q = p;
    *q = 7;
    z += *q;
    w -= *p;
    p = &w;
    *q += *p;
    z += *(&w);
    p = q;
    *p = *q;

    cout << w << z << *p << *q;
    return 0;
}
```

memoria

0x00001	
0x00002	
0x00003	
0x00004	
	.
	.
	.
0x00020	
0x00021	
0x00022	
0x00023	

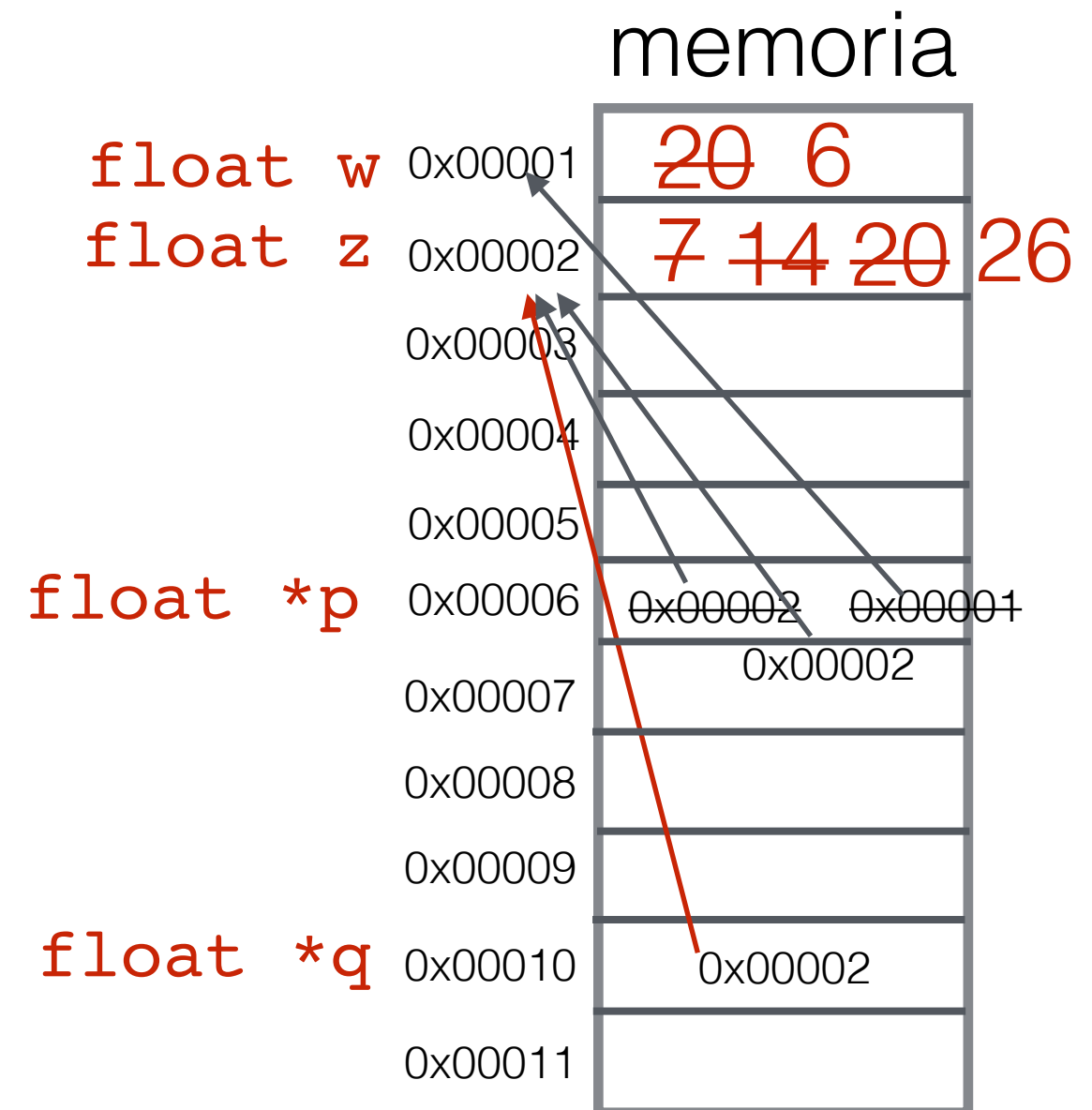
Ejercicio

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

```
int main() {
    float w, z;
    float *p, *q;

    w = 20;
    p = &z; //
    q = p;
    *q = 7;
    z += *q;
    w -= *p;
    p = &w;
    *q += *p;
    z += *(&w);
    p = q;
    *p = *q;

    cout << w << z << *p << *q;
    return 0;
}
```



¿Qué imprime el programa? Justifique realizando esquema de memoria.

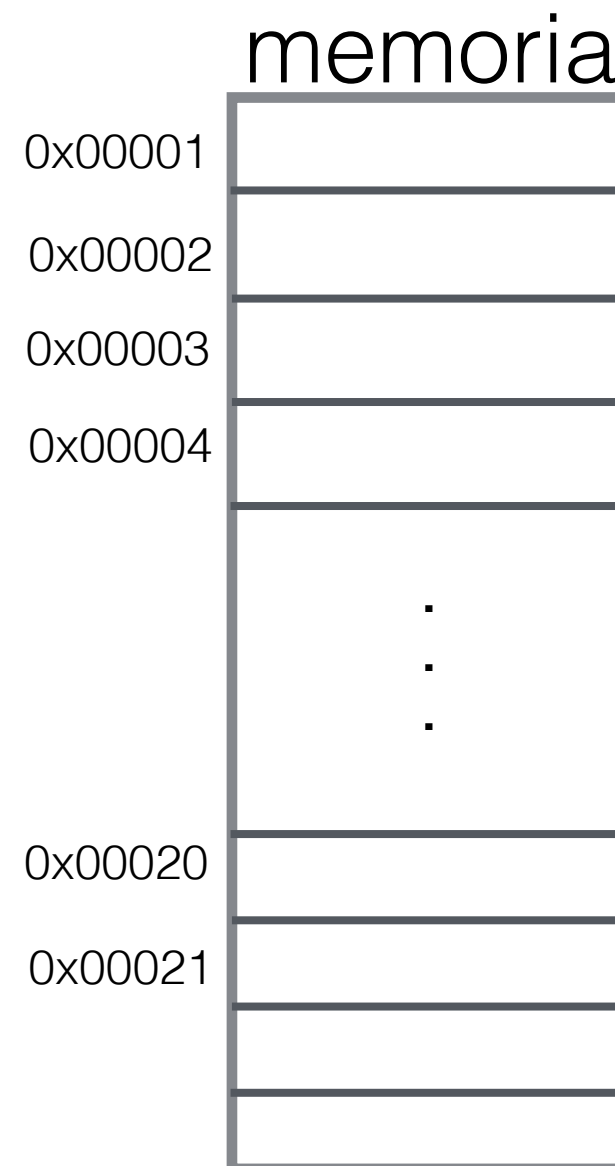
```
int main() {  
    int a;  
    int b;  
    int *c;  
  
    a = 2;  
    b = 3;  
    cout<< "a: " << a << ", b: " << b << endl;  
  
    c = &a;  
    *c = 5;  
    cout<< "a: " << a << ", b: " << b << endl;  
  
    c = &b;  
    *c = 6;  
    cout<< "a: " << a << ", b: " << b << endl;  
    cout << "*c: " << *c << endl;  
  
    b = 8;  
    cout << "*c: " << *c << endl;  
    return 0;  
}
```

Ejercicio

- Genere una función llamada *sumoDoble* de tipo `int` la cual recibe un puntero. La función suma el doble de su valor al valor actual de entrada definido en el `main` del código.

Ejercicio

- Genere una función llamada *sumoDoble* de tipo void la cual recibe un puntero. La función suma el doble de su valor al valor actual de entrada definido en el main del código.



Ejercicio

- Genere una función *calculaPromedio* del tipo void que recibe 4 parámetros, donde con los 3 primeros calcula el promedio y el resultado se maneja en el 4to parámetro que es un puntero.

Ejercicios

- Desde hace un tiempo las monedas de \$1 y \$5 dejaron de circular, lo que ha obligado a aproximar hacia el número superior (cifras terminadas en 5, 6, 7, 8, 9) o inferior (cifras terminadas en 1, 2, 3, 4) según corresponda. No obstante, cuando el pago es con cheque o tarjeta (crédito o débito) los montos no se truncan. Usted decide apoyar a un cajero para que siempre cobre lo justo. En base a lo anterior se le pide:
 - Cree la función **void valorapagar(int *monto)** que devuelve el monto que debe pagar un cliente si el pago lo hace en efectivo, el monto representa el valor con aproximación.
 - Cree la función **int pagodefinitivo(int *monto, int *formadepago)**, que devuelve el monto que debe pagar el cliente asumiendo que:
 - Si formadepago = 1 paga en efectivo, (llama a función **valorapagar()**)
 - Si formadepago = 2 paga con cheque
 - Si formadepago = 3 paga con TCrédito
 - Si formadepago = 4 paga con TDébito.

Programe el método main() de modo que reciba de teclado el monto a pagar y la forma de pago y utilizando las funciones definidas anteriormente permita imprimir el valor real a pagar.

Ejercicio

- Un conocido matemático descubrió un conjunto de números enteros que denominó como los meta-números. Un meta-número es número compuesto por **al menos 3** cifras **primas seguidas**. Por ende un meta-numero nunca puede tener menos de 3 cifras. Escriba un programa que permita a cualquier mortal conocer si el número ingresado es o no un meta-número. Resuelva utilizando al menos las funciones:
 - *bool esPrimo(int * n)*
 - *bool esMetanumero(int *n).*
- **E.g: Ingrese un número: 135790**
 - *Es un meta-número!*