

# CS1112: Programación II

Unidad 3: Punteros

Sesión de Teoría 4-5-6

**Profesor:**

**José Antonio Fiestas Iquira** [jfiestas@utec.edu.pe](mailto:jfiestas@utec.edu.pe)

**Material elaborado por:**

**Maria Hilda Bermejo, José Fiestas, Ruben Rivas**



# Índice:

## Unidad 3: Punteros

1. Definición y usos
2. Manejo de memoria dinámica,  
arreglos dinámicos unidimensionales
3. Arreglos dinámicos bidimensionales

# 3.1

## Unidad 3: Punteros, definición y usos



# Logro de la sesión:

Al finalizar la sesión, los alumnos desarrollan sus programas utilizando punteros. En específico:

- Comprenden la definición y uso de punteros.
- Comprenden y usan la memoria de forma optimizada con arreglos dinámicos.
- Utilizan matrices dinámicas en un código en C++.



## 42nd St, Manhattan

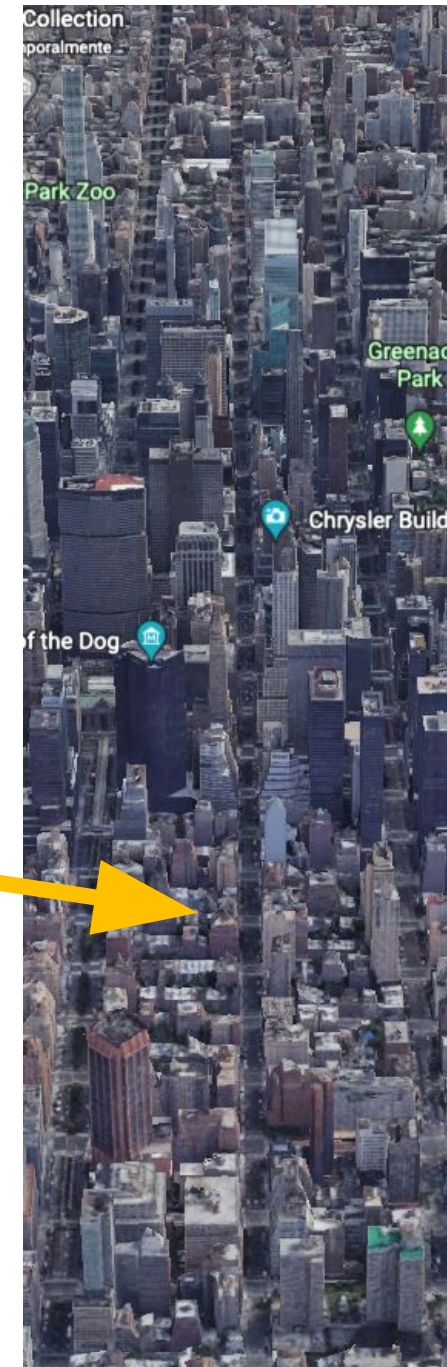
En una calle, cada domicilio tiene una **dirección** asignada.

En un computador, cada Byte tiene una **dirección de memoria** asignada



## 42nd St, Manhattan

Un **puntero** me permite acceder a la **dirección de memoria** y manipular eficientemente **mayor cantidad de datos**



# ¿Qué ocurre con el código desde que se escribe hasta que se ejecuta?

header file



source  
code

```
#include  
<stdio.h>  
int main()  
{  
    printf("Hola");  
    ;  
    return 0;  
}
```

Compiler

object file



Libraries

Link

executable  
code

```
100010101010101  
000100101010111  
011111100110000  
001011001101010  
010111011100011  
011111001111000  
000110011110101  
010010010101000
```

Una vez que se tiene el código ejecutable, el programa sube a memoria principal (RAM)

Memoria  
RAM

# Uso de la memoria primaria en C++

## Segmento de Información Externa

Argumentos externos (argc, argv)

## Pila (Stack)

Memoria automática donde se asigna las variables estáticamente.



## Montón (Heap)

Memoria donde se asigna variables dinámicamente.

## Segmento Estático y Global

Memoria donde se asigna variables globales y tipo static.

## Segmento de Código

Memoria donde se guarda el segmento de código



# Representación simplificada de la memoria

- Cada Byte tiene su propia dirección

**1 Byte**

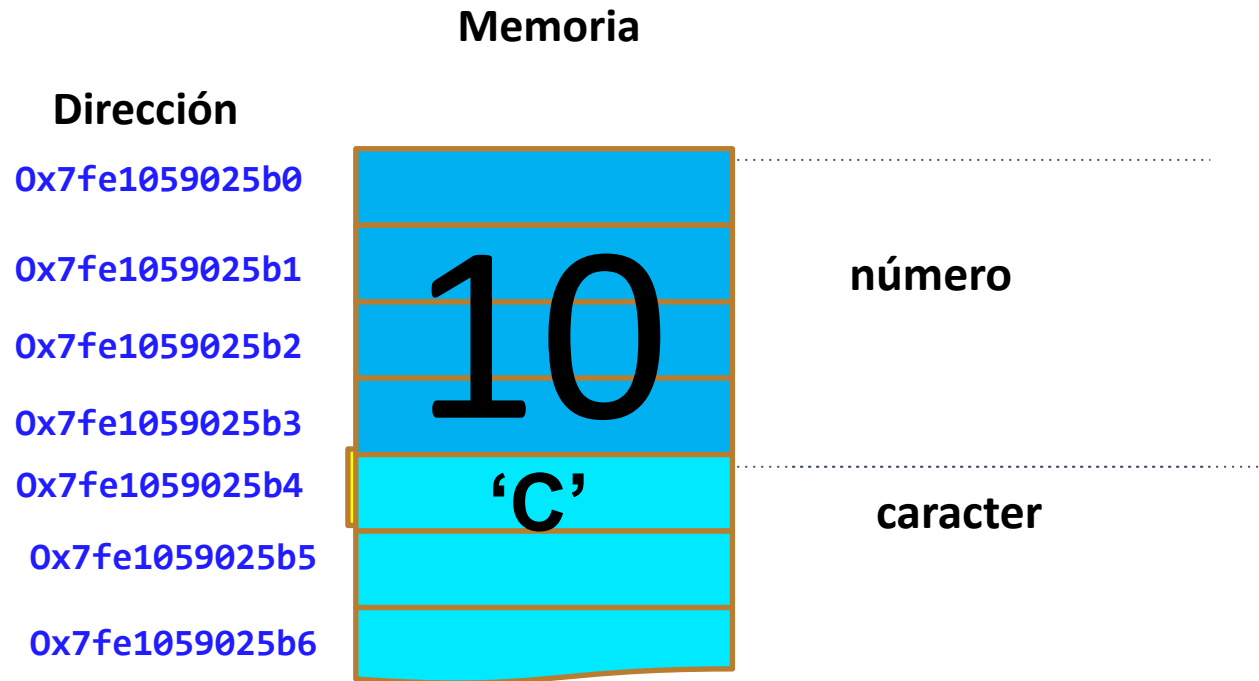
0	1	0	1	0	0	1	1
---	---	---	---	---	---	---	---

Dirección	Memoria
0x7fe1059025b0	
0x7fe1059025b1	
0x7fe1059025b2	
0x7fe1059025b3	
0x7fe1059025b4	
0x7fe1059025b5	
0x7fe1059025b6	
0x7fe1059025b7	

# Representación de una variable

```
int numero = 10;  
char character = '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?

# Dirección de una variable

La forma explícita de obtener la dirección de una variable es por medio del operador **&**

Ejemplo:

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

int main()
{ int numero = 10;
  float x=12.75;

  cout << "Numero = " << numero << "\n";
  cout << "La direccion donde esta la variable numero es " << &numero << "\n";

  cout << "x = " << x << "\n";
  cout << "La direccion donde esta la variable x es " << &x;

  return 0;
}
```

Salida del programa:

Numero = 10

La direccion donde esta la variable numero es 0x7fff5841fa38

x = 12.75

La direccion donde esta la variable x es 0x7fff5841fa34

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

Un **puntero**, es un tipo de componente que almacena una dirección de memoria (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.

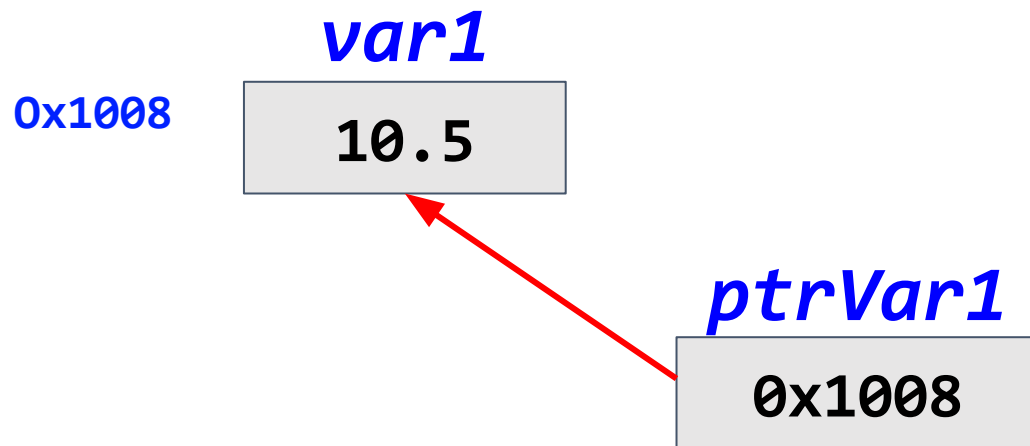
```
int ival = 42;  
int* p = &ival;    // p contiene la dirección de ival;  
                    // p es un puntero a ival  
cout << p << *p << &ival << &p;
```

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 Clion utiliza 8 bytes.

Dirección

Memoria

0x1008

0x1009

0x100A

0x100B

0x100C

0x100D

0x100E

0x100F

0x1010

0x1011

0x1012

0x1013

0x1014

0x1015

0x1016

0x1017

10.5

*var1*

0x1008

*ptrVar1*

# Punteros Nulos

Un ***nullptr*** no apunta a ningún objeto.

```
int* pi = nullptr; // es equivalente a int *pi=0;
int* p2 = 0;       // inicializa p2 con la constante literal 0
int w = 200;
pi = w;            // error, no se puede
                   // asignar un int a un puntero
```

# Más sobre punteros

```
int i = 42;  
int *pi = 0;      // pi es inicializado pero no tiene la dirección de ningún objeto  
int *pi2 = &i;    // pi2 se inicializa y contiene la dirección de i  
int *pi3;         // pi3 no está inicializado  
pi3 = pi2;        // pi3 y pi2 tienen la dirección del mismo objeto (i);  
pi2 = 0;          // pi2 ahora no apunta a ningún objeto.
```

```
int ival = 72;  
pi = &ival;       // El valor de pi cambia, ahora pi apunta a ival  
*pi = 0;          // el valor en ival cambió a cero, pi no cambió
```

# Más sobre punteros

```
int ival = 1024;  
int *pi = 0;           // pi es válido, pi tiene asignado nullptr;  
int *pi2 = &ival;      // pi2 es un puntero válido, tiene la dirección de ival
```

```
if(pi) // pi tiene el valor 0, entonces si se evalúa la condición es falsa
```

```
if(pi2) // pi2 es un puntero a ival, entonces no vale cero, si se evalúa la condición  
es verdadera
```



# Ejercicios:

1. ¿Qué hace el siguiente código?

```
int i = 42;  
int *pi = &i;  
*pi = *pi * *pi;
```

```
int w = 3;  
int *px = nullptr;  
int *pz;  
px = &w;
```

2. Indica si hay definiciones ilegales y ¿por qué?

```
int i = 0;
```

1. `double *p = &i;`
2. `int *pi = i;`
3. `int *p = &i;`

# Ejercicios:

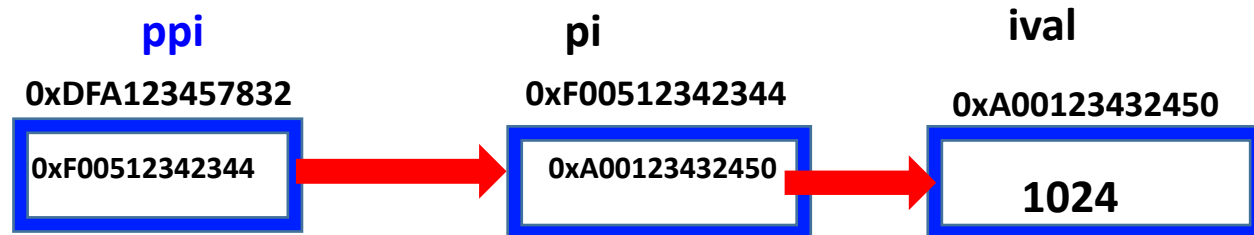
3. ¿Por qué la inicialización de p es legal y la de lp es ilegal?

```
int i = 42;  
void *p = &i;  
long *lp = &i;
```

# 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 = 2;
```

```
// refVal refers to ival (es otro nombre de ival)  
// asigna 2 al objeto al  
// que se refiere refVal,  
// es decir ival = 2;
```

```
int &refVal12;
```

```
// 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.
```



# Analizando código

```
typedef int TNumero;
int main() {
    TNumero x = 5;
    TNumero y = 4;
    TNumero* p;
    TNumero** pp;
    TNumero& r = x;
    x = 7;
    p = &r;
    pp = &p;
    f1(x);
    f1(5);
    f1(*p);
    f1(r);
    f1(**pp);

    return 0;
}
```

```
void f1(TNumero n)
{
    n++;
}
```

```
typedef int TNumero;

int main()
{ TNumero  x=5, y=4, *p, **pp;
  int &r=x;

  x=7;
  p =&r;
  pp = &p;

  f2(x);
  f2(5);  // error
  f2(x+4); // error
  f2(*p);
  f2(r);  // f2(x);
  return(0);
}
```

```
void f2(TNumero & rn)
{
    rn++;
}
```

```
typedef int TNumero;
```

```
int main()  
{ TNumero x=5, y=4, *p, **pp;  
  int &r=x;  
  
  x=7;  
  p =&r;  
  pp = &p;  
  
  f3(&x);  
  f3(&r);  
  f3(p);  
  f3(*pp); // f3(&x); f3(p);  
  return(0);  
}
```

```
void f3(TNumero * pnum)  
{  
  ++*pnum;  
  pnum= nullptr;  
}
```

```
typedef int TNumero;
```

```
int main()
```

```
{ TNumero x=5, y=4, *p, **pp;  
  TNumero &r=x;
```

```
x=7;
```

```
p = &r;
```

```
pp = &p;
```

```
f4(&x); // error
```

```
f4(&r); // error
```

```
f4(p);
```

```
p = &x;
```

```
f4(*pp); // f4(p);
```

```
return(0);
```

```
}
```

```
void f4(TNumero *& rp)
```

```
{
```

```
  ++*rp;
```

```
  rp=nullptr;
```

```
}
```



## Comparando Transferencia de parámetros

```

int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}

```

```

int fSuma(int a, int b)
{
    return (a+b);
}

template <typename T>
void swap(T &a, T &b)
{ T temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}

```

Se va a analizar paso a paso este programa y se muestra lo que se imprime en la pantalla.

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```

oX7fff530d2a58  
x 5

oX7fff530d2a54  
y 4

oX7fff530d2a48  
p

oX7fff530d2a40  
q

oX7fff530d2a38  
pp

```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

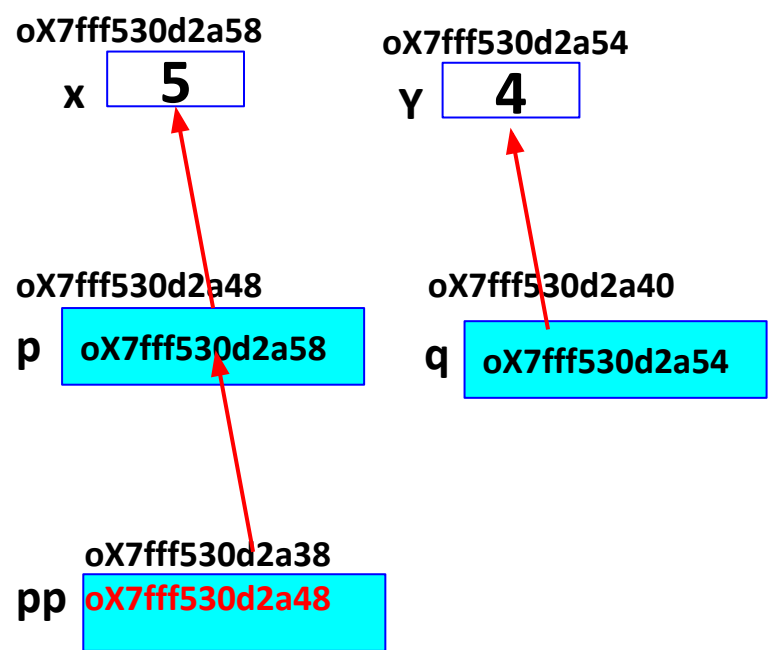
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}
```

Pantalla:

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

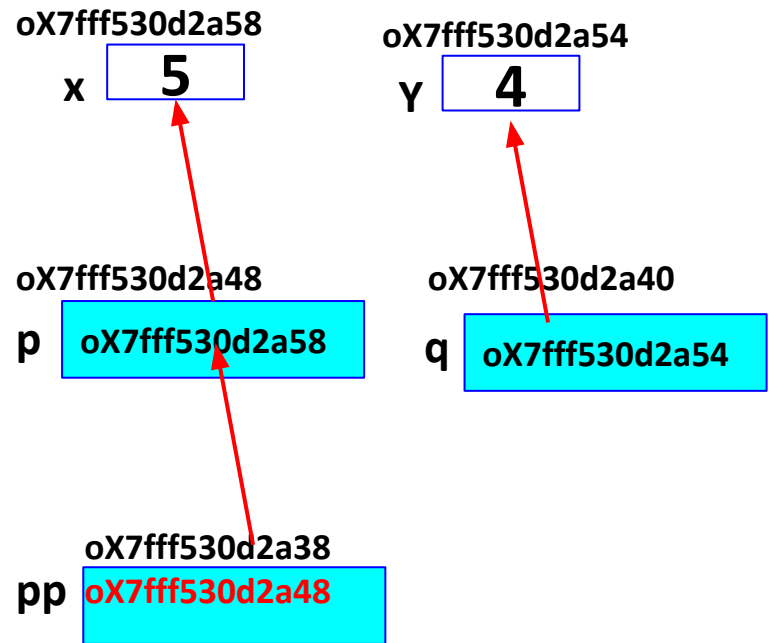
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}
```

Pantalla:

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

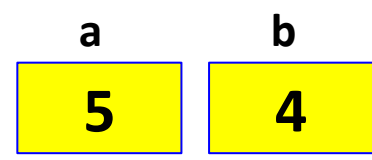
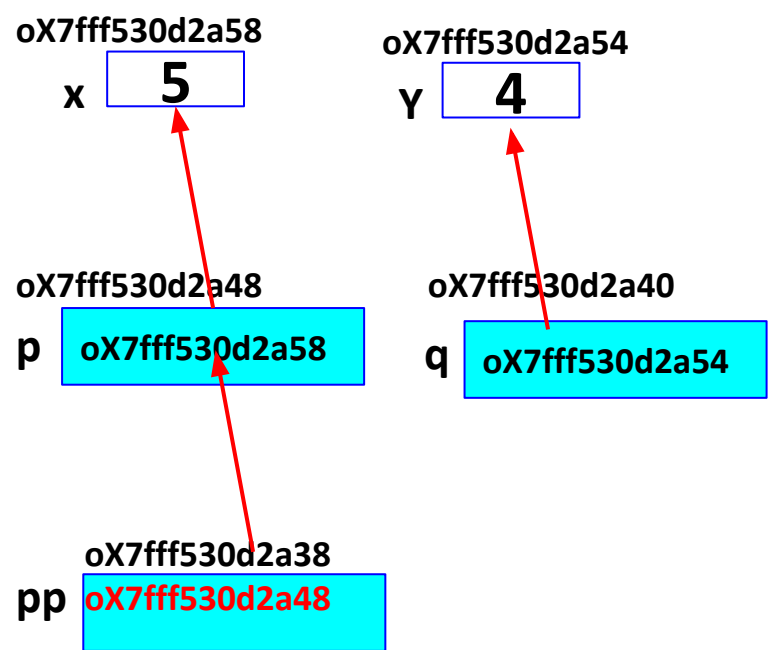
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}
```

Pantalla:  
Suma = 9

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

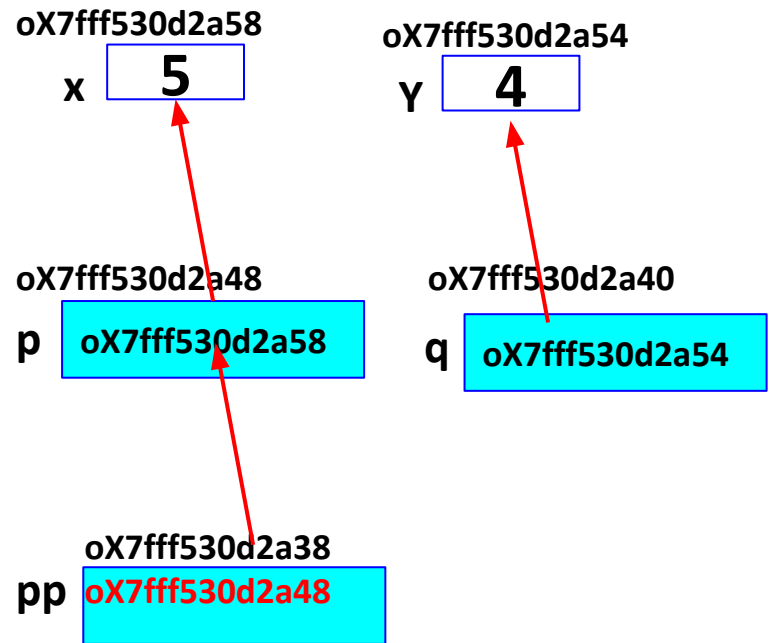
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}
```

Pantalla:  
Suma = 9

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

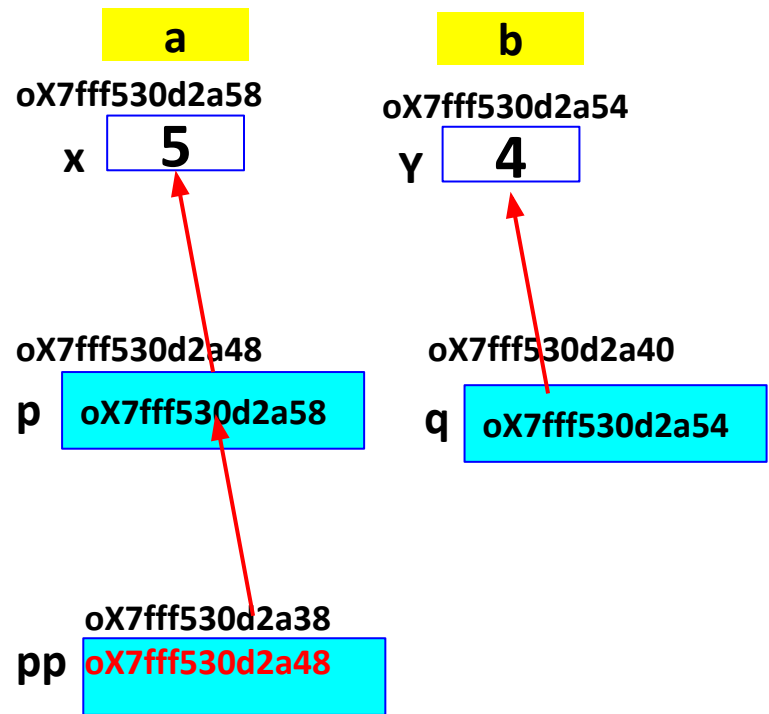
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap



```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

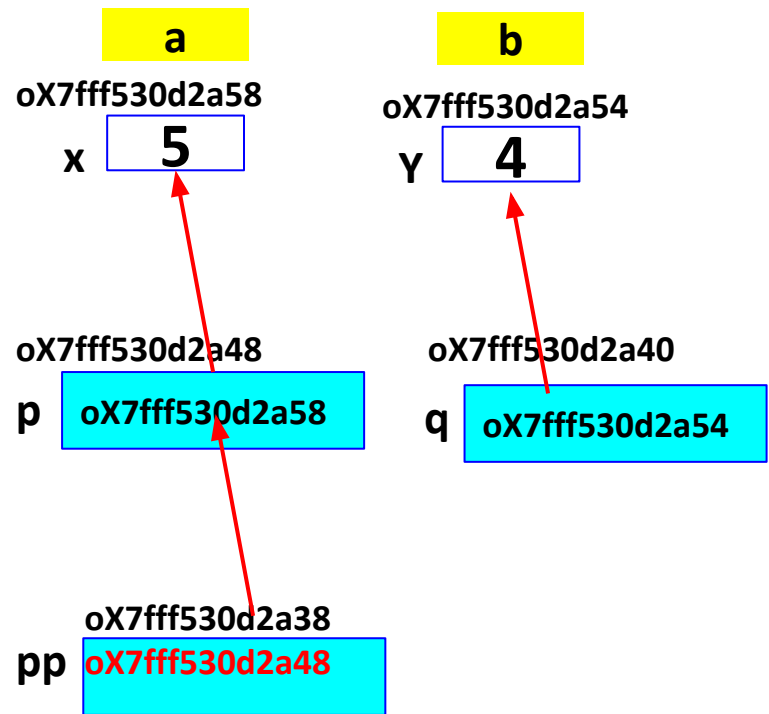
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



temporal  
**5**

```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
**Suma = 9**  
Primer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

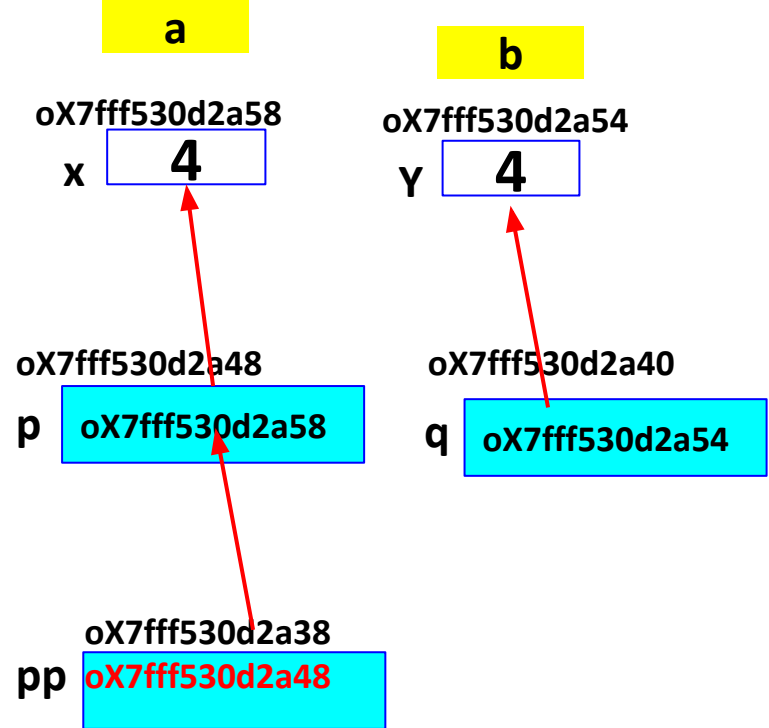
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



temporal  
**5**

```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

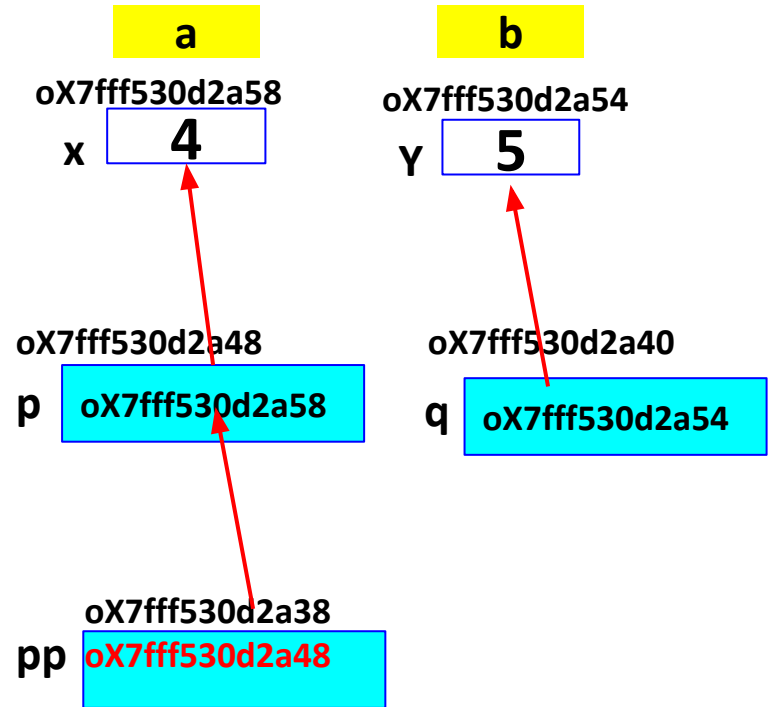
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



temporal  
**5**

```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

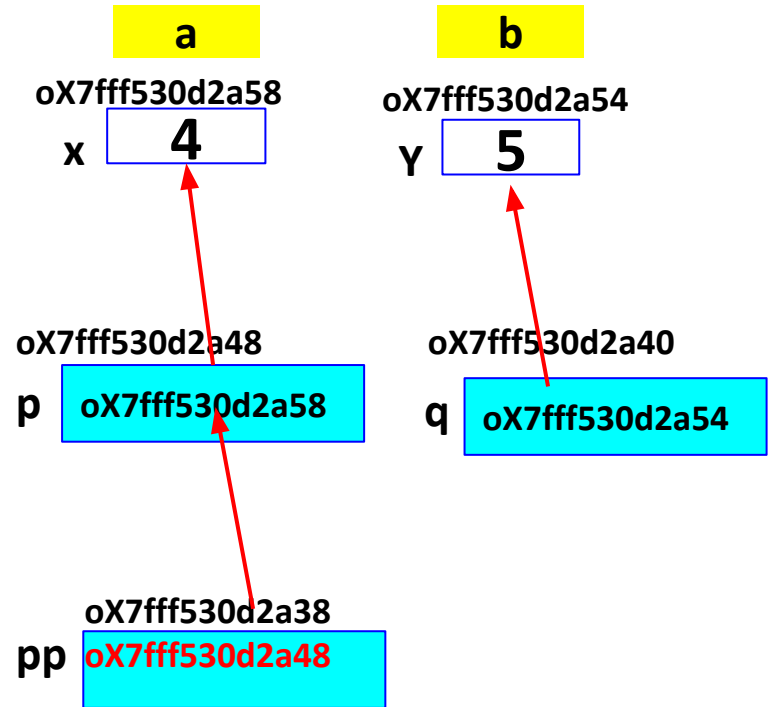
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



temporal  
5

```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

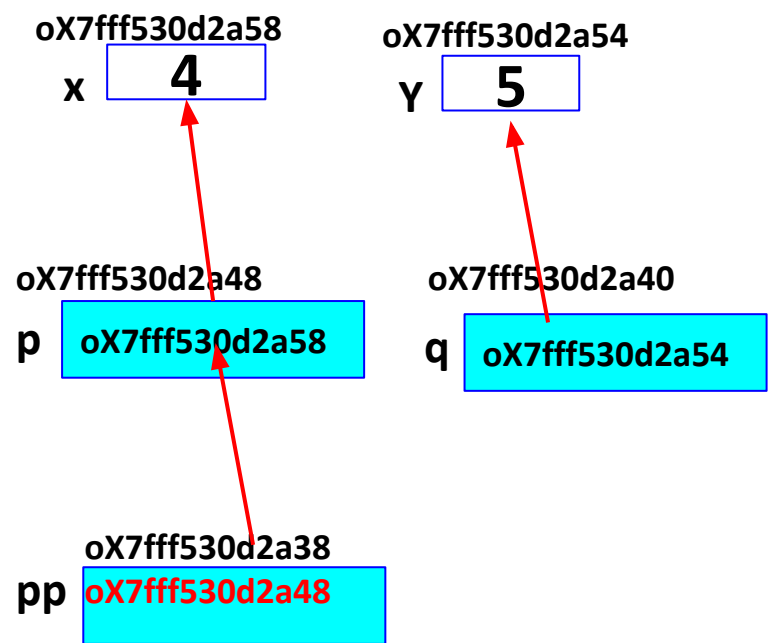
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

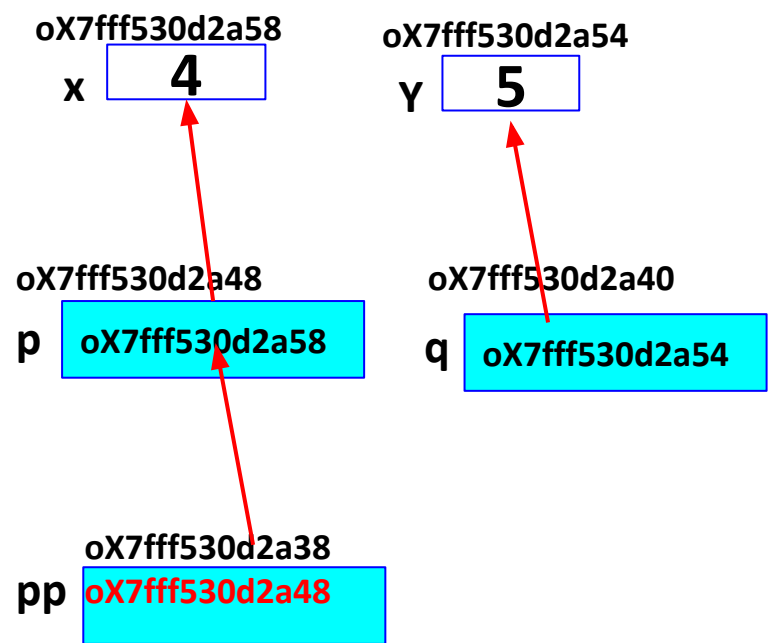
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

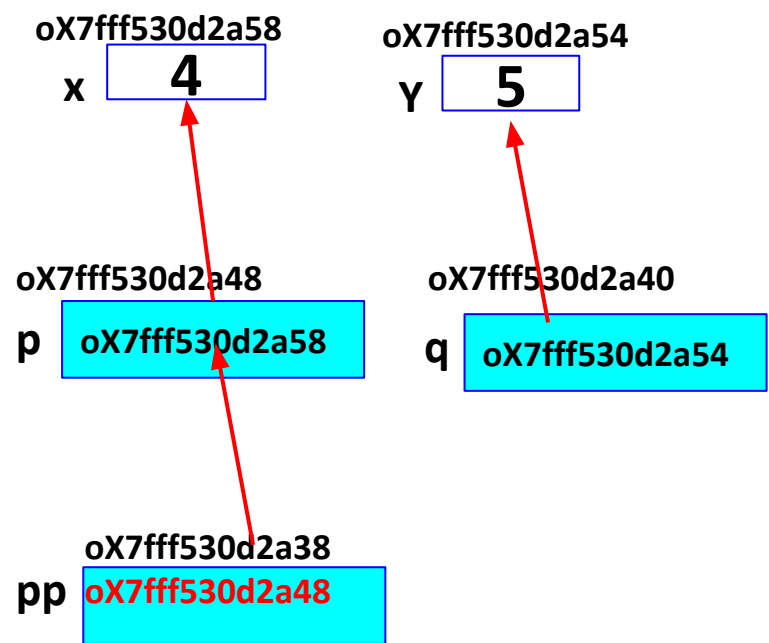
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

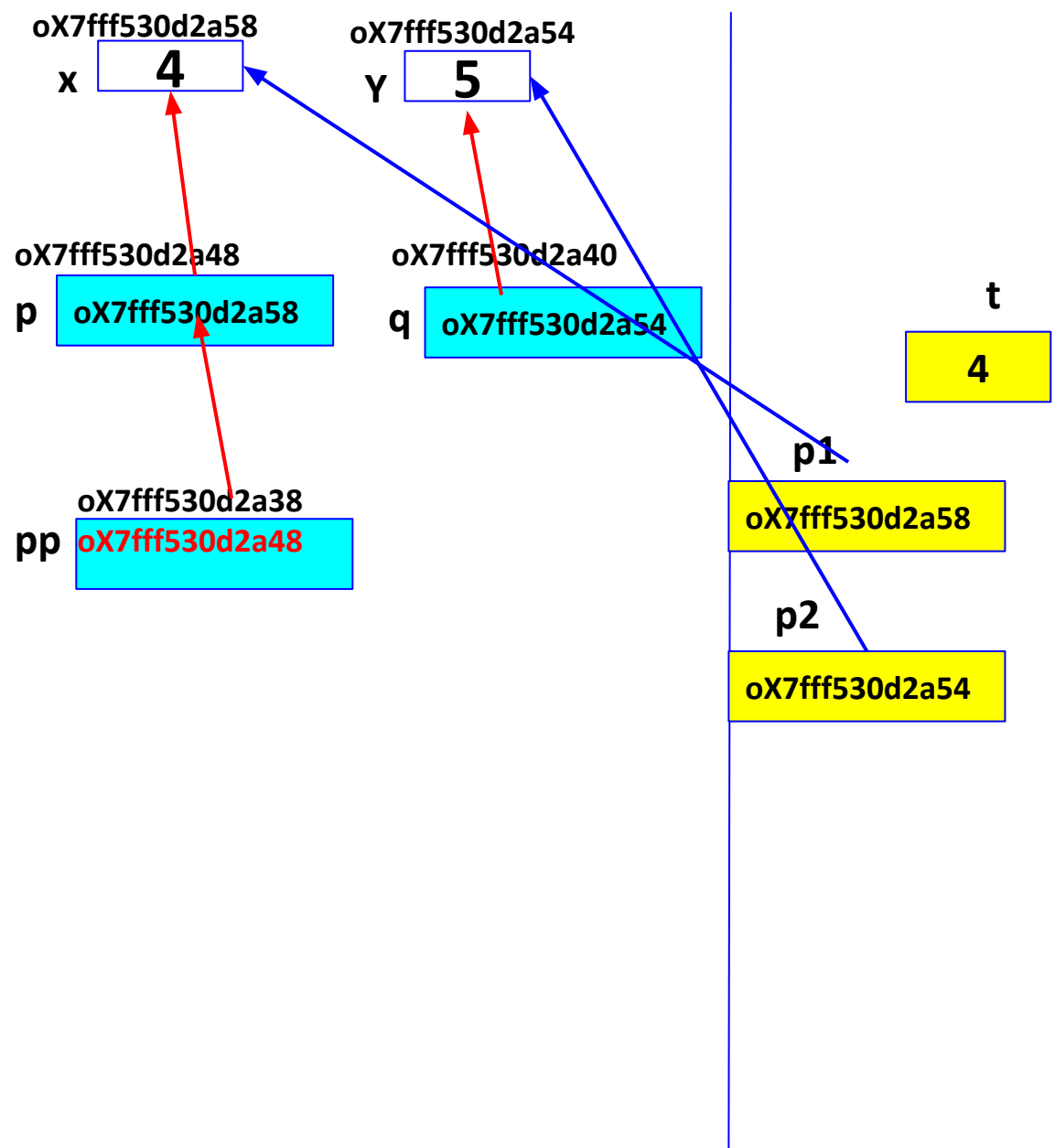
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap



```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

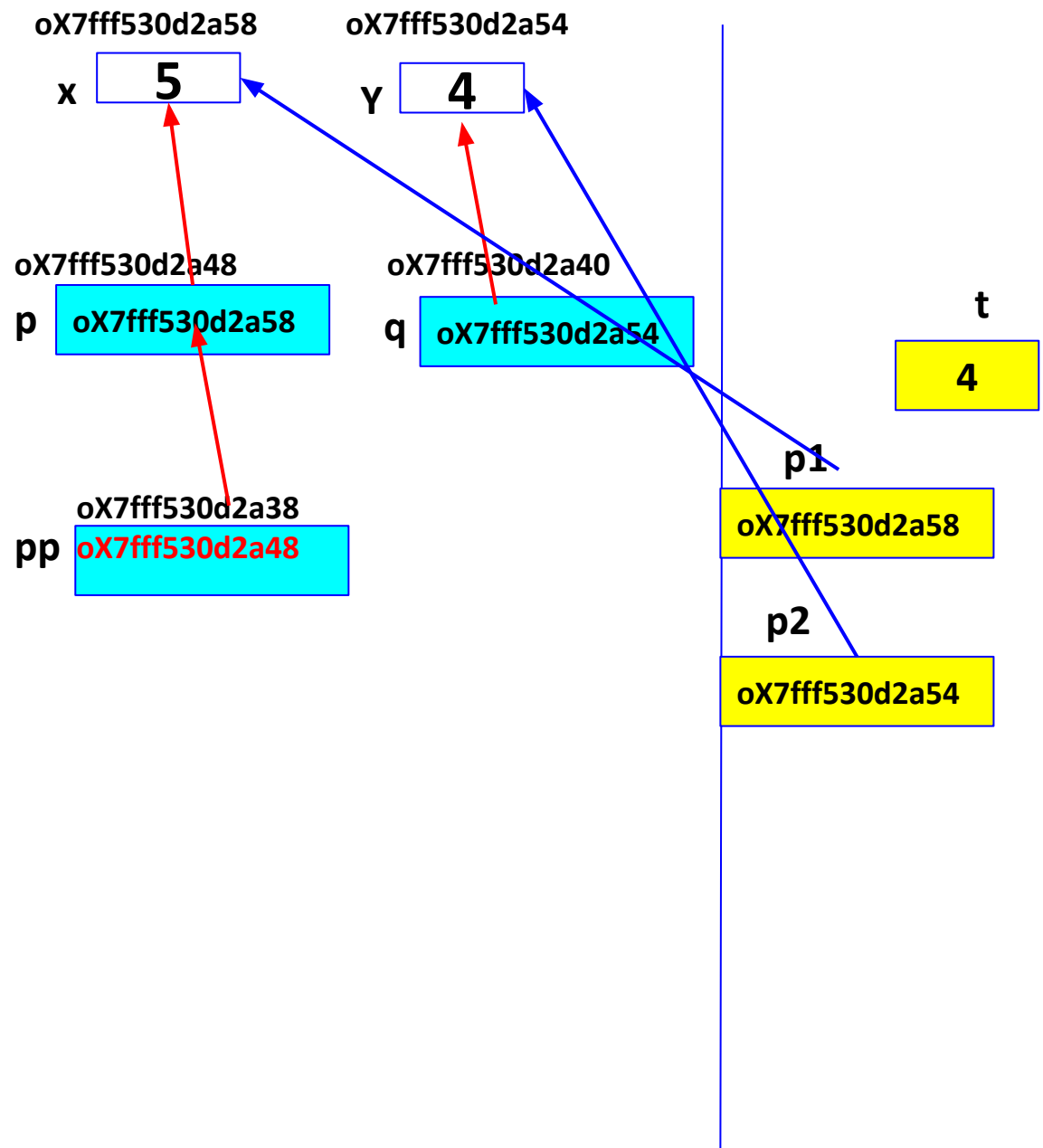
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

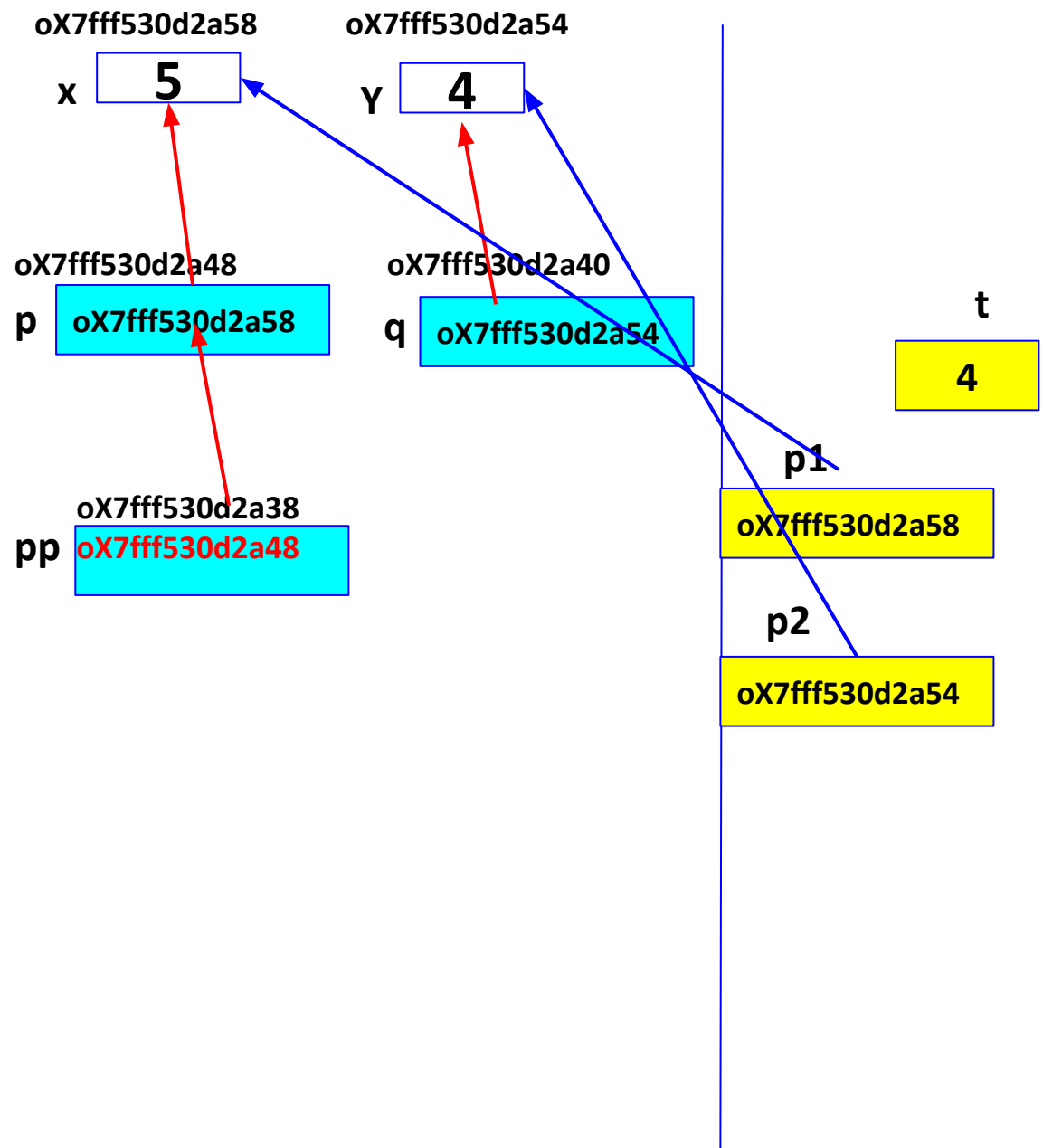
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap  
x = 5 y = 4

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

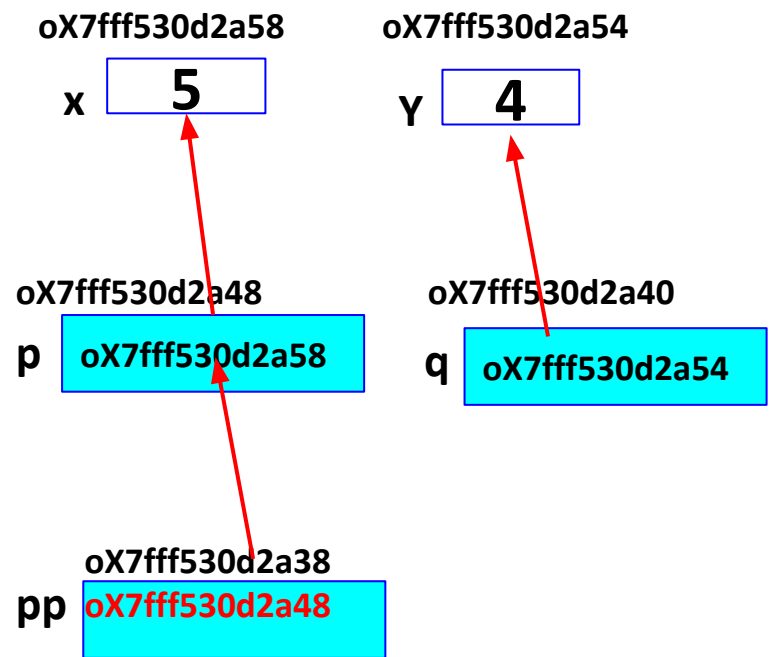
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap  
x = 5 y = 4

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

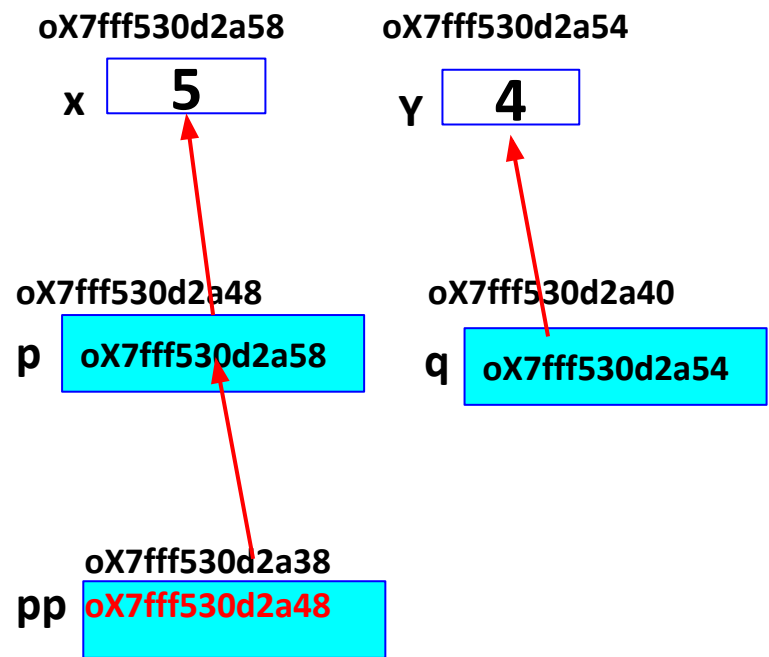
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap  
x = 5 y = 4  
  
Tercer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

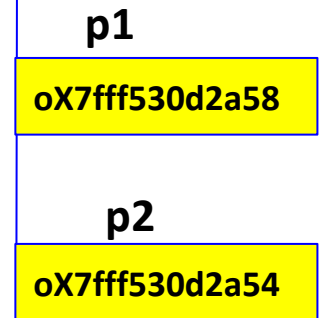
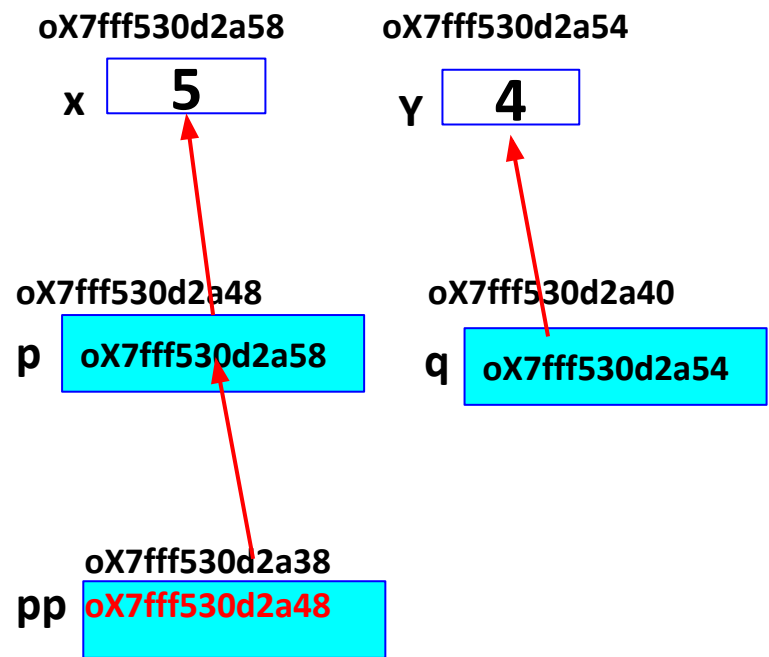
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap  
x = 5 y = 4  
  
Tercer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

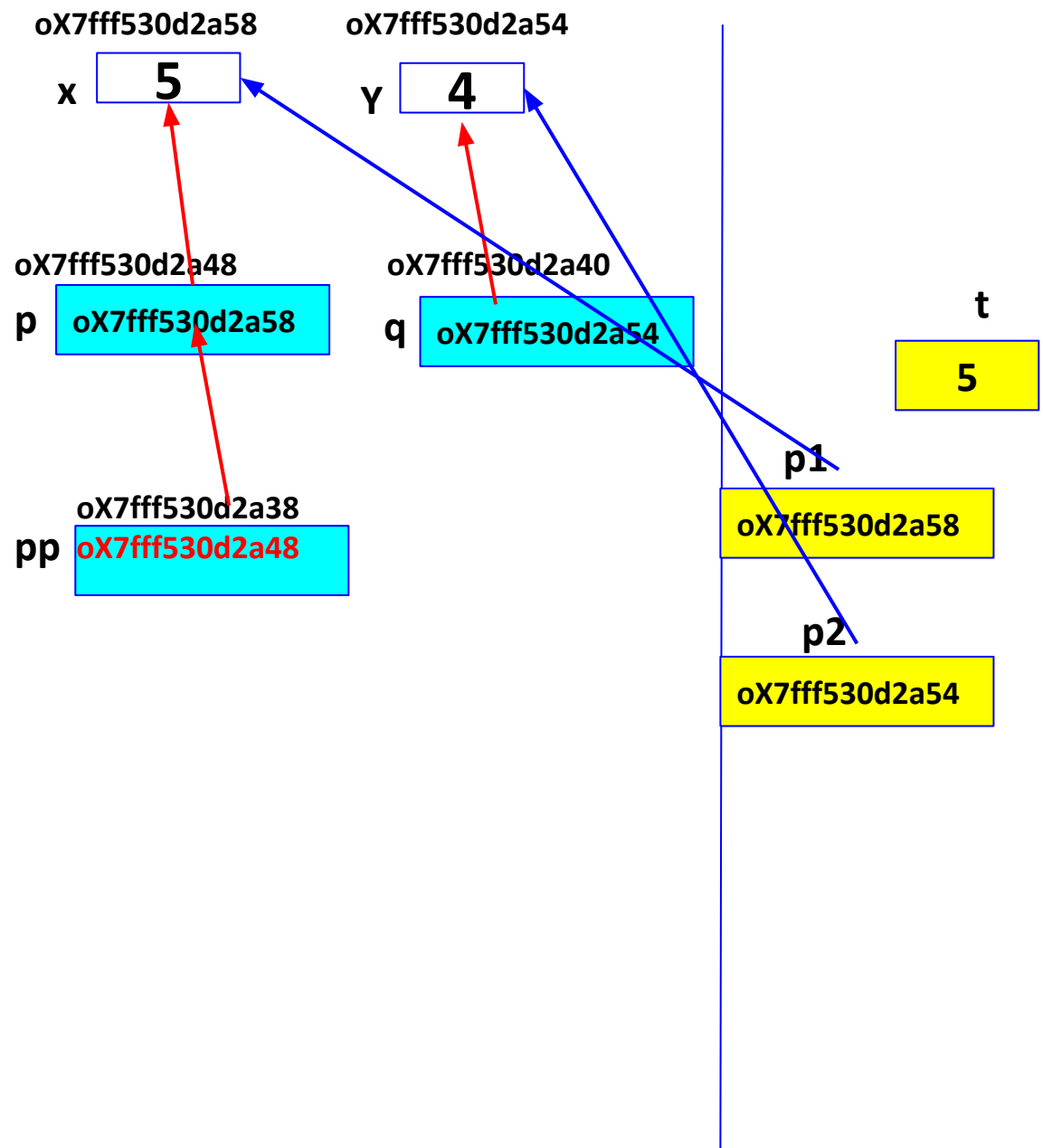
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap

x =4 y = 5

Segundo swap

x = 5 y = 4

Tercer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

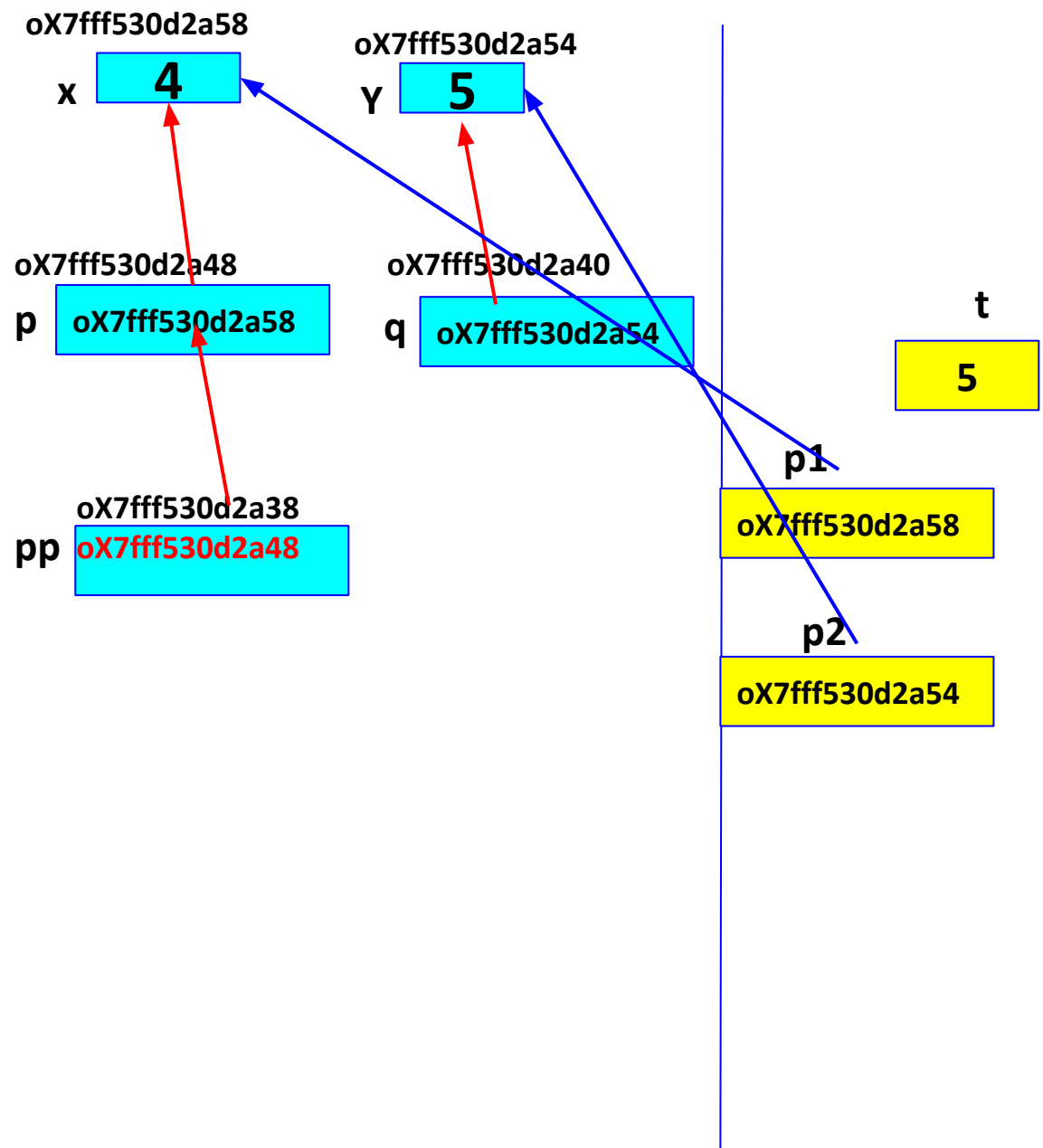
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:  
Suma = 9  
Primer swap  
x =4 y = 5  
  
Segundo swap  
x = 5 y = 4  
  
Tercer swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

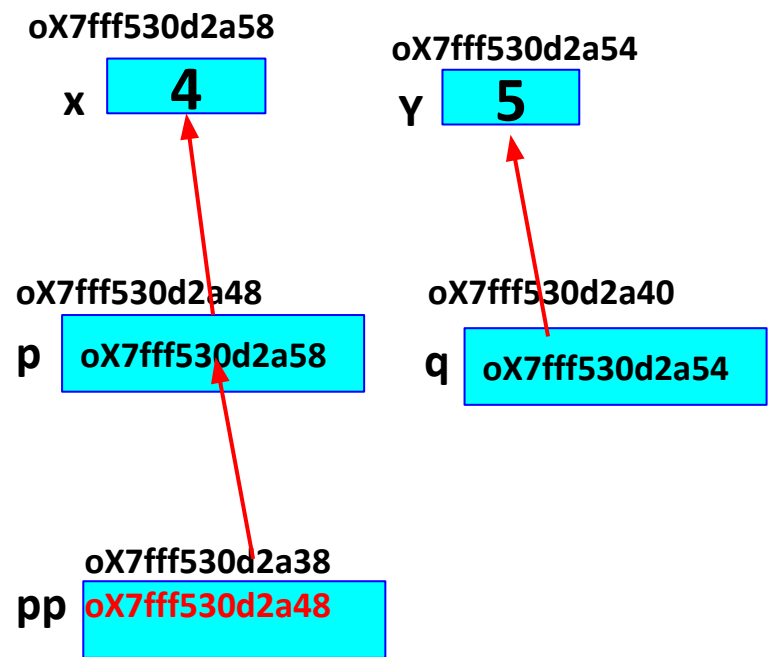
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5



```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

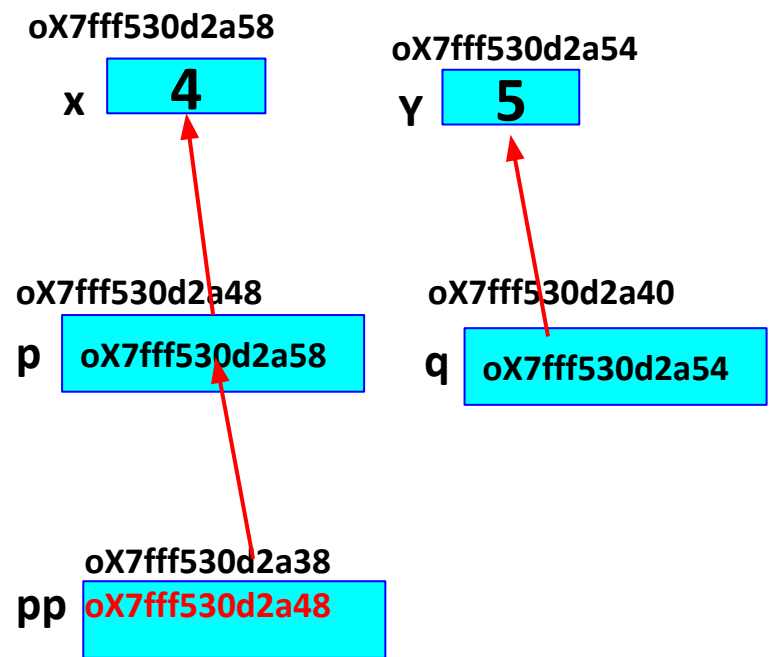
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

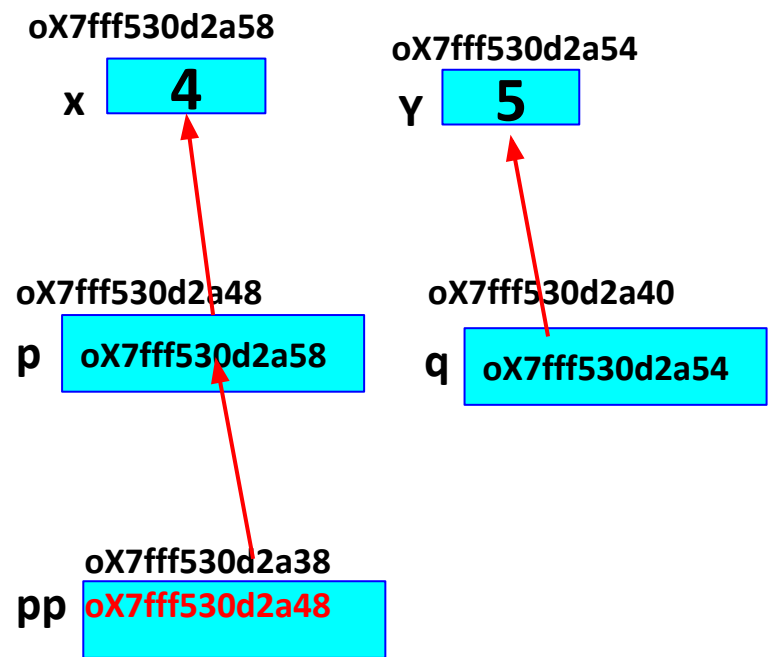
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5

Cuarto swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

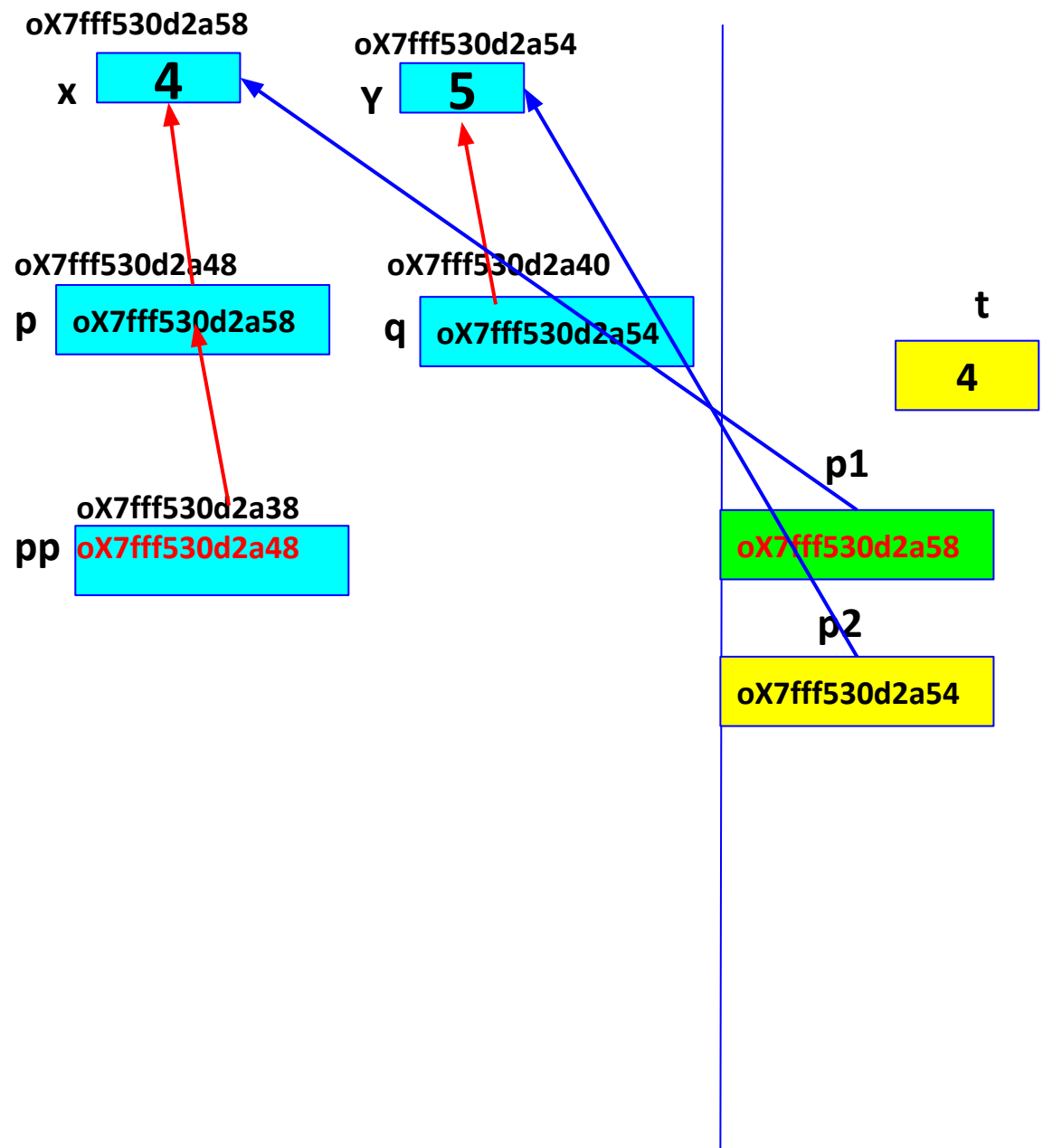
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5

Cuarto swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

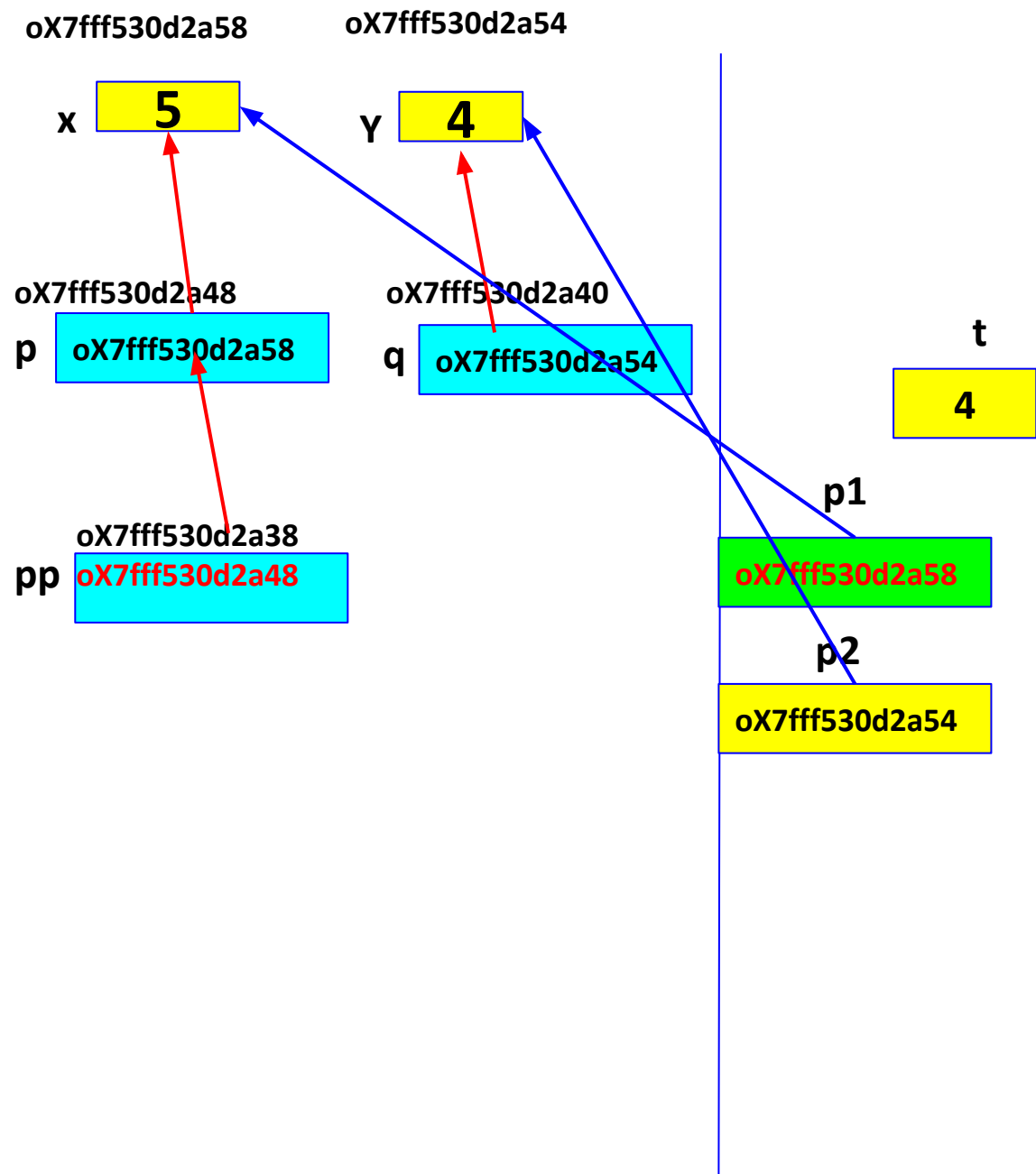
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap

x =4 y = 5

Segundo swap

x = 5 y = 4

Tercer swap

x = 4 y = 5

Cuarto swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

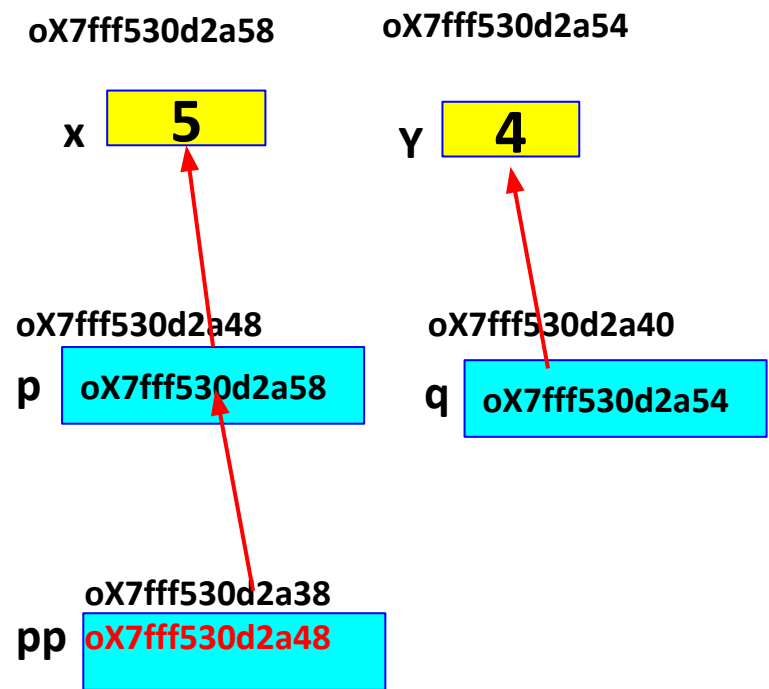
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5

Cuarto swap  
x = 5 y = 4

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

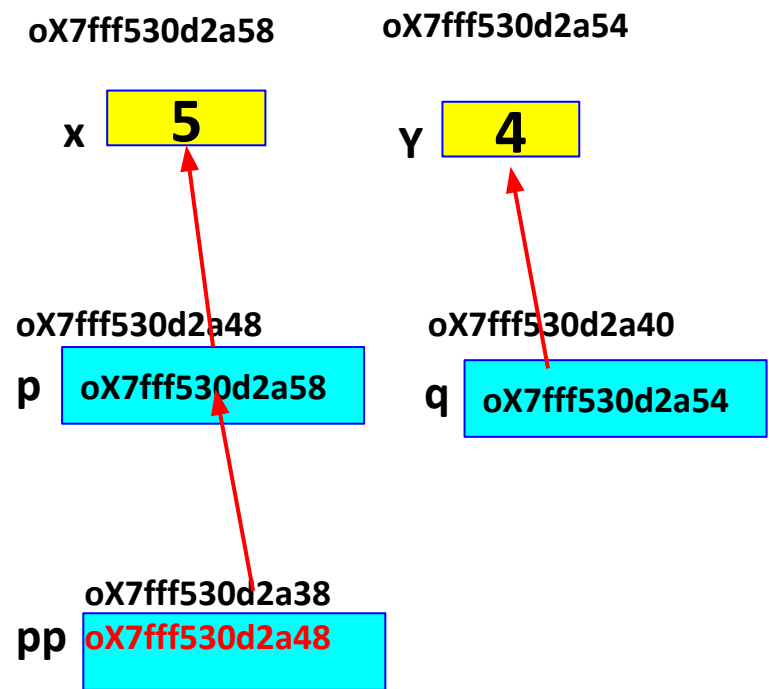
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}

Pantalla:
Suma = 9
Primer swap
x =4 y = 5
Segundo swap
x = 5 y = 4
Tercer swap
x = 4 y = 5

Cuarto swap
x = 5 y = 4

Quinto swap
```

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

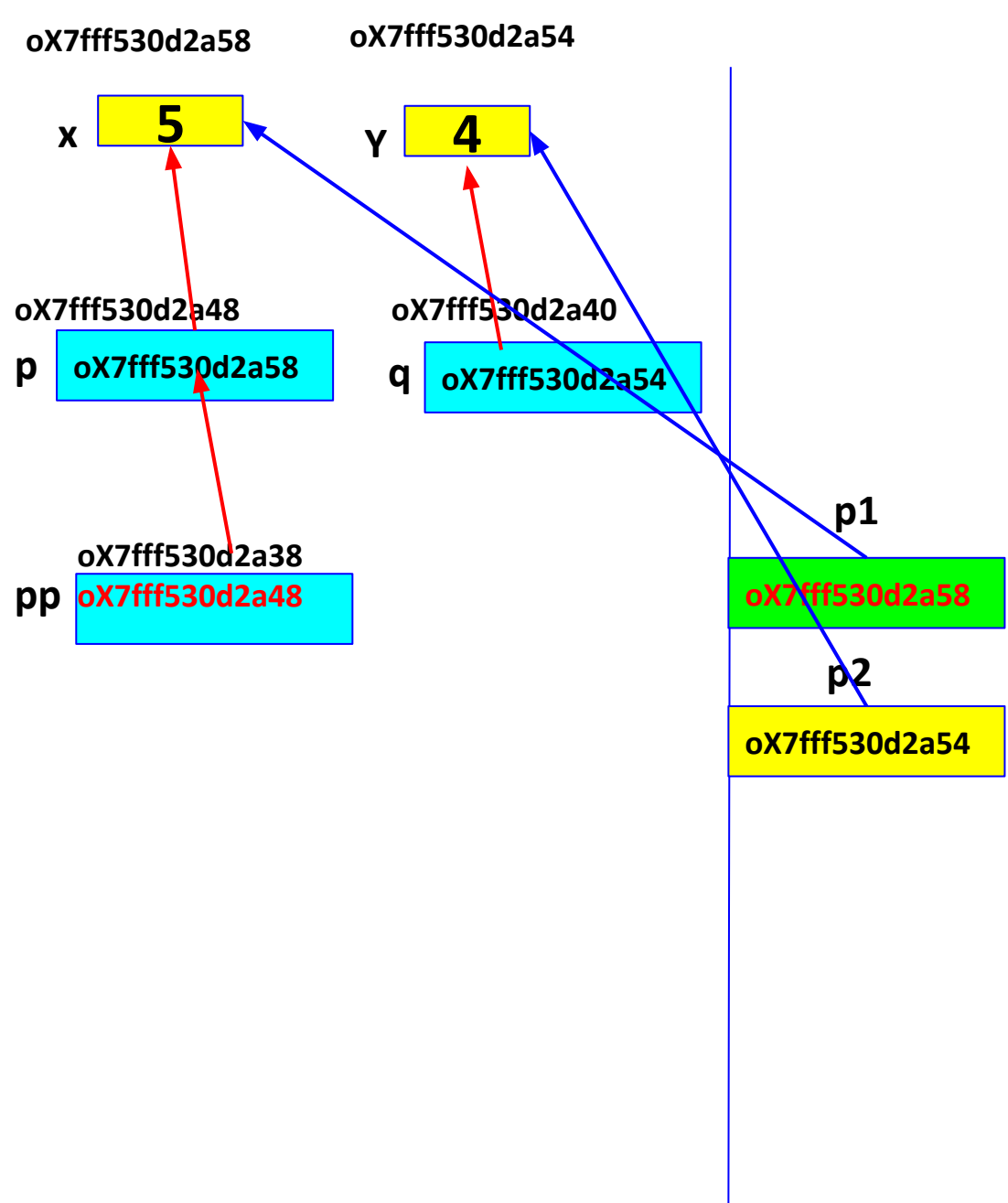
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{ int temporal = a;
  a=b;
  b=temporal;
}

void swap(int *p1, int *p2)
{ int t = *p1;
  *p1 = *p2;
  *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap  
x =4 y = 5

Segundo swap  
x = 5 y = 4

Tercer swap  
x = 4 y = 5

Cuarto swap  
x = 5 y = 4

Quinto swap

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

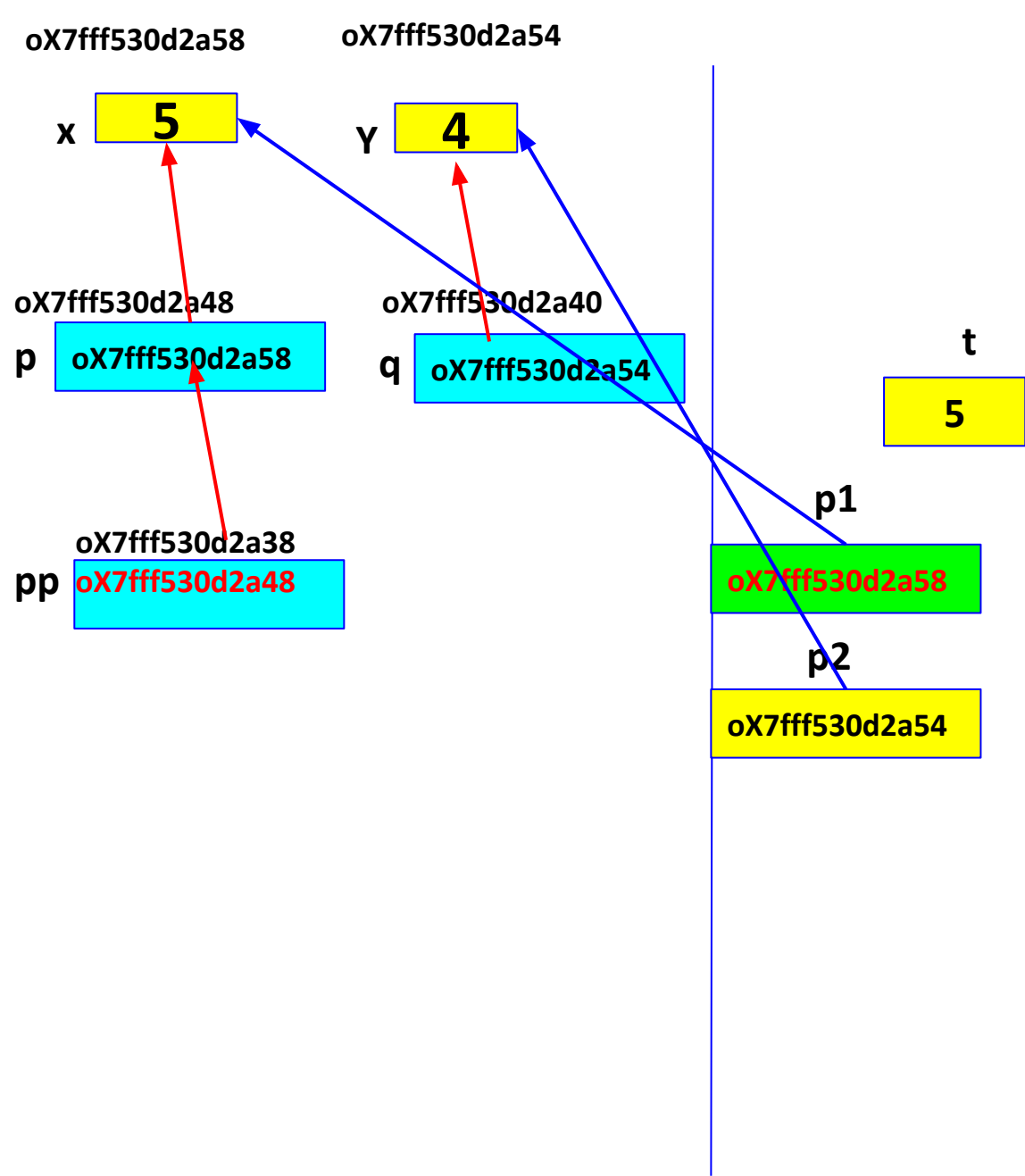
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}

Pantalla:
Suma = 9
Primer swap
x =4 y = 5
Segundo swap
x = 5 y = 4
Tercer swap
x = 4 y = 5

Cuarto swap
x = 5 y = 4

Quinto swap
```



```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x =" << x << " y =" << y << "\n\n";

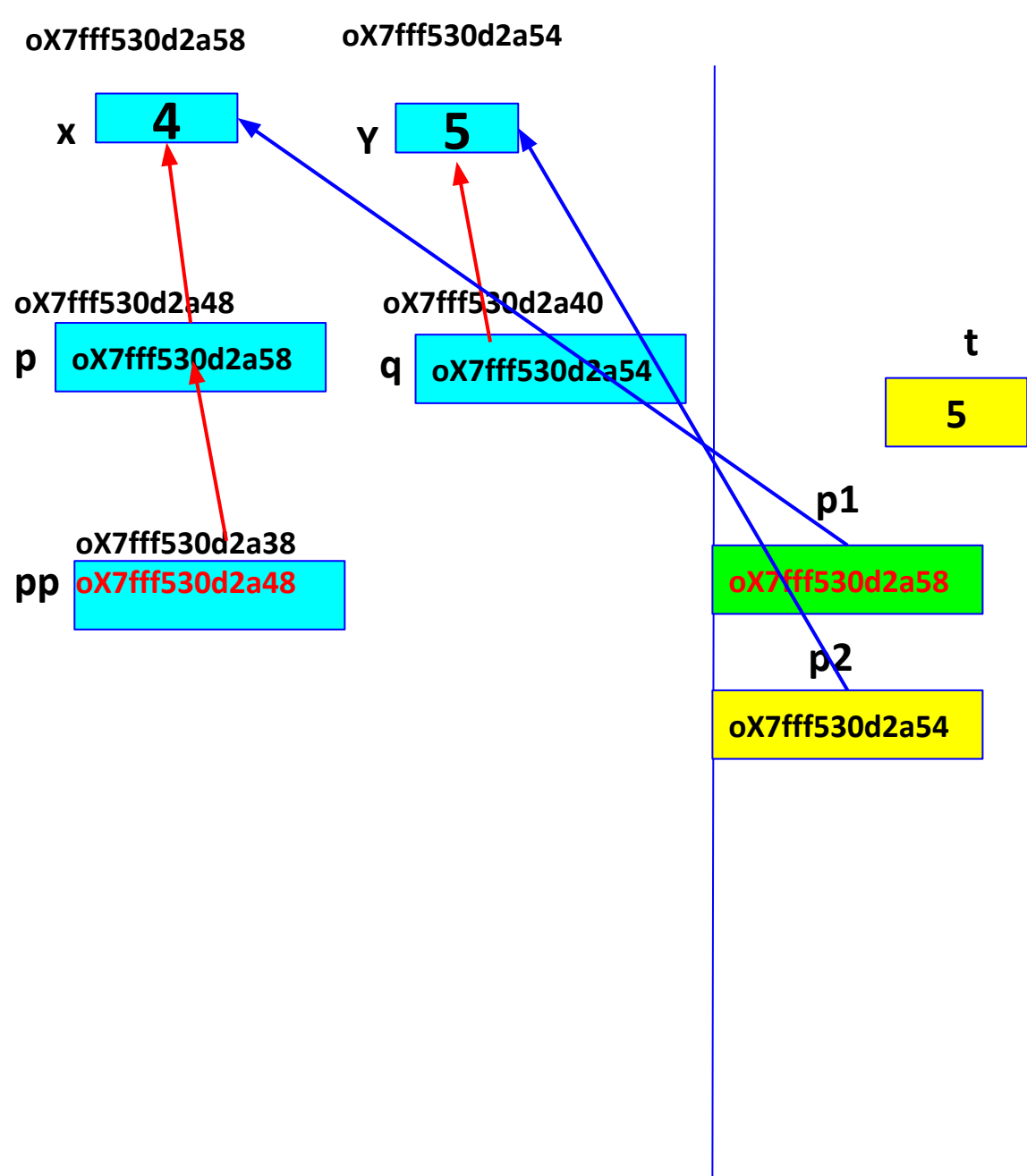
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x =" << x << " y =" << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x =" << x << " y =" << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}

Pantalla:
Suma = 9
Primer swap
x =4 y = 5
Segundo swap
x = 5 y = 4
Tercer swap
x = 4 y = 5

Cuarto swap
x = 5 y = 4

Quinto swap
```

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

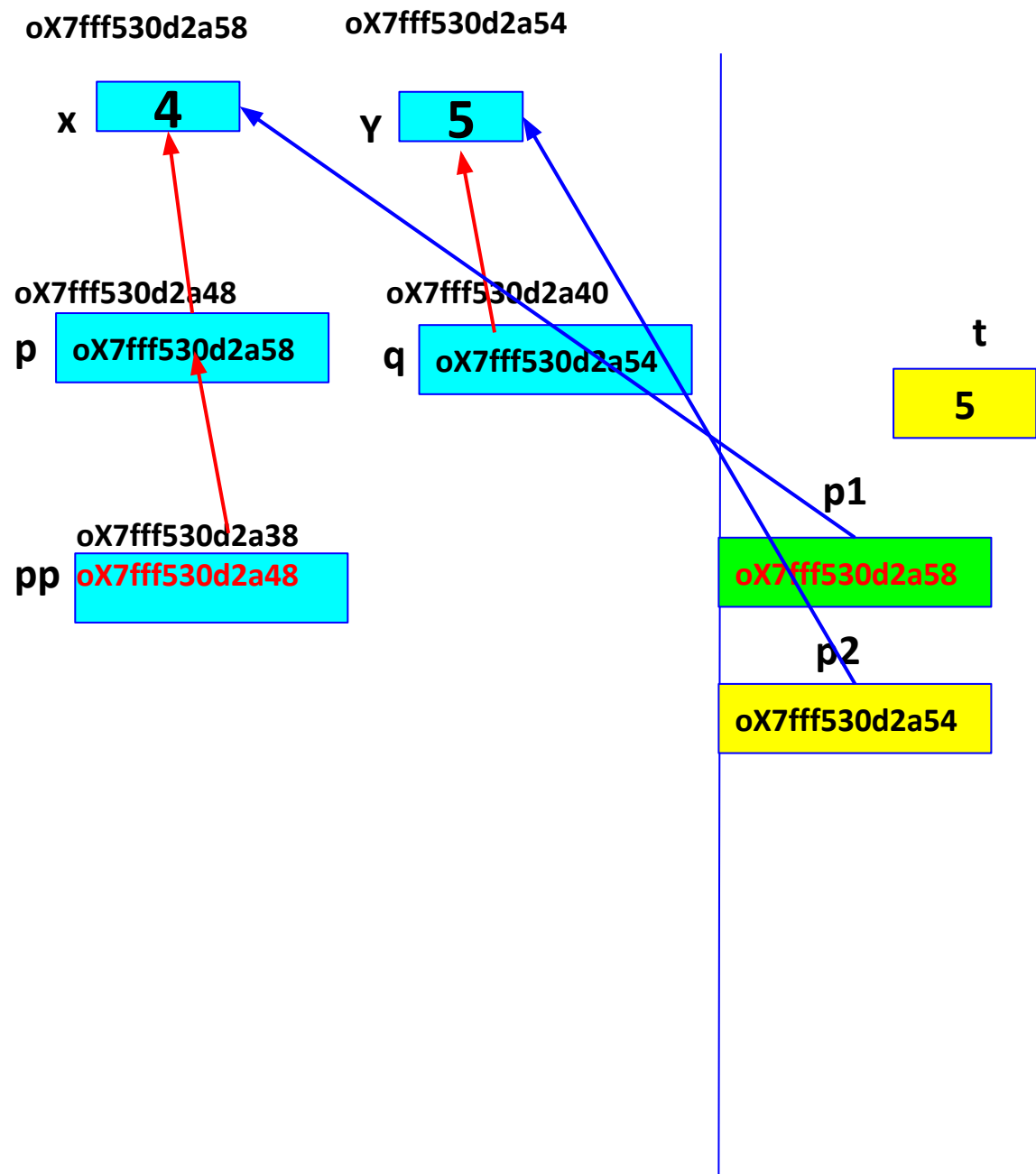
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}

Pantalla:
Suma = 9
Primer swap
x =4 y = 5
Segundo swap
x = 5 y = 4
Tercer swap
x = 4 y = 5

Cuarto swap
x = 5 y = 4

Quinto swap
x = 4 y = 5
```

```
int main()
{
    int x=5, y=4, *p, *q, **pp;
    p = &x; q = &y; pp = &p;

    cout <<"Suma = " << fSuma(x,y) << "\n\n";

    cout <<"Primer swap\n";
    swap(x,y);
    cout << "x = " << x << " y = " << y << "\n\n";

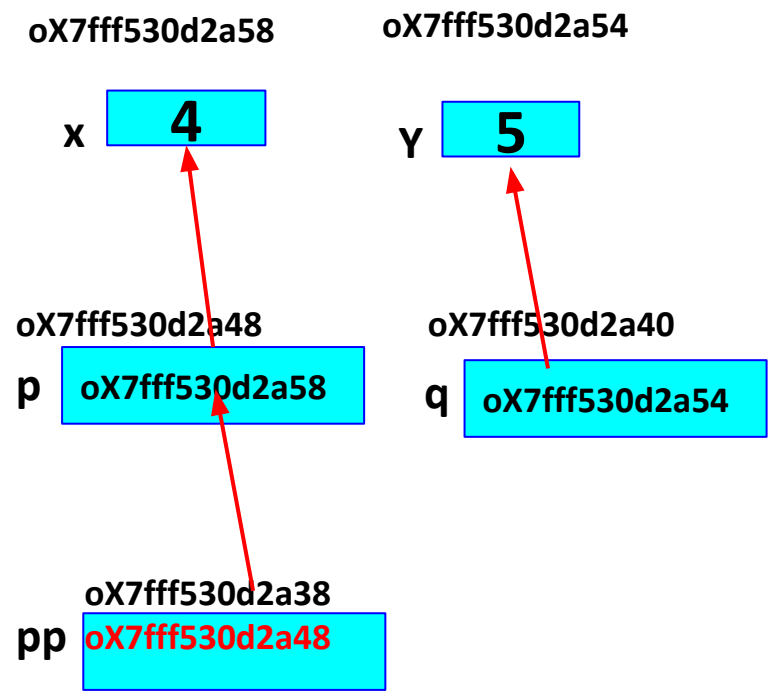
    cout <<"Segundo swap\n";
    swap(&x, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Tercer swap\n";
    swap(p, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Cuarto swap\n";
    swap(*pp, &y);
    cout << "x = " << x << " y = " << y << "\n\n";

    cout <<"Quinto swap\n";
    swap(*pp, q);
    cout << "x = " << x << " y = " << y << "\n\n";

    return 0;
}
```



```
int fSuma(int a, int b)
{
    return (a+b);
}

void swap(int &a, int &b)
{
    int temporal = a;
    a=b;
    b=temporal;
}

void swap(int *p1, int *p2)
{
    int t = *p1;
    *p1 = *p2;
    *p2 = t;
}
```

Pantalla:

Suma = 9

Primer swap

x =4 y = 5

Segundo swap

x = 5 y = 4

Tercer swap

x = 4 y = 5

Cuarto swap

x = 5 y = 4

Quinto swap

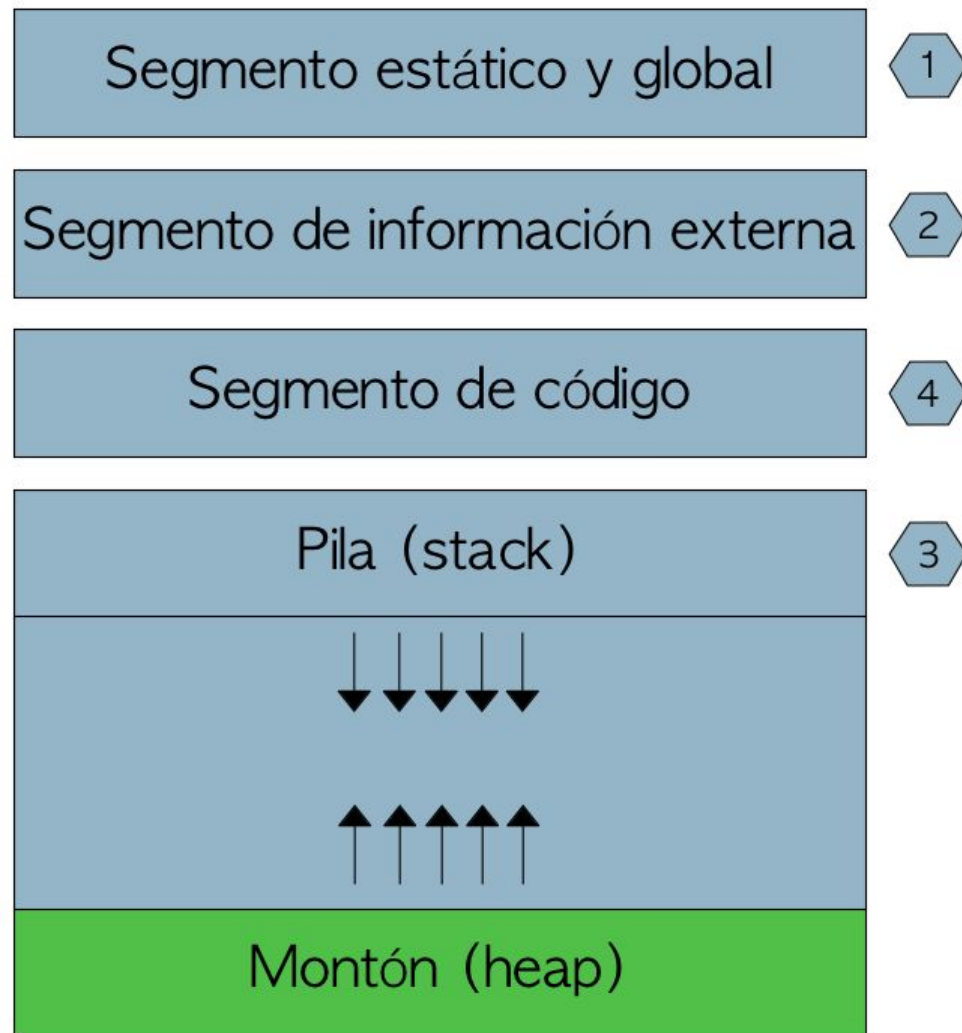
x = 4 y = 5

# 3.2

## Unidad 3: Memoria dinámica (arreglos)

UTEC

# Programa de C++ en la memoria primaria



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

```
int varGlobal = 20;
```

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

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

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

# Operadores para asignar y liberar memoria dinámica

**new**    asignar memoria

**delete**    libera memoria asignada por new

# **new** asigna memoria dinámicamente

```
int *pi = new int;    // p apunta a un espacio asignado dinámicamente
```

**new** construye un objeto de tipo `int` en un espacio libre de memoria y retorna el puntero a ese objeto. El objeto no se inicializa.

```
string *ps= new string;    // string vacio
```

```
int *pi1 = new int;        // pi1 puntero a un int, el entero no se ha inicializado
```

```
int *pi2 = new int(1024);   // pi2 apunta a un objeto int que tiene el valor 1024
```

```
int *pi3 = new int();       // pi3 apunta a un objeto int que tiene el valor cero
```

# Liberando memoria dinámica:

Para prevenir que la memoria se sature, se debe eliminar el espacio asignado dinámicamente una vez que se haya terminado de utilizar.

```
delete p; // libera el espacio  
          // p debe ser un puntero a memoria asignada  
          dinámicamente
```



# Acceso al Heap

```
int* ptrMonton = nullptr;
```

```
int* ptrVar = nullptr;
```

....

```
int var = 20;
```

```
ptrVar = &var;
```

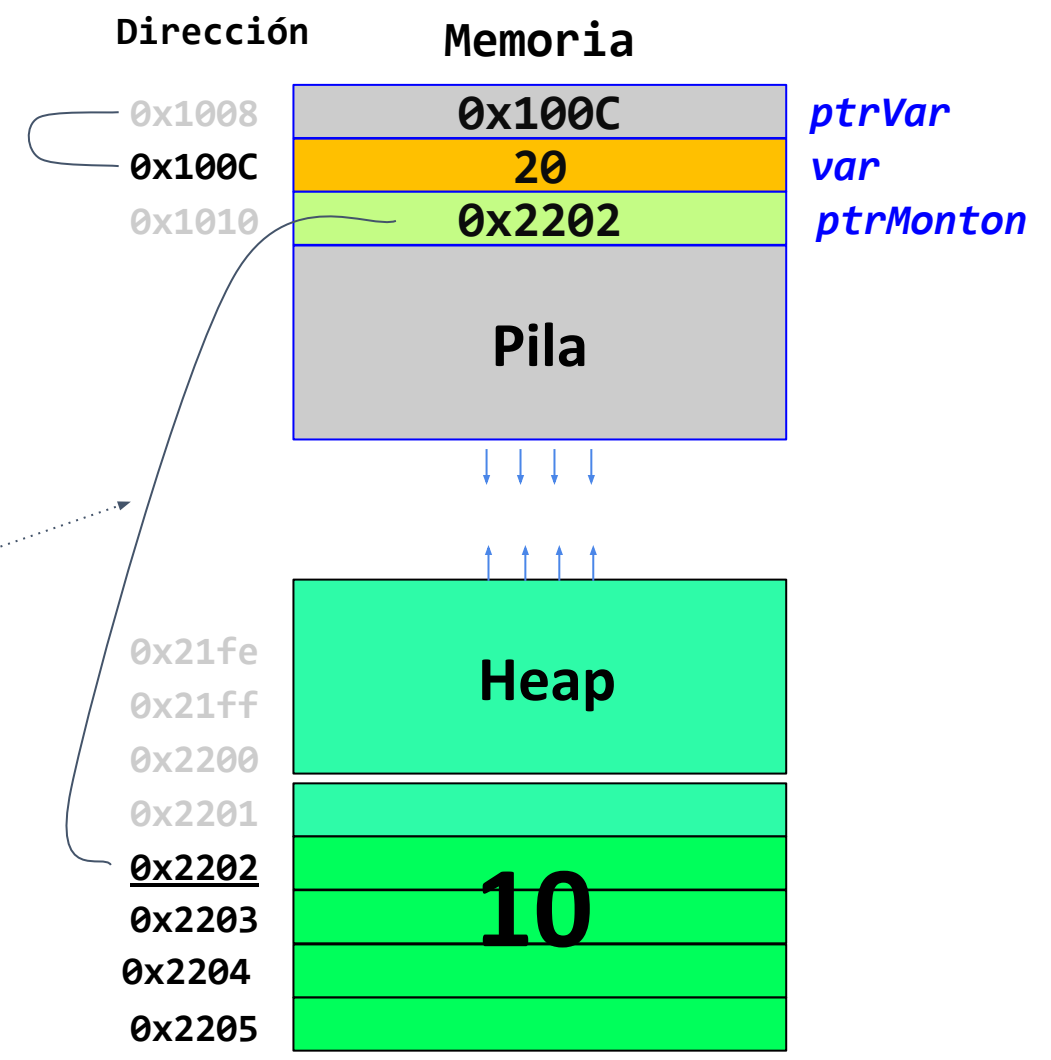
....

```
ptrMonton = new int;
```

```
*ptrMonton = 10;
```

....

```
delete ptrMonton;
```



# Ejemplo 1:

Desarrolla un programa que permita leer dos números de tipo double, se almacenen utilizando memoria dinámica y luego halle la suma, la diferencia y el producto de estos números.

```
#include <iostream>
using namespace std;
typedef double TNumero;

int main()
{ TNumero *pnumero1= nullptr, *pnumero2= nullptr;
  pnumero1 = new double;
  pnumero2 = new double;

  cout << "Numero 1 : ";
  cin >> *pnumero1; //-- se lee el número en el sitio apuntado por el puntero
  cout << "Numero 2 : ";
  cin >> *pnumero2;
  cout << "\n";
  cout << "La Suma es      : " << *pnumero1 + *pnumero2 << "\n";
  cout << "La Diferencia es : " << *pnumero1 - *pnumero2 << "\n";
  cout << "El Producto es  : " << *pnumero1 * *pnumero2 << "\n";
  delete pnumero1;
  delete pnumero2;
  pnumero1= nullptr;
  pnumero2= nullptr;
  return 0;
}
```

```
Numero 1 : 5
Numero 2  : 3

La Suma es : 8
La Diferencia es : 2
El Producto es : 15
```

# Importante:

En el programa anterior, un double utiliza 8 bytes para ser almacenado, y un puntero a double también 8 bytes

- ¿ Cuántos bytes de memoria se necesita para almacenar todas las variables definidas en la función main?
- ¿ Se usa espacio de la pila?
- ¿ Se usa espacio del heap?

# Ejemplo 02: objetos creados dinámicamente existen hasta que sean liberados de la memoria:

Veamos funciones que retornan memoria dinámica.  
Es responsabilidad del programa liberar la memoria.

```
double* f1(double arg){  
    return new double(arg);  
}  
  
void f2(double* arg)  
{  
    *arg = 3.222;  
    double* p=f1(*arg);  
    // se debe eliminar p  
    delete p;  
}
```

# Resumen

- El **puntero** es un tipo de datos que permite almacenar la memoria de una variable o una sección de la memoria.
- El **puntero** cuenta con un operador especial conocido como **dereferencia** que permite acceder a los valores almacenados en la dirección almacenada en el puntero.
- Los **punteros** pueden ser usados para acceder **por referencia** a una variable o parámetro y es la única forma de acceder a la memoria dinámica.

# Arreglos

- Arreglos Nativos Estáticos (Built-in)
- **Arreglos Dinámicos**

# Arreglos estáticos

Un **array** es una estructura de datos, que permite almacenar elementos **del mismo tipo**, los que se acceden por su posición.

Los arrays pueden ser **estáticos** y **dinámicos**.

Un **array estático** tiene un tamaño definido y su tamaño no puede variar en todo el programa. Para definir un array estático, se tiene que saber su tamaño previamente.

Un **array dinámico** es más flexible y su tamaño se puede decidir cuando se ejecuta el programa.



# Array estático:

```
typedef int type_int ;  
type_int A[10]={9,45,3,7,10,25,14,15,100,120};
```

Una vista lógica se muestra a continuación :

			0	1	2	3	4	5	6	7	8	9	
A			9	45	3	7	10	25	14	15	100	120	

Para hacer referencia al elemento se usa el índice:

```
cout << A[3];  //---- imprime el 7  
cout <<A[1];  //-----imprime el 45
```

# Definición de arrays:

```
int arr[10];           // es un array de 10 elementos.  
int a2[]={0,1,2};      //-- es un array de 3 elementos  
int a3[5] = {0,1,2};    //-- equivale a int a[]={0,1,2,0,0};  
  
string a4[3]={"hi", "bye"}; //--- equivale a: string a4[]{"hi", "bye", ""};  
int a5[2]={0,1,2};      //--- es un error: muchos inicializadores.
```

```
const unsigned sz=3;  
int a1[sz]={0,1,2};      //-- define un array de 3 elementos  
                        //-- con valores 0,1,2
```

```
int w=3;  
float F[w] = {1,2,3};    //-- error: Solo se conoce el valor de w  
                        //-- en tiempo de ejecución, no de compilación
```

# Definición de arrays:

```
typedef int type_int;  
typedef string type_str;
```

```
type_int arr[10];           // es un array de 10 elementos.  
type_int a2[] = {0, 1, 2}; //--- es un array de 3 elementos  
type_int a3[5] = {0, 1, 2}; //--- equivale a int a[]={0,1,2,0,0};  
type_str a4[3] = {"hi", "bye"}; //--- equivale a: string a4[]{"hi", "bye", ""};  
type_int a5[2] = {0, 1, 2}; //--- es un error: muchos inicializadores.
```

```
const unsigned sz = 3;  
int a1[sz] = {0, 1, 2}; //--- define un array de 3 elementos  
                        //--- con valores 0, 1, 2
```

```
int w=3;  
float F[w] = {1, 2, 3}; //--- error: Solo se conoce el valor de  
                        //--- w en tiempo de ejecución, no de compilación
```

Los elementos del array se almacenan de manera contigua en la memoria del computador.

```
typedef int type_int;  
type_int A[5]={0, 10, 20, 30, 40};
```

**A** representa la dirección de memoria donde se encuentra el primer elemento del array.

<b>A</b>	
0x7fe1059025b0	<b>0</b>
0x7fe1059025b1	
0x7fe1059025b2	
0x7fe1059025b3	
0x7fe1059025b4	<b>10</b>
0x7fe1059025b5	
0x7fe1059025b6	
0x7fe1059025b7	
0x7fe1059025b8	<b>20</b>
0x7fe1059025b9	
0x7fe1059025ba	
0x7fe1059025bb	
0x7fe1059025bc	<b>30</b>
0x7fe1059025bd	
0x7fe1059025be	
0x7fe1059025bf	
0x7fe1059025c0	<b>40</b>
0x7fe1059025c1	
0x7fe1059025c2	
0x7fe1059025c3	

# Los elementos de un array, se imprimen recorriendo uno a uno los elementos del array (ejemplo 03)

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

typedef int type_int;

int main()
{
    const int tam = 7;
    type_int A[tam]={ 71,72,73,74,75,76,77};

    //--- se recorre el array y se imprimen los datos
    for(int i=0; i<tam; i++)
        cout << "A[" << i << "]= " << A[i] << "\n";

    //-- se imprimen las direcciones donde se encuentra cada elemento
    cout << "\nInicio del array " << A << "\n";

    cout << "\nImprimimos las direcciones de cada casillero del
        Arreglo A \n";
    for(int i=0; i<tam; i++)
        cout << "&A[" << i << "]= " << &A[i] << " Guarda el " << A[i] << "\n";

    return 0;
}
```

## Se imprime:

```
A[0]=71
A[1]=72
A[2]=73
A[3]=74
A[4]=75
A[5]=76
A[6]=77
```

Inicio del array 0x7fff52a26a00

Imprimimos las direcciones de cada casillero del Arreglo A

&A[0]= 0x7fff52a26a00	Guarda el 71
&A[1]= 0x7fff52a26a04	Guarda el 72
&A[2]= 0x7fff52a26a08	Guarda el 73
&A[3]= 0x7fff52a26a0c	Guarda el 74
&A[4]= 0x7fff52a26a10	Guarda el 75
&A[5]= 0x7fff52a26a14	Guarda el 76
&A[6]= 0x7fff52a26a18	Guarda el 77

# Arreglos dinámicos

# Array dinámicos (ejemplo 04):

```
int *pia = new int[10];      // bloque de 10 unidades de int
int *pia2 = new int[10] ();  // bloque de 10 int
                               // inicializados con cero
int *pia3 = new int[10] {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

Para liberar el espacio de memoria:

```
delete [] pia;
```

# Array dinámicos (ejemplo 04):

```
typedef int type_t;
typedef unsigned int type_h;
type_h size = 10;
type_t* pia = new type_t[size];      // bloque de 10 unidades de int
type_t* pia2 = new type_t[size] ();  // bloque de 10 int
                                         // inicializados con cero
type_t* pia3 = new type_t[size] {0,1,2,3,4,5,6,7,8,9};

// Para liberar el espacio de memoria:
delete [] pia;
```



# Se crea un array de 5 elementos en el heap:

```
int *p = nullptr;
```

```
p = new int[5];
```

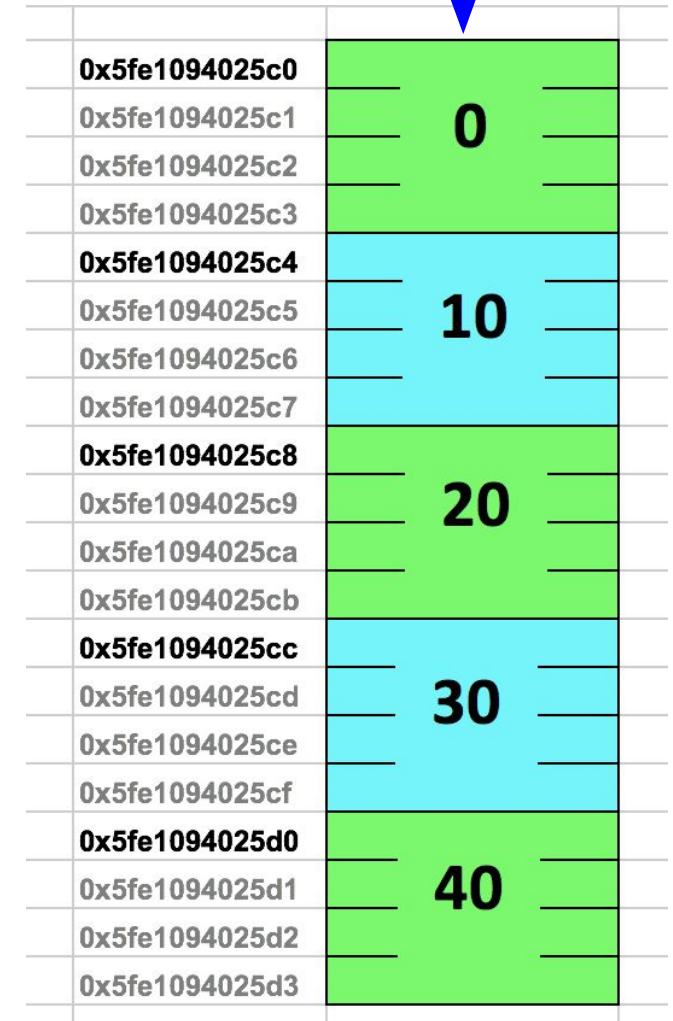
```
for(size_t i = 0; i < 5 ;i++)
```

```
    p[i] = i*10;
```

0xb12345678abc

*p*

0x5fe1094025c0



0x5fe1094025c0	0
0x5fe1094025c1	
0x5fe1094025c2	
0x5fe1094025c3	
0x5fe1094025c4	10
0x5fe1094025c5	
0x5fe1094025c6	
0x5fe1094025c7	
0x5fe1094025c8	20
0x5fe1094025c9	
0x5fe1094025ca	
0x5fe1094025cb	
0x5fe1094025cc	30
0x5fe1094025cd	
0x5fe1094025ce	
0x5fe1094025cf	
0x5fe1094025d0	40
0x5fe1094025d1	
0x5fe1094025d2	
0x5fe1094025d3	

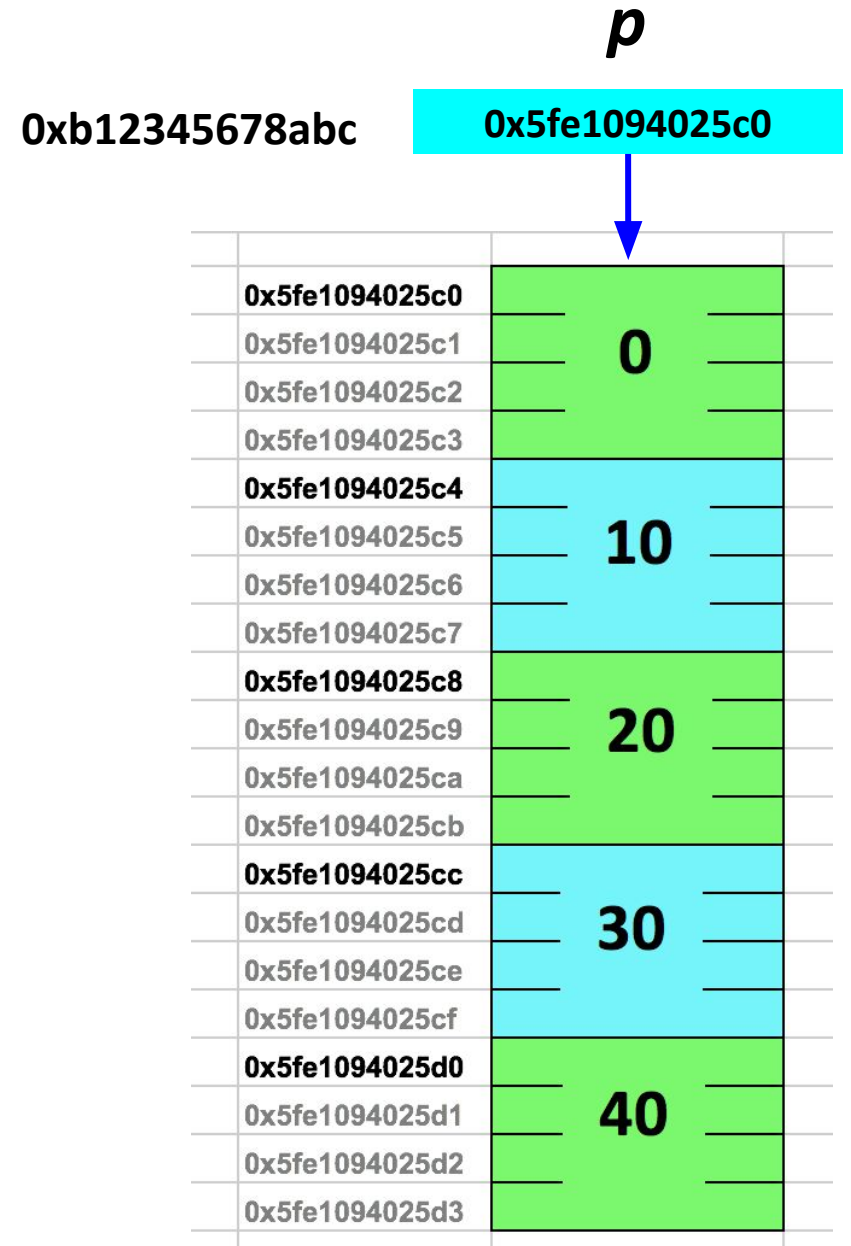
# Se crea un array de 5 elementos en el heap:

```
typedef int type_t;  
typedef unsigned int type_h;
```

```
type_t *p = nullptr;  
type_h size = 5;
```

```
p = new type_t[size];  
for(type_h i = 0; i < size ;i++)  
    p[i] = i*10;
```

```
delete [] p;
```



# Ejemplo 05:

Realice un programa que permita leer como dato un número que representa la cantidad de elementos que tendrá un array dinámico. Luego realice lo siguiente:

- Crear el array, llenarlo con números aleatorios entre 0 y 999.
- Imprimir el array
- Generar a partir de ese array dos nuevos arrays dinámicos, el primero con los múltiplos de 5 y el segundo con los múltiplos de 7 que tenga el primer array.

## Ejemplo:

Numero de elementos : 21

Array

987 184 343 795 671 916 947 160 629 453 615 865 134 270 775 212 998 474 68 152 967

Multiplos de cinco

795 160 615 865 270 775

Multiplos de siete

987 343

# Solución 1 - Usando array dinámicos

En la solución se utilizan los siguientes archivos.

Main.cpp  
UFunciones.h  
UFunciones.cpp

## main.cpp

```
#include <iostream>
#include "UFunciones.h"
using namespace std;

int main()
{
    size_t numDatos, c5=0, c7=0;
    int *pA,*pM5, *pM7;

    srand(time(nullptr));
    cout << "Numero de elementos : ";
    cin >> numDatos;
    pA = PideEspacio(numDatos);
    GeneraDatosAlAzar(pA,numDatos);
    cout << "\nArray\n";
    Imprimir(pA,numDatos);
    c5=ContarMultiplos(pA,numDatos,5);
    c7=ContarMultiplos(pA,numDatos,7);
    pM5 = PideEspacio(c5);
    pM7 = PideEspacio(c7);
```

```
LlenaMultiplos(pA,numDatos,pM5,5);
LlenaMultiplos(pA,numDatos,pM7,7);
cout << "\n\nMultiplos de cinco\n";
Imprimir(pM5,c5);
cout << "\n\nMultiplos de siete\n";
Imprimir(pM7,c7);

Eliminar(pA);
Eliminar(pM5);
Eliminar(pM7);
return 0;
}
```

## UFunciones.h

```
#ifndef MULTIPLOS_VERSION1_UFUNCIONES_H
#define MULTIPLOS_VERSION1_UFUNCIONES_H

#include <iostream>
#include <cstdint>
#include <cstdlib>
#include <iomanip>
using namespace std;

int * PideEspacio(size_t numDatos);
void  GeneraDatosAlAzar(int *pA,size_t numDatos);
void  Imprimir(int *pA,size_t numDatos);
size_t ContarMultiplos(int *pA,size_t numDatos, int mul);
void  LlenaMultiplos(int *pA,size_t numDatos,int * pMul, int mul);
void  Eliminar(int *& pA);

#endif //MULTIPLOS_VERSION1_UFUNCIONES_H
```

```
#include "UFunciones.h"
```

```
int * PideEspacio(size_t numDatos)
{
    //-----
    int *pInicio = new int[numDatos];
    return pInicio;
}
```

```
void GeneraDatosAlAzar(int *pA, size_t numDatos)
{
    //-----
    for(size_t i=0; i<numDatos; i++)
        pA[i]= rand()%1000;
}
```

```
void Imprimir(int *pA, size_t numDatos)
{
    //-----
    for(size_t i=0; i<numDatos; i++)
        cout << setw(5) << pA[i];
}
```



```
size_t ContarMultiplos(int *pA,size_t numDatos, int mul)
{
//-----
size_t c=0;

for(size_t i=0; i<numDatos; i++)
    if(pA[i]%mul==0)
        c++;
return c;
}
```

```
void LlenaMultiplos(int *pA,size_t numDatos,int * pMul, int mul)
{
//-----
size_t c=0;

for(size_t i=0; i<numDatos; i++)
    if(pA[i]%mul==0)
        pMul[c++]=pA[i];
}
```

```
void Eliminar(int *& pA)
{
//-----
delete []pA;
pA= nullptr; //-- se lacra el puntero
}
```

## Solución 2 - Usando vector (tema semana 7)

## main.cpp

```
#include <iostream>
#include <random>
#include <iomanip>
#include "UFunciones.h"

using namespace std;

int main()
{ random_device rd; //--para generar números al azar
  unsigned long n=0;

  cout << "Numero de elementos : ";
  cin >> n;

  vector<int> a(n);
  auto x = a;
  for(auto & item:a)
    item = rd()%1000;
  Imprimir(a);

  vector<int> m5;
  for(auto item:a)
    if( item%5==0)
      m5.push_back(item);
```

```
cout << "\n\nMultiplos de 5 \n";
cout << "Hay " << m5.size() << "\n";
Imprimir(m5);

vector<int> m7;
for(auto item:a)
  if(item%7==0)
    m7.push_back(item);
cout << "\n\nMultiplos de 7 \n";
cout << "Hay " << m7.size() << "\n";
Imprimir(m7);
return 0;
}
```

## UFunciones.h

```
#ifndef MULTIPLOS_VERSION2_UFUNCIONES_H
#define MULTIPLOS_VERSION2_UFUNCIONES_H

#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

void Imprimir(vector<int> v);

#endif //MULTIPLOS_VERSION2_UFUNCIONES_H
```

## UFunciones.cpp

```
#include "UFunciones.h"

void Imprimir(vector<int> v)
{
    //-----
    for (auto item:v)
        cout << setw(5) << item;
}
```

# Ejemplo (ordenamiento):

El siguiente programa muestra el ordenamiento de 30 datos utilizando Quick Sort.

Los elementos del array están entre 1 y 20 inclusive.

Pantalla de salida:

```
12 11 7 15 6 4 16 19 15 14 4 4 18 19 15 14 17 11 18 11 1 5 18 5 1 14 19 19 14 13  
1 1 4 4 4 5 5 6 7 11 11 11 12 13 14 14 14 14 15 15 15 16 17 18 18 18 19 19 19 19
```

## main. cpp

```
#include "ordenar.h"

int main() {

    size n = 30;

    auto arreglo=generar_arreglo_aleatorio(n,1,20);
    mostrar_arreglo(arreglo, n);

    ordenar_arreglo(arreglo, n);
    mostrar_arreglo(arreglo, n);

    liberar_arreglo(arreglo);
    return 0;
}
```

## ordenar.h

```
#ifndef ORDENAR_H
#define ORDENAR_H

#include <random>

using number = int;
using size = int;
using Void = void;
using device = std::random_device;
using distribution = std::uniform_int_distribution<number>;

auto randint(number first, number last) -> number;
number* generar_arreglo_aleatorio(size n, number first,
                                   number last);
Void liberar_arreglo(number* arreglo);
Void mostrar_arreglo(number* arreglo, size n);
Void swap(number& a, number& b);
size partition(number* arreglo, size n,
               size first, size last);
Void quicksort_(number* arreglo, size n,
                size first, size last);
Void ordenar_arreglo(number* arreglo, size n);

#endif
```

## ordenar.h

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

```
auto randint(number first, number last) -> number {
    distribution dist(first, last);
    device dev;
    return dist(dev);
}
```

```
number* generar_arreglo_aleatorio(size n, number first, number last) {
    number* arreglo = new number[n];
    for (size i = 0; i < n; ++i)
        arreglo[i] = randint(first, last);
    return arreglo;
}
```

```
Void liberar_arreglo(number* &arreglo) {
    delete[] arreglo;
    arreglo = nullptr;
}
```

```
Void mostrar_arreglo(number* arreglo, size n) {
    for (size i = 0; i < n; ++i)
        std::cout << arreglo[i] << " ";
    std::cout << std::endl;
}
```

```

void swap(number& a, number& b) {
    number temp = a;
    a = b;
    b = temp;
}

```

```

size partition(number* arreglo, size n, size first, size last) {
    auto pivot = arreglo[first];
    auto left = first + 1;
    auto right = last;
    while (true) {
        while (left <= right && arreglo[left] <= pivot) left++;
        while (right >= left && arreglo[right] >= pivot) right--;
        if (right < left) break;
        swap(arreglo[left], arreglo[right]);
    }
    swap(arreglo[first], arreglo[right]);
    return right;
}

```

```

Void quicksort_(number* arreglo, size n, size first, size last) {
    if (first < last) {
        auto split_point = partition(arreglo, n, first, last);
        quicksort_(arreglo, n, first, split_point - 1);
        quicksort_(arreglo, n, split_point + 1, last);
    }
}

```

```

Void ordenar_arreglo(number* arreglo, size n) {
    quicksort_(arreglo, n, 0, n - 1);
}

```



```

template<typename T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}

size partition(number* arreglo, size n, size first, size last) {
    auto pivot = arreglo[first];
    auto left = first + 1;
    auto right = last;
    while (true) {
        while (left <= right && arreglo[left] <= pivot) left++;
        while (right >= left && arreglo[right] >= pivot) right--;
        if (right < left) break;
        swap(arreglo[left], arreglo[right]);
    }
    swap(arreglo[first], arreglo[right]);
    return right;
}

Void quicksort_(number* arreglo, size n, size first, size last) {
    if (first < last) {
        auto split_point = partition(arreglo, n, first, last);
        quicksort_(arreglo, n, first, split_point - 1);
        quicksort_(arreglo, n, split_point + 1, last);
    }
}

Void ordenar_arreglo(number* arreglo, size n) {
    quicksort_(arreglo, n, 0, n - 1);
}

```



# 3.3

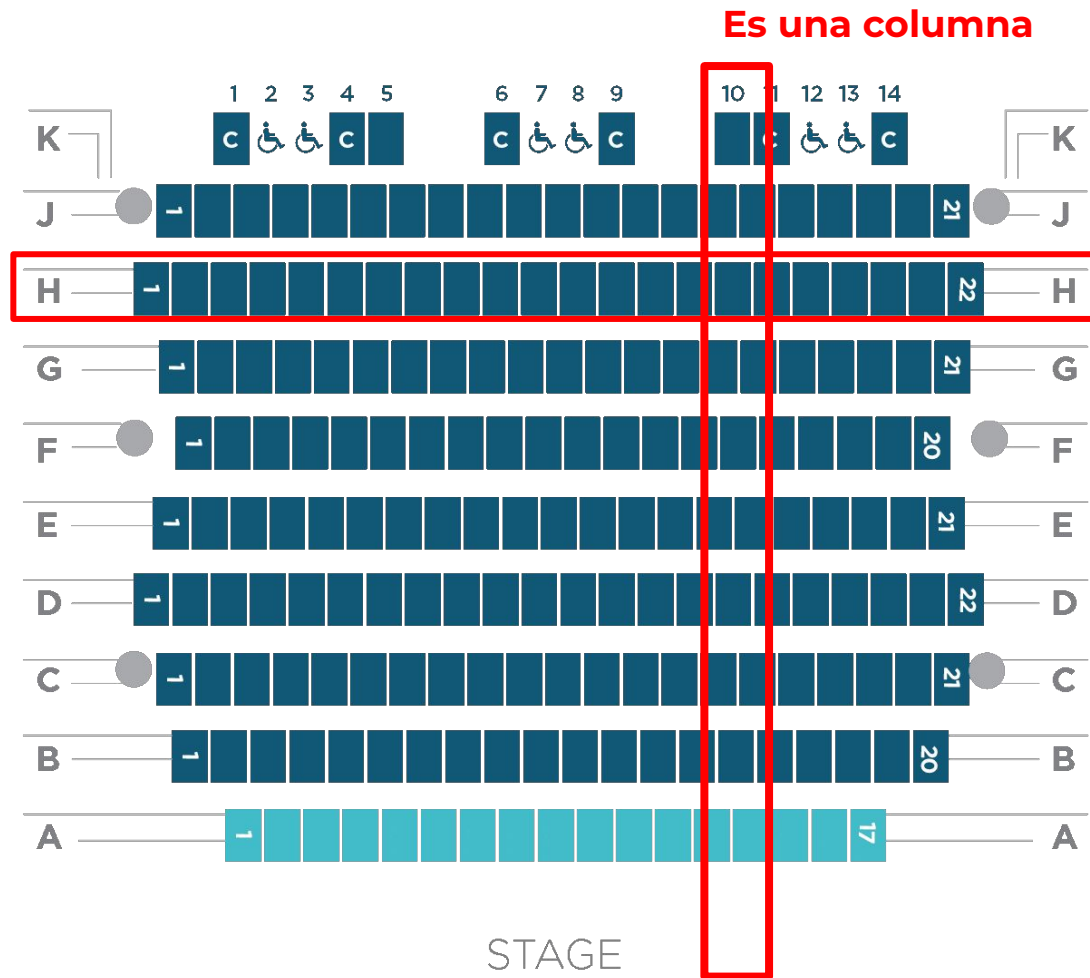
Unidad 3:  
Matrices estáticas (arreglos de  
arreglos)  
Matrices dinámicas

UTEC

# Arrays de arrays (Matrices)

---

Es una  
fila



```
typedef char type_char;  
type_char cine[10][14];
```

Lo que solía denominarse arrays multidimensionales son actualmente **arrays de arrays**.

Declaración/definición de un array de arrays:

```
int ia[3][4]; // array de tamaño 3, cada elemento es un array de 4  
enteros
```

```
int arr[10][20][30] = {0};  
// array de tamaño 10, cada elemento es un array de tamaño 20,  
que a su vez, es un array de 30 enteros  
// elementos inicializados en 0
```

usando typedef:

```
typedef int type_M1;
```

```
unsigned const int FIL = 3, COLU= 4;
```

```
type_M1 a[FIL ][COLU]  // array de tamaño 3, cada elemento es un array de 4  
enteros
```

```
typedef int type_M2;
```

```
unsigned const int TAM1 = 10, TAM2 = 20, TAM3 = 30;
```

```
type_M2 arr[TAM1][TAM2][TAM3] ={0};
```

```
//  array de tamaño 10, cada elemento es un array de tamaño 20, que a su vez, es  
un array de 30 enteros
```

```
// elementos inicializados en 0
```

# Inicialización de los elementos de un array de arrays:

```
int AA[3][4]={           // 3 elementos, cada elemento es array de tamaño 4
    {0, 1, 2, 3},        // valores iniciales de la fila 0
    {4, 5, 6, 7},        // valores iniciales de la fila 1
    {8,9,10,11}          // valores iniciales de la fila 2
};
```

		0	1	2	3								
AA	0	0	1	2	3								
	1	4	5	6	7								
	2	8	9	10	11								
AA		0	1	2	3	4	5	6	7	8	9	10	11

usando typedef:

```
typedef int type_int;  
unsigned const int FIL = 3, COL = 4;  
type_int AA[FIL][COL]={      // 3 elementos, cada elemento es array de tamaño 4  
    {0, 1, 2, 3},           // valores iniciales de la fila 0  
    {4, 5, 6, 7},           // valores iniciales de la fila 1  
    {8,9,10,11}             // valores iniciales de la fila 2  
};
```



```
typedef int type_ matrix1;  
unsigned const int FIL=3,COL=4;  
type_ matrix1 ia[FIL][COL] = { { 0 }, { 4 }, { 8 } };  
// se inicializa sólo el elemento 0 de cada fila
```

	0	1	2	3
0	0			
1	4			
2	8			

```
typedef int type_ matrix2;  
type_ matrix2 ix[FIL][COL] = { 0, 3, 6, 9 };  
//se inicializa sólo la fila 0, las demás filas se inicializan  
en cero.
```

	0	1	2	3
0	0	3	6	9
1	0	0	0	0
2	0	0	0	0

```
constexpr size_t rowCnt = 3, colCnt = 4;  
typedef int type_M1;  
type_M1 ia[rowCnt][colCnt]; // se declaran 12 elementos  
  
// por fila  
for( size_t i=0; i<rowCnt; i++)  
    //--- por columna en cada fila  
    for( size_t j=0; j<colCnt; j++)  
        ia[i][j] = i*colCnt + j;  
// se definen los elementos de la matriz
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

Ejemplo:

```
typedef int type_int;  
size_t FIL=4,COL=4;  
type_int Dosmil48[FIL][COL];
```

filas

columnas

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4

Orden de la matriz 4x4

Pasando un array multidimensional a funciones:

```
void funcion1( type_int matrix[][10], size_t rowSize, size_t colSize);
```

El parámetro es un puntero a un array de 10 enteros .

El array nunca se pasa por valor, lo que envía es el puntero al primer elemento

# ¿Cómo se asigna un dato a un casillero de la matriz?

```
typedef int type_M1;  
size_t FIL = 4, COL = 4;  
type_M1 Dosmil48[ FIL ] [ COL ];
```

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4

**Dosmil48[2][1] = 4;**

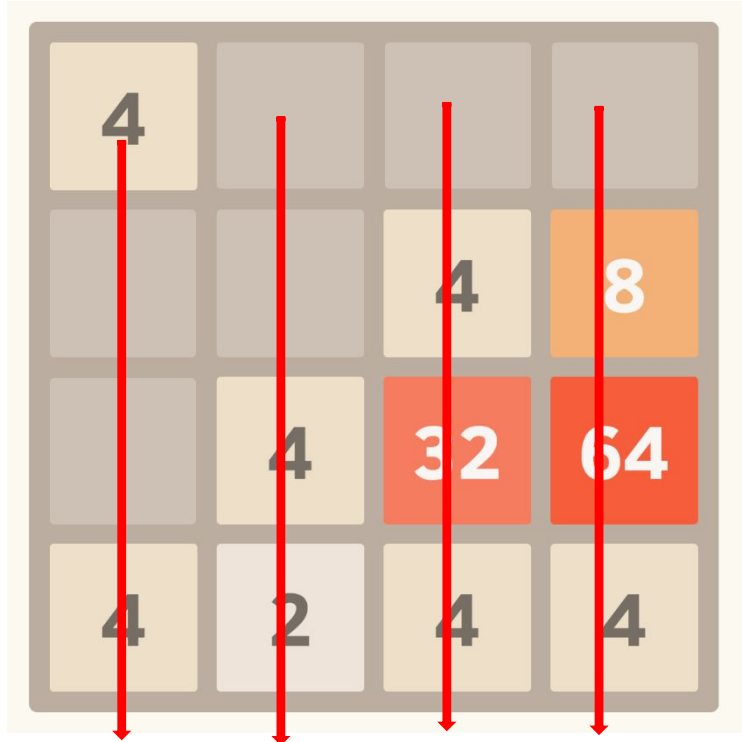
**cout << Dosmil48[2][3];**

Recorrido fila por fila:

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4

Recorrido columna por columna:

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4



Por ejemplo cuando se quiere leer datos desde el teclado y almacenarlos en la matriz.

```
int Dosmil48[ 4 ] [ 4 ];
```

Se recorre la matriz fila por fila

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4

```
for(int f=0; f<4; f++)  
{  
    for(int c=0; c<4; c++)  
    { cout << "Dosmil48[" << f << "][" << c << "]=";  
      cin >> Dosmil48[f][c];  
    }  
}
```



Por ejemplo cuando se quiere leer datos desde el teclado y almacenarlos en la matriz.

```
typedef int type_M1;  
size_t FIL = 4, COL =4;  
type_M1 Dosmil48[ FIL ] [ COL ];
```

Se recorre la matriz fila por fila

	0	1	2	3
0	4			
1			4	8
2		4	32	64
3	4	2	4	4

```
for(int f=0; f<FIL; f++)  
{  
    for(int c=0; c<COL; c++)  
    { cout << "Dosmil48[" << f << "][" << c << "]=";  
      cin >> Dosmil48[f][c];  
    }  
}
```

## Ejemplo: 1

Desarrolle un programa que permita generar aleatoriamente números enteros (entre 0 y 99), los almacene en un array de arrays, cuyas características se muestra en la figura:

	M							
	0	1	2	3	4	5	6	7
0	22	45	67	12	34	21	3	56
1	21	45	65	76	77	89	99	88
2	5	65	34	52	21	33	45	23
3	21	56	78	80	32	45	22	27
4	45	67	56	23	45	4	43	87
5	55	23	23	4	12	3	23	65
6	78	45	45	12	34	23	22	43
7	12	23	28	33	65	12	12	44

continúa

Luego realice lo siguiente:

- 1.Imprima el array de arrays.
- 2.Hallar el dato más pequeño del array de arrays.
- 3.Imprimir la suma de los elementos de la diagonal
- 4.Hallar la suma de los elementos que están por encima de la diagonal.

Ejemplo de la Salida del programa:

65	29	75	29	68	98	70
8	38	63	13	18	71	42
94	69	46	38	4	27	57
97	93	59	45	97	9	79
23	72	79	42	36	72	53
14	65	27	60	92	93	7
94	28	99	61	37	79	24

El menor elemento almacenado en el array es 4

La suma de la diagonal es igual a 347

Suma de los elementos por encima de la diagonal : 1019

El código está distribuido en estos archivos:

main.cpp

Arrays.h

Arrays.cpp

# Arrays.h

```
#ifndef EJEMPLO1_RECORRIDOS_ARRAYS_H
#define EJEMPLO1_RECORRIDOS_ARRAYS_H

#include <iostream>
#include <cstdint> //-- para usar size_t
using namespace std;

typedef long int TipoEntero;
constexpr size_t nFILAS = 8, nCOLUMNAS = 8;

void LlenarArray(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas);
void ImprimirArray(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas);
TipoEntero ElMenor(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas);
TipoEntero SumadeDiagonal(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas);
TipoEntero SumaPorEncimadelaDiagonal(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas);

#endif //EJEMPLO1_RECORRIDOS_ARRAYS_H
```

## main.cpp

```
#include <iostream>
#include <ctime>
#include "Arrays.h"
using namespace std;

int main()
{
    TipoEntero M[nFILAS][nCOLUMNAS];

    srand(time(nullptr));
    cout<<"\n";
    LlenarArray(M,nFILAS, nCOLUMNAS);
    ImprimirArray(M,nFILAS, nCOLUMNAS);
    cout<<"\n";
    cout<<"El menor elemento almacenado en el array es " << ElMenor(M,nFILAS,nCOLUMNAS);
    cout<<"\n";
    cout<<"La suma de la diagonal es igual a " << SumadeDiagonal(M,nFILAS,nCOLUMNAS);
    cout<<"\n";
    cout<<"Suma de los elementos por encima de la diagonal : "
    << SumaPorEncimadelaDiagonal(M,nFILAS,nCOLUMNAS);
    cout<<"\n";
    return 0;
}
```

# Arrays.cpp

```
#include <cstdlib>
#include <iomanip>
#include "Arrays.h"
```

```
void LlenarArray(TipoEntero M[][nCOLUMNAS],size_t filas, size_t columnas)
{
    //-----
    for(size_t f=0; f<filas; f++)
        for(size_t c=0; c<columnas; c++)
            M[f][c] = rand()%100;
}
```

```
void ImprimirArray(TipoEntero M[][nCOLUMNAS],size_t filas, size_t columnas)
{
    //-----
    for(size_t f=0; f<filas; f++)
    {
        for (size_t c = 0; c < columnas; c++)
            cout << setw(5) << M[f][c];
        cout<<"\n";
    }
}
```

# Arrays.cpp

```
TipoEntero ElMenor(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas)
```

```
{//-----
```

```
    TipoEntero Menor;
```

```
    Menor=M[0][0];
```

```
    for(size_t f=0; f<filas; f++)
```

```
        for (size_t c = 0; c < columnas; c++)
```

```
            if(M[f][c]<Menor)
```

```
                Menor =M[f][c] ;
```

```
    return Menor;
```

```
}
```

```
TipoEntero SumadeDiagonal(TipoEntero M[][nCOLUMNAS], size_t filas, size_t columnas)
```

```
{//-----
```

```
    TipoEntero Suma=0;
```

```
    for(size_t f=0; f<filas; f++)
```

```
        Suma+=M[f][f];
```

```
    return Suma;
```

```
}
```



# Arrays.cpp

```
TipoEntero SumaPorEncimadelaDiagonal(TipoEntero M[ ][nCOLUMNAS], size_t filas, size_t columnas)
```

```
{//-----
```

```
-----
```

```
TipoEntero Suma=0;
```

```
for(size_t f=0; f<filas-1; f++)
```

```
    for (size_t c = f+1; c < columnas; c++)
```

```
        Suma+=M[f][c];
```

```
return Suma;
```

```
}
```

	M							
	0	1	2	3	4	5	6	7
0	22	45	67	12	34	21	3	56
1	21	45	65	76	77	89	99	88
2	5	65	34	52	21	33	45	23
3	21	56	78	80	32	45	22	27
4	45	67	56	23	45	4	43	87
5	55	23	23	4	12	3	23	65
6	78	45	45	12	34	23	22	43
7	12	23	28	33	65	12	12	44

# Matrices dinámicas

Se crea una matriz de 4 x 7 en el heap:

	0	1	2	3	4	5	6
0	23	34	45	56	12	50	6
1	76	43	98	35	8	63	12
2	67	2	32	73	12	15	18
3	38	5	3	79	16	20	24

pMatriz

0x7fff5399ea90

00x7fe1094025b0



00x7fe1094025b0	
0x7fe1094025b8	
0x7fe1094025c0	
0x7fe1094025c8	

```
typedef int type_M1;  
size_t nFILA = 4, nCOLUMNA = 7;  
type_M1 **pMatriz= nullptr;  
pMatriz = new type_M1*[nFILA];
```

```

type_M1 **pMatriz = nullptr;
pMatriz = new type_M1*[nFILA];

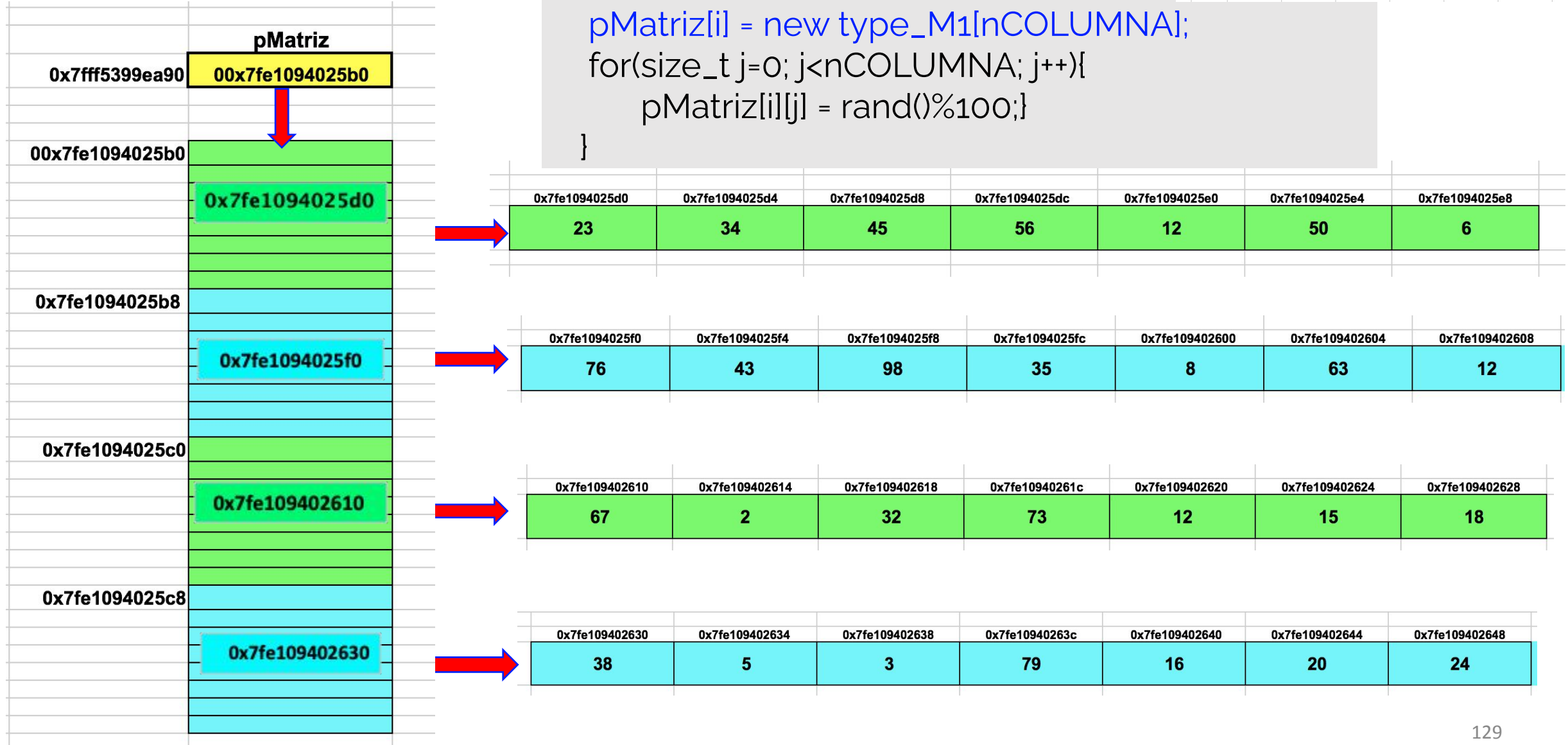
```

```

for(size_t i=0; i<nFILA; i++){
    pMatriz[i] = new type_M1[nCOLUMNA];
    for(size_t j=0; j<nCOLUMNA; j++){
        pMatriz[i][j] = rand()%100;}
}

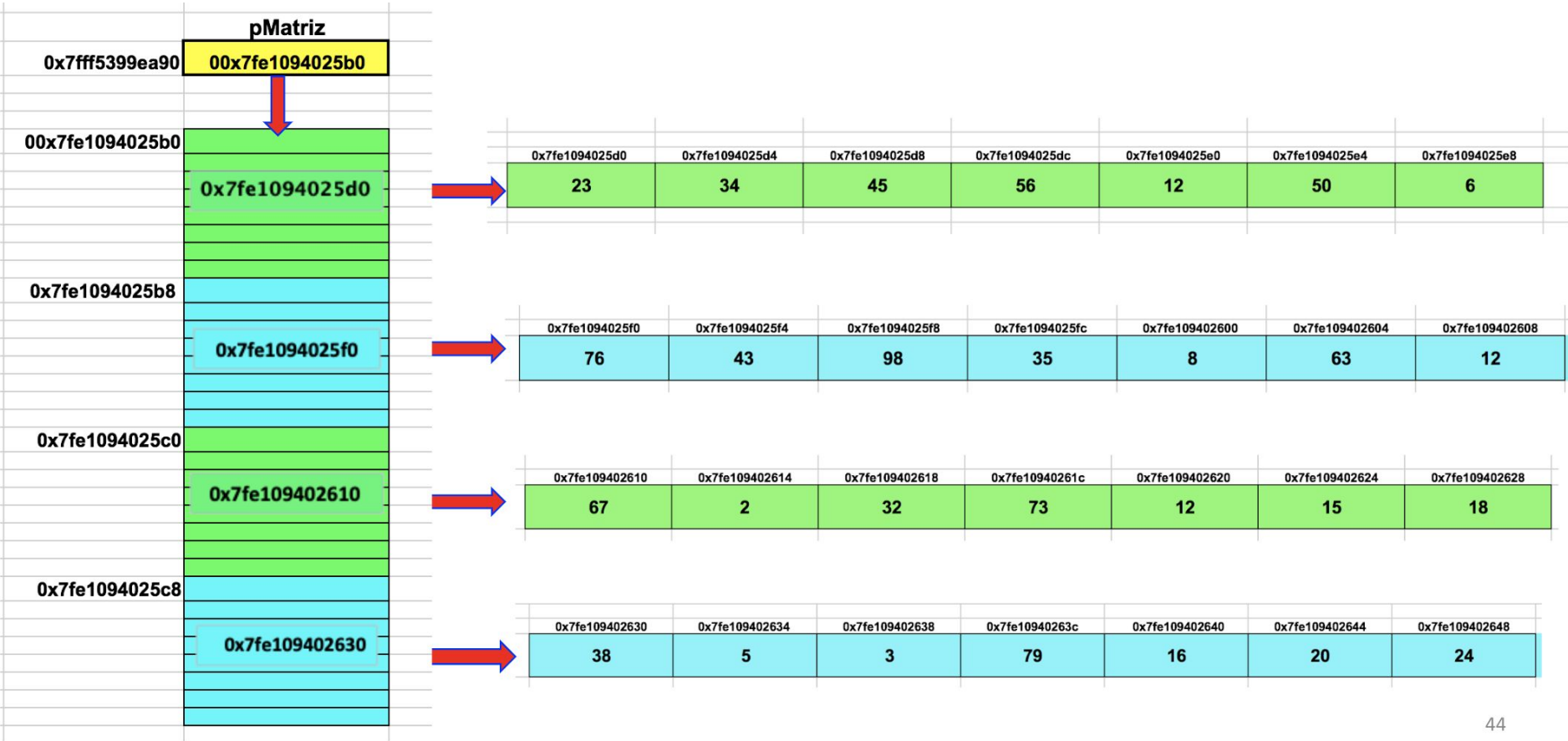
```

	0	1	2	3	4	5	6
0	23	34	45	56	12	50	6
1	76	43	98	35	8	63	12
2	67	2	32	73	12	15	18
3	38	5	3	79	16	20	24



En la matriz definida dinámicamente, y suponiendo que el compilador del Clion utiliza 4 bytes para almacenar un dato int y 8 bytes para almacenar un puntero.

¿ Cuántos bytes en total ocupa la matriz?

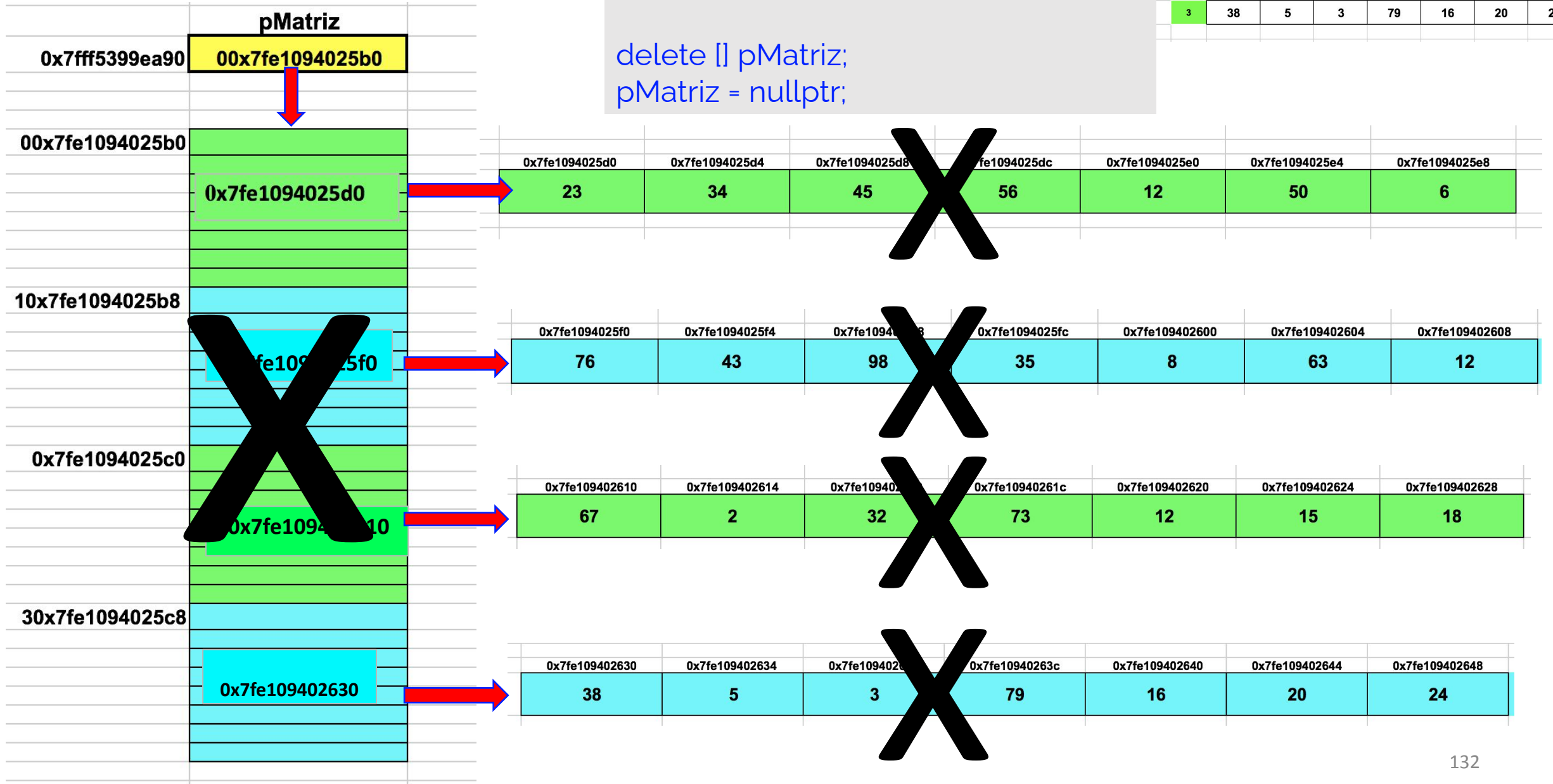


Para liberar memoria

```
for(size_t i=0; i<nFILE; i++)
    delete [] pMatriz[i]
```

```
delete [] pMatriz;
pMatriz = nullptr;
```

	0	1	2	3	4	5	6
0	23	34	45	56	12	50	6
1	76	43	98	35	8	63	12
2	67	2	32	73	12	15	18
3	38	5	3	79	16	20	24





El código está distribuido en estos archivos:

main.cpp

Arrays.h

Arrays.cpp

```
#ifndef ARRAYSDINAMICOS_ARRAYS_H
#define ARRAYSDINAMICOS_ARRAYS_H
```

## Arrays.h

```
#include <iostream>
#include <ctime>
#include <iomanip>
#include <cstdint>
```

```
typedef long int TipoEntero;
constexpr size_t nFILAS = 8, nCOLUMNAS = 8;
constexpr unsigned int maximoNumeroAleatorio = 100;
```

```
void InicializarArrayConNumerosAleatorios(TipoEntero ***matrix);
void printMatrix(TipoEntero *const *matrix);
TipoEntero ElMenor(TipoEntero *const *matrix);
TipoEntero SumadeDiagonal(TipoEntero *const *matrix);
TipoEntero SumaPorEncimadelaDiagonal(TipoEntero *const *matrix);
void printData(TipoEntero *const *matrix);
```

```
#endif //ARRAYSDINAMICOS_ARRAYS_H
```

## main.cpp

```
#include "Arrays.h"
```

```
int main()
```

```
{
```

```
    TipoEntero **matrix = nullptr;
```

```
    InicializarArrayConNumerosAleatorios(&matrix);
```

```
    printMatrix(matrix);
```

```
    printData(matrix);
```

```
    return 0;
```

```
}
```

```
· 80 43 87 6 34 40 82 38
· 3 7 36 80 87 20 78 25
· 33 64 81 56 4 97 36 11
· 20 41 78 64 96 62 61 5
· 39 18 99 59 3 22 39 10
· 15 22 96 31 35 95 88 65
· 57 70 86 75 89 62 6 31
· 59 80 44 4 4 29 75 76
```

```
· El menor elemento almacenado en el array es 3
```

```
· La suma de la diagonal es igual a: 412
```

```
· Suma de los elementos por encima de la diagonal: 1339
```

```
· Process finished with exit code 0
```

# Arrays.cpp

```
#include "Arrays.h"
```

```
void InicializarArrayConNumerosAleatorios(TipoEntero ***matrix)
```

```
{//-----  
    *matrix = new TipoEntero*[nFILAS];  
    for(int i=0; i<nFILAS;i++){  
        (*matrix)[i] = new TipoEntero[nCOLUMNAS];  
    }  
    srand(time(nullptr));  
    for(int i = 0; i < nFILAS; i++) {  
        for(int j = 0; j < nCOLUMNAS; j++) {  
            (*matrix)[i][j] = rand() % maximoNumeroAleatorio;  
        }  
    }  
}
```

```
void printMatrix(TipoEntero *const *matrix)
```

```
{  
    for(size_t f=0; f < nFILAS; f++)  
    {  
        for (size_t c = 0; c < nCOLUMNAS; c++)  
            std::cout << std::setw(5) << matrix[f][c];  
        std::cout<<std::endl;  
    }  
}
```

# Arrays.cpp

```
TipoEntero ElMenor(TipoEntero *const *matrix)
{
    //-----
    TipoEntero Menor;
    Menor=matrix[0][0];
    for(size_t f=0; f<nFILAS; f++)
        for (size_t c = 0; c < nCOLUMNAS; c++)
            if(matrix[f][c]<Menor)
                Menor =matrix[f][c] ;
    return Menor;
}
```

```
TipoEntero SumadeDiagonal(TipoEntero *const *matrix)
{
    TipoEntero Suma=0;

    for(size_t f=0; f<nFILAS; f++)
        Suma+=matrix[f][f];
    return Suma;
}
```

```

TipoEntero SumaPorEncimadelaDiagonal(TipoEntero *const *matrix)
{
    TipoEntero Suma=0;

    for(size_t f=0; f<nFILAS-1; f++)
        for (size_t c = f+1; c < nCOLUMNAS; c++)
            Suma+=matrix[f][c];
    return Suma;
}

void printData(TipoEntero *const *matrix)
{
    std::cout << std::endl;
    std::cout << "El menor elemento almacenado en el array es "
                << ElMenor(matrix)
                << std::endl;

    std::cout << "La suma de la diagonal es igual a: "
                << SumadeDiagonal(matrix)
                << std::endl;

    std::cout << "Suma de los elementos por encima de la diagonal: "
                << SumaPorEncimadelaDiagonal(matrix)
                << std::endl;
}

```

## Arrays.cpp

	M							
	0	1	2	3	4	5	6	7
0	22	45	67	12	34	21	3	56
1	21	45	65	76	77	89	99	88
2	5	65	34	52	21	33	45	23
3	21	56	78	80	32	45	22	27
4	45	67	56	23	45	4	43	87
5	55	23	23	4	12	3	23	65
6	78	45	45	12	34	23	22	43
7	12	23	28	33	65	12	12	44

# Resumen

- Los arreglos y matrices estáticas permiten crear de forma simple una colección de datos donde el acceso a cada dato por medio de los subíndices.
- Los arreglos y matrices dinámicos requieren de mayor cantidad de pasos para crear la colección y requiere que sean liberados manualmente.
- Los arreglos y matrices dinámicos permiten generar colecciones de datos de mayor tamaño debido a que permiten un acceso al heap.

¡Nos vemos en la siguiente clase!

