CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1
CICLO 2018-2

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Unidad 6: Punteros

http://bit.ly/2p3fgiD

Profesores:

Ernesto Cuadros-Vargas, PhD. *ecuadros@utec.edu.pe*

María Hilda Bermejo, M. Sc. *mbermejo@utec.edu.pe*

Telegram:

1. **Configurar tu cuenta**

2. **Link: http://bit.ly/2OW5Ss9**

SISAP:

Evento: SISAP 2018 – 11th International Conference on Similarity Search and Applications
Fechas: October 7-9 Lima, Perú
Resumen: http://www.sisap.org/2018/

The 11th International Conference on Similarity Search and Applications (SISAP) is an annual forum for researchers and application developers in the area of similarity data management. It aims at the technological problems shared by numerous application domains, such as data mining, information retrieval, multimedia, computer vision, pattern recognition, computational biology, geography, biometrics, machine learning, and many others that make use of similarity search as a necessary supporting service.

Inscripciones: https://eventos.spc.org.pe/spire2018/registration_sisap.html

SPIRE:

SPIRE 2018: 25th International Symposium on String Processing and Information Retrieval
Fechas: October 9-11 Lima, Perú
Resumen: https://eventos.spc.org.pe/spire2018/venue.html

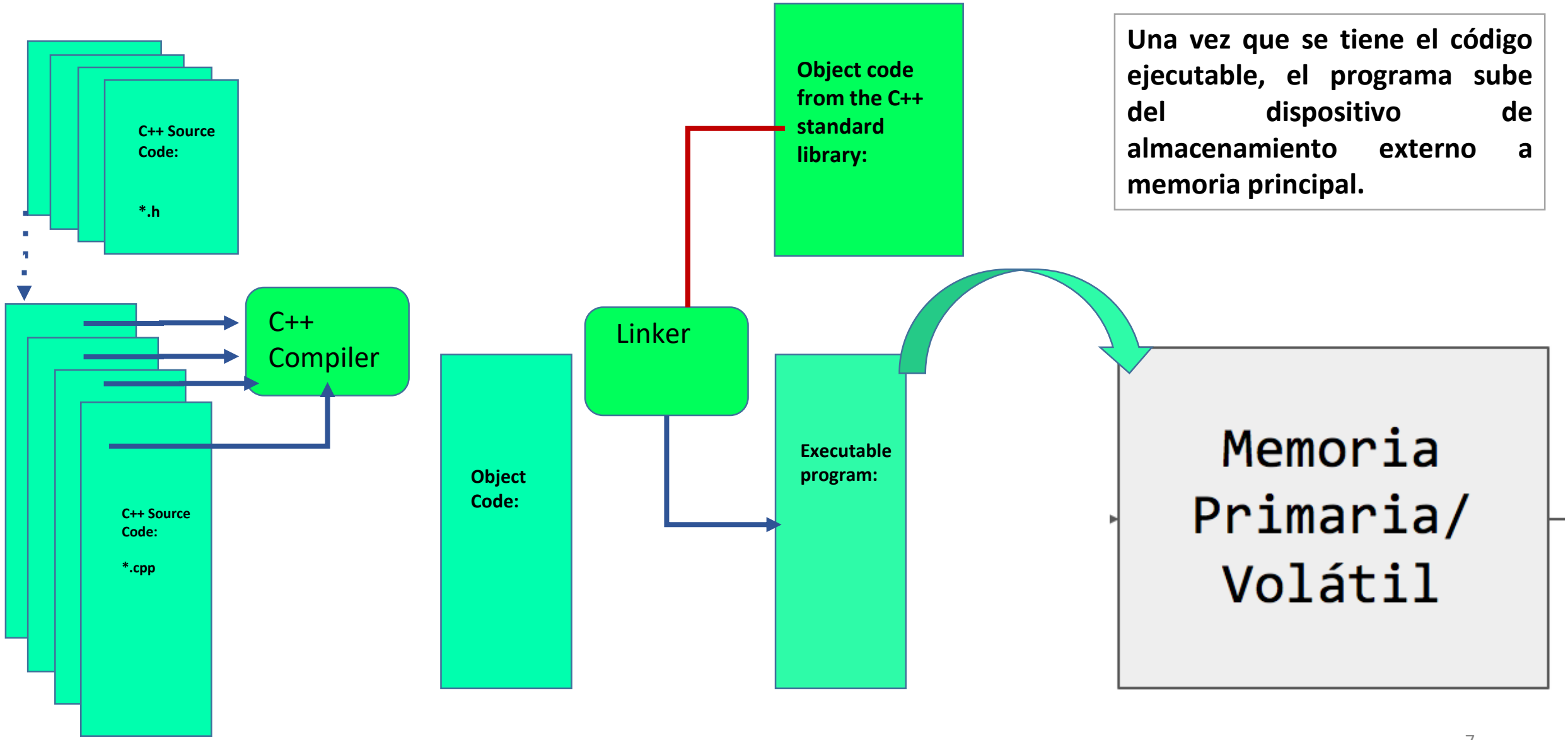
SPIRE 2018 is the 25th edition of the annual Symposium on String Processing and Information Retrieval. SPIRE has its origins in the South American Workshop on String Processing, which was first held in Belo Horizonte, Brazil, in 1993. Since 1998 the focus of the workshop has also included information retrieval, due to its increasing relevance to and inter-relationship with string processing.
SPIRE 2018 will be held in UTEC Lima, Peru.
Inscripciones: https://eventos.spc.org.pe/spire2018/registration.html

Logro de la sesión:

Al finalizar la sesión, los alumnos desarrollan sus programas utilizando punteros y arrays dinámicos.

Punteros

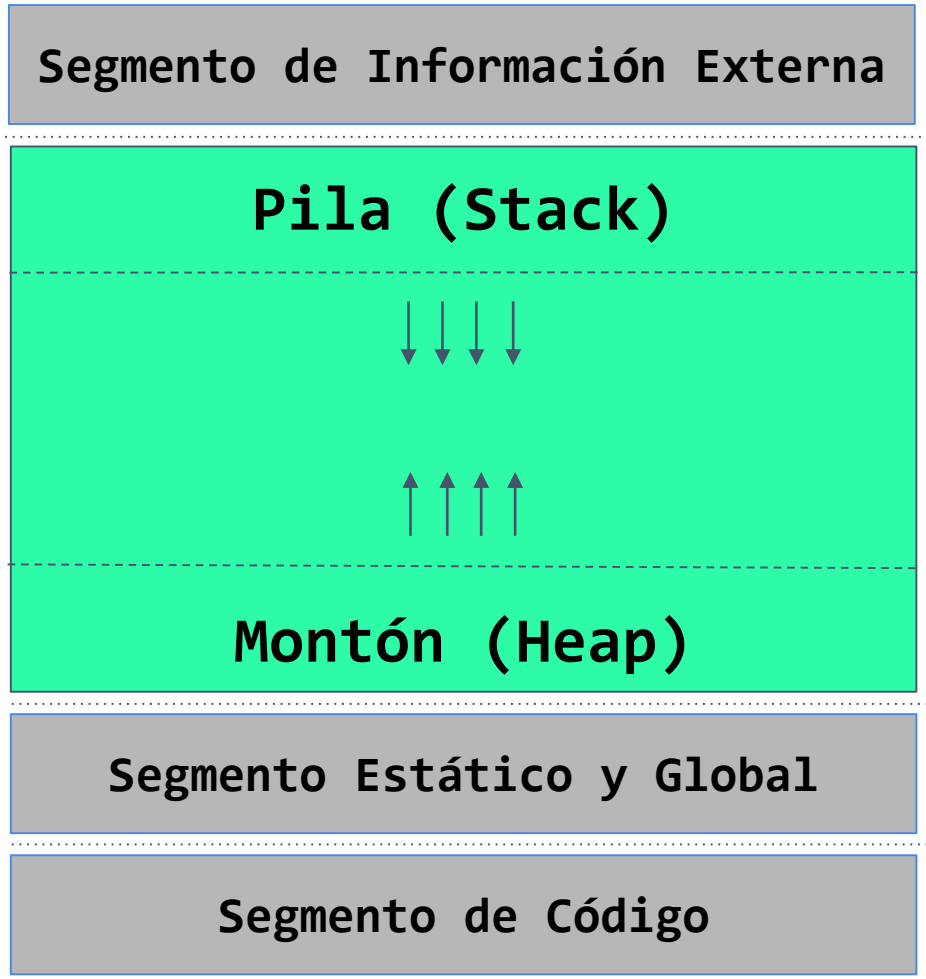
¿Qué ocurre con el código desde que se escribe hasta que se ejecuta?



Una vez que se tiene el código ejecutable, el programa sube del dispositivo de almacenamiento externo a memoria principal.

Stroustrup, B; Programming Principles and Practice Using C++

Uso Uso de la memoria primaria en C++



Argumento externos (argc, argv)

Memoria automática donde se asigna las variables estáticamente.

Memoria donde se asigna variables dinámicamente.

Memoria donde se asigna variables globales y tipo static.

Memoria donde se guarda el segmento de codigo

Representación simplificada de la memoria

- Cada Byte tiene su propia dirección

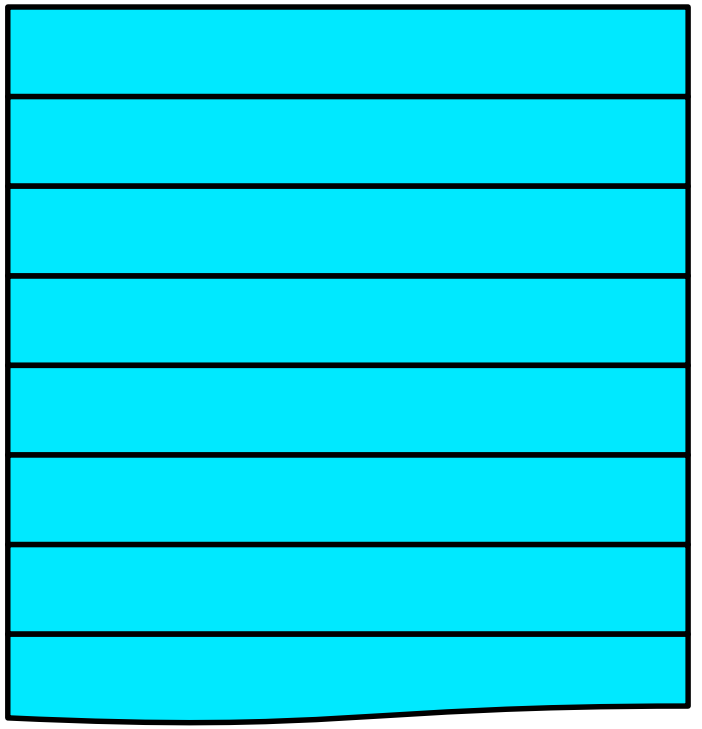
1 Byte

0	1	0	1	0	0	1	1

Dirección

0x1008
0x1009
0x100A
0x100B
0x100C
0x100D
0x100F
0x1010

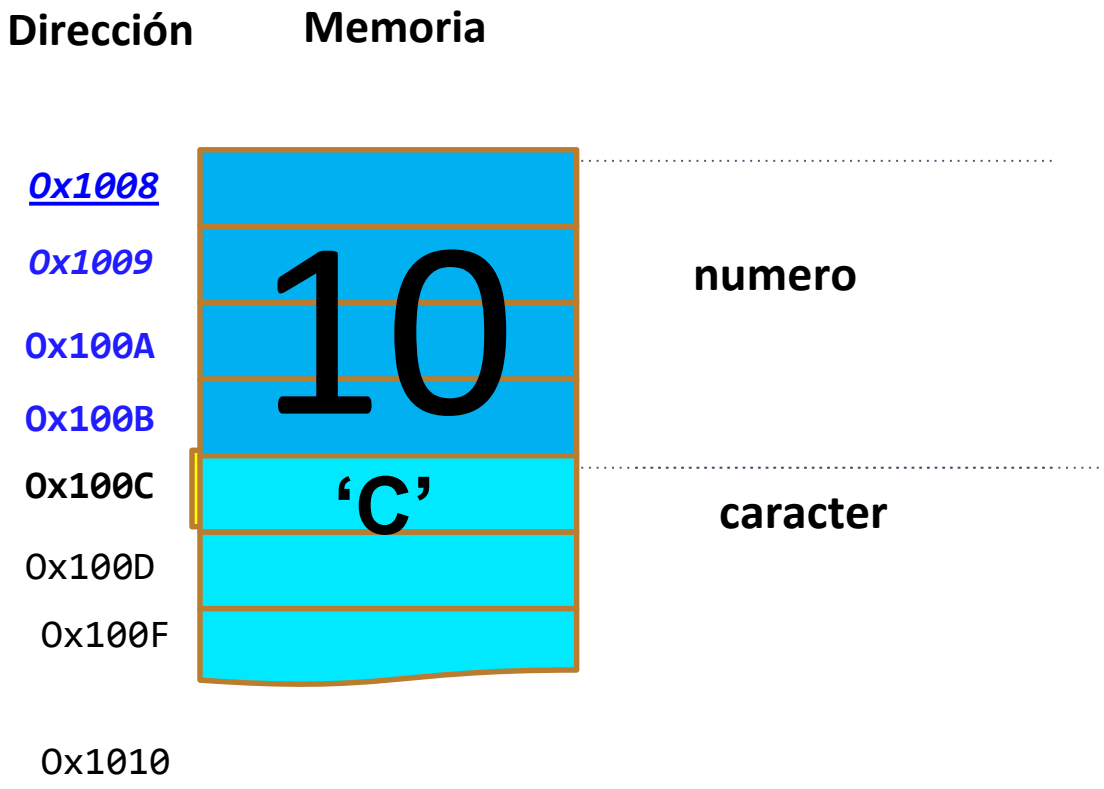
Memoria



Representación de una variable

- Variable: nombre alfanumérico y **tipo de dato**.

```
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.
- **La dirección de menor valor es la dirección**

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

Dirección de una variable

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

Ejemplo:

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

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

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

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

    return 0;
}
```

```
Numero = 10
La direccion de la variable numero es 0x7fff599c7a98

x = 12.75
La direccion de la variable x es 0x7fff599c7a94
Process finished with exit code 0
```

¿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 asi:

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

La segunda instrucción define **p** como un 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 null pointer, 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 direccion de **ival**; **p** es un puntero a **ival**
cout << *p; // * se accede al objeto al cual apunta **p**; se imprime 42

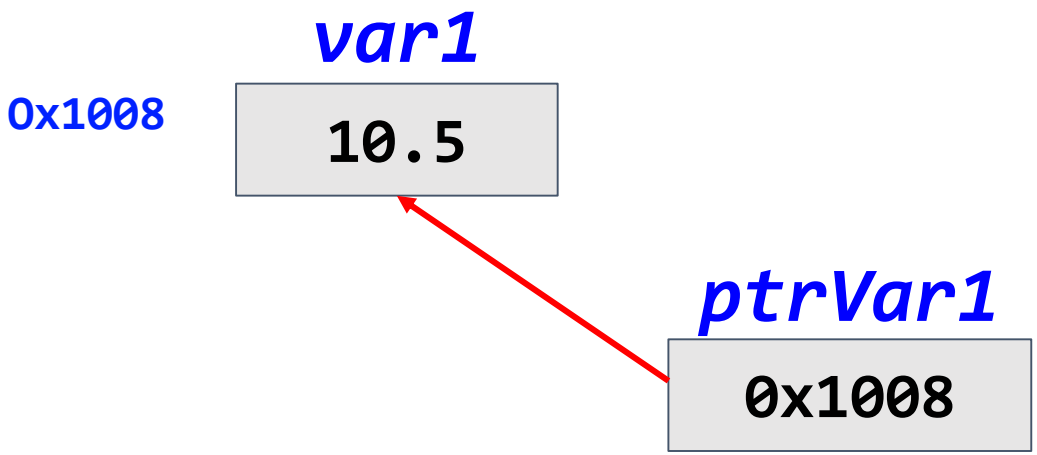
Desreferenciando un puntero se accede al objeto que es apuntado por el puntero.

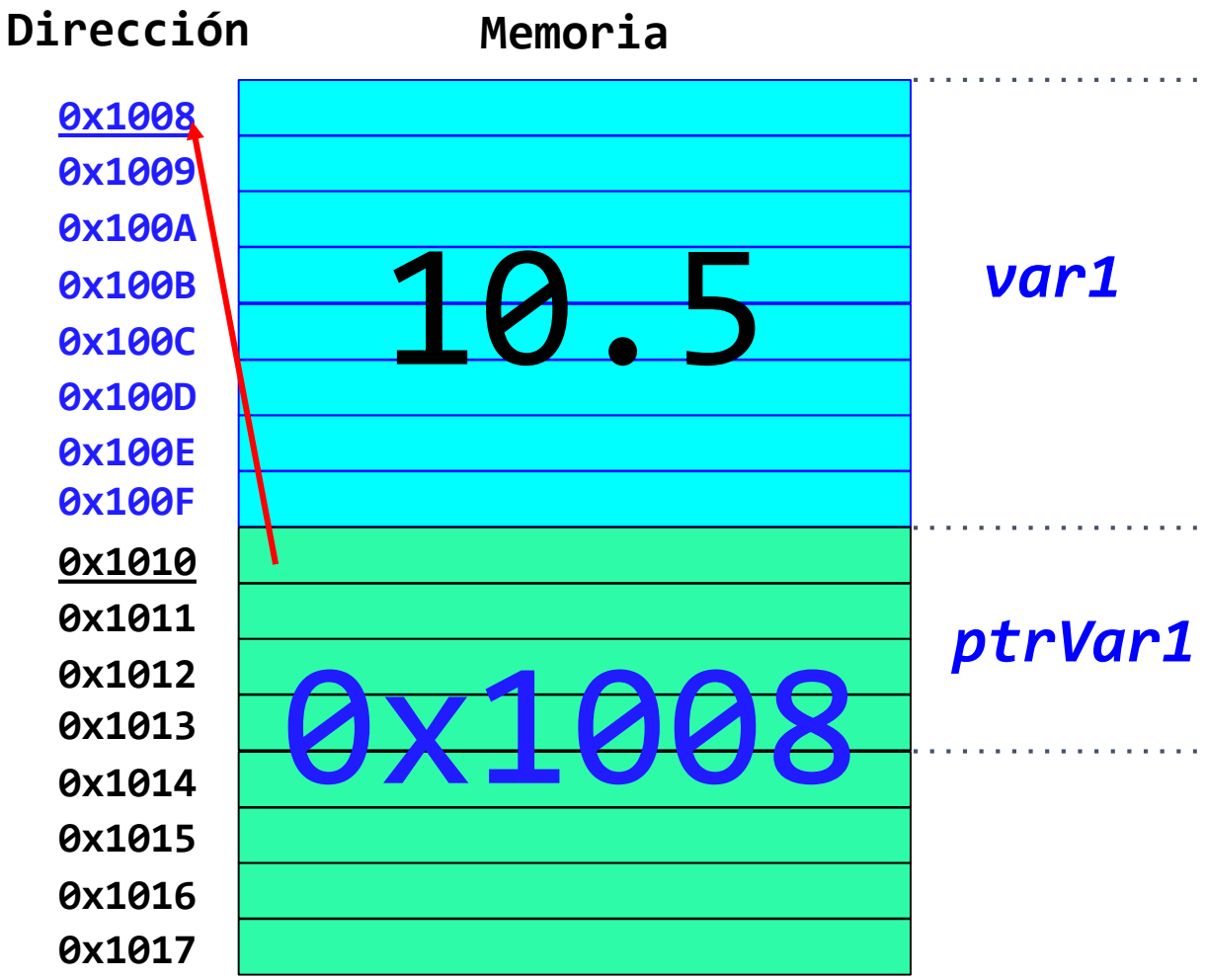
***p = 0;** // * se accede al objeto, se asigna un nuevo valor a **ival** a través de **p**
// Cuando se asigna a ***p,** estamos asignando el valor 0 al objeto al cual **p** apunta

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.

Null Pointers

Un **null pointer** 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 zero=0;
pi = zero; **//** error, no se puede asignar un **int** a un puntero

Mas 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** tiene la dirección del mismo objeto en el ejemplo 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

Mas 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;
```

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

```
int i =0;
```

a) `double *p = &i;`

b) `int *pi =i;`

c) `int *p=&i;`

Ejercicios:

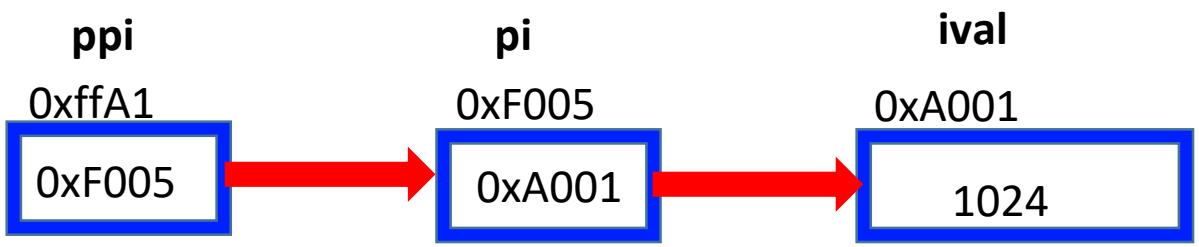
3. ¿Por qué la inicialización de **p** es legal y la de **lp** es ilegal?

```
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;
i**nt *pi = &ival**; // **pi** es un puntero a un **int**
int **ppi = π // **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 asigna 2 a **ival**

int &refVal2; // error: una referencia debe ser inicializado

Recordando Transferencia de parametros

```
int main()
{
  int   x=5, y=4, *p, *q, **pp;
  p = &x;          q = &y; pp = &p;
 cout <<"Suma = " << fSuma(x,y) << endl;
 swap(x,y);
 cout << "x = "<<  x << "y=" << y << endl;

 swap(&x, &y);
 cout << "x = "<<  x << "y=" << y << endl;

 swap(p, q);
 swap(*pp, &y);   swap(*pp, q);

 cout << "x = "<<  x << "y=" << y << endl;
 return(0);
}
```

```
void 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;
}
```

Referencia a Punteros:

Una referencia no es un objeto. Por lo tanto no se 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.
```

Recordando Transferencia de parametros

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

  swap(x,y);

  swap(p, q);

  swap(&x, &y);

}
```

```
void 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;
}
```

Analizando código

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

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

  f1(x);
  f1(5);
  f1(*p);
  f1(r);
  f1(**pp);
 return(0);
}
```

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

```
int main()
{
   int   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(int & rn)
{
  rn++;
}
```

```
int main()
{
   int   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);
}
```

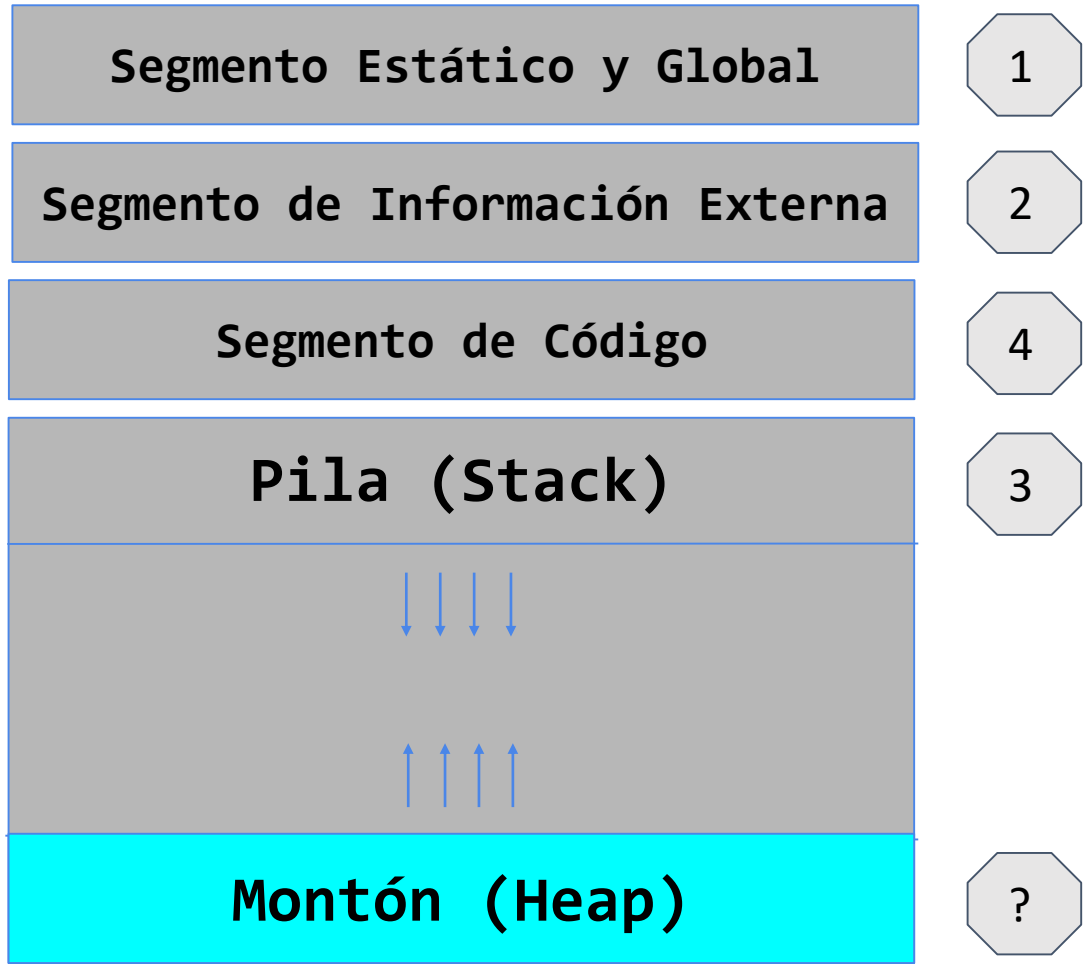
```
void f3(int * pn)
{
  ++*pn;
  pn = nullptr;
}
```

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

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

 f4(&x);  // error
 f4(&r);     // error
 f4(p);
 p = &x;
 f4(**p);   // f4(p);
 return(0);
}
```

```
void f4(int *& rp)
{
  ++*rp;
   rp=nullptr;
}
```

Manejando memoria directamente

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.

Operadores para asignar y liberar memoria dinámica

new asigna 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

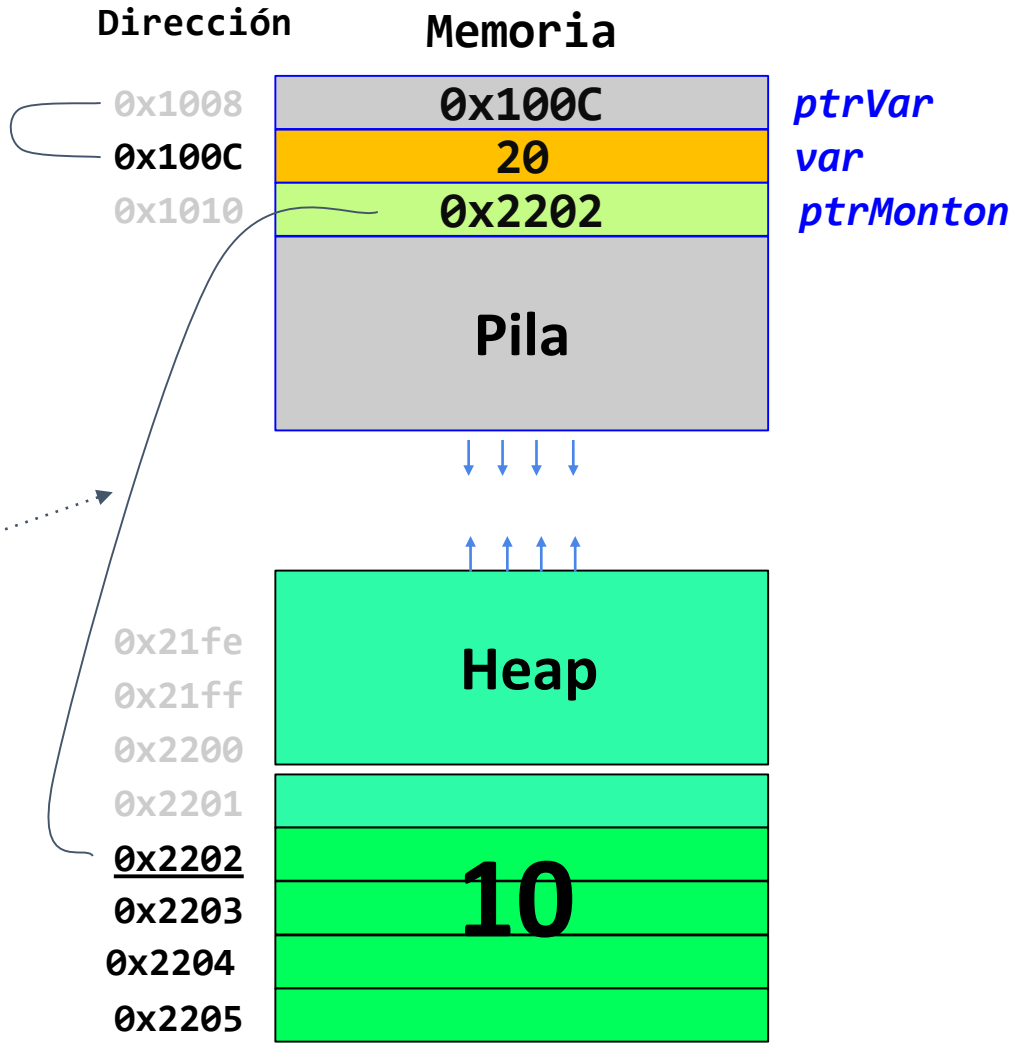
Liberando memoria dinámica:

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



Ejemplo 1:

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

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {  double *pnumero1= nullptr, *pnumero2= nullptr;
7                                                    ¿
8     pnumero1 = new double;
9     pnumero2 = new double;
10
11    cout << "Numero 1 : ";
12    cin >> *pnumero1;      //-- se lee el numero en el sitio apuntado por el puntero
13    cout << "Numero 2  : ";
14    cin >> *pnumero2;
15    cout << "\n";
16    cout << "La Suma es : " << *pnumero1  + *pnumero2 << "\n";
17    cout << "La Diferencia es : " << *pnumero1  - *pnumero2 << "\n";
18    cout << "El Producto es : " << *pnumero1  * *pnumero2 << "\n";
19    delete pnumero1;
20    delete pnumero2;
21    return 0;
22  }
```

```
Numero 1 : 5
Numero 2  : 3

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

Importante:

Suponiendo que un dato de tipo double en el ambiente de Clion utiliza 8 bytes para ser almacenado.

¿ 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?

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {  double *pnumero1= nullptr, *pnumero2= nullptr;
7
8     pnumero1 = new double;
9     pnumero2 = new double;
10
11    cout << "Numero 1 : ";
12    cin >> *pnumero1;      //-- se lee el numero en el sitio apuntado por el puntero
13    cout << "Numero 2  : ";
14    cin >> *pnumero2;
15    cout << "\n";
16    cout << "La Suma es : " << *pnumero1  + *pnumero2 << "\n";
17    cout << "La Diferencia es : " << *pnumero1  - *pnumero2 << "\n";
18    cout << "El Producto es : " << *pnumero1  * *pnumero2 << "\n";
19    delete pnumero1;
20    delete pnumero2;
21    return 0;
22  }
```

Los objetos creados dinámicamente existen hasta que sean liberados de la memoria:

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

```
Foo * factory (T arg)
{
   // process arg as parameter
  return new Foo(arg);   //-- caller es responsible for deleting this memory
}

void use_factory(arg)
{
   Foo *p =factory(arg);
  //  use p but do not delete it
  //   p goes out of scope, but the memory to which p point is not freed!.
}
```

```
void use_factory(arg)
{
   Foo *p =factory(arg);
   // use p
  delete p;
}
```

Arreglos dinámicos

Array dinámicos:

```
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};
```

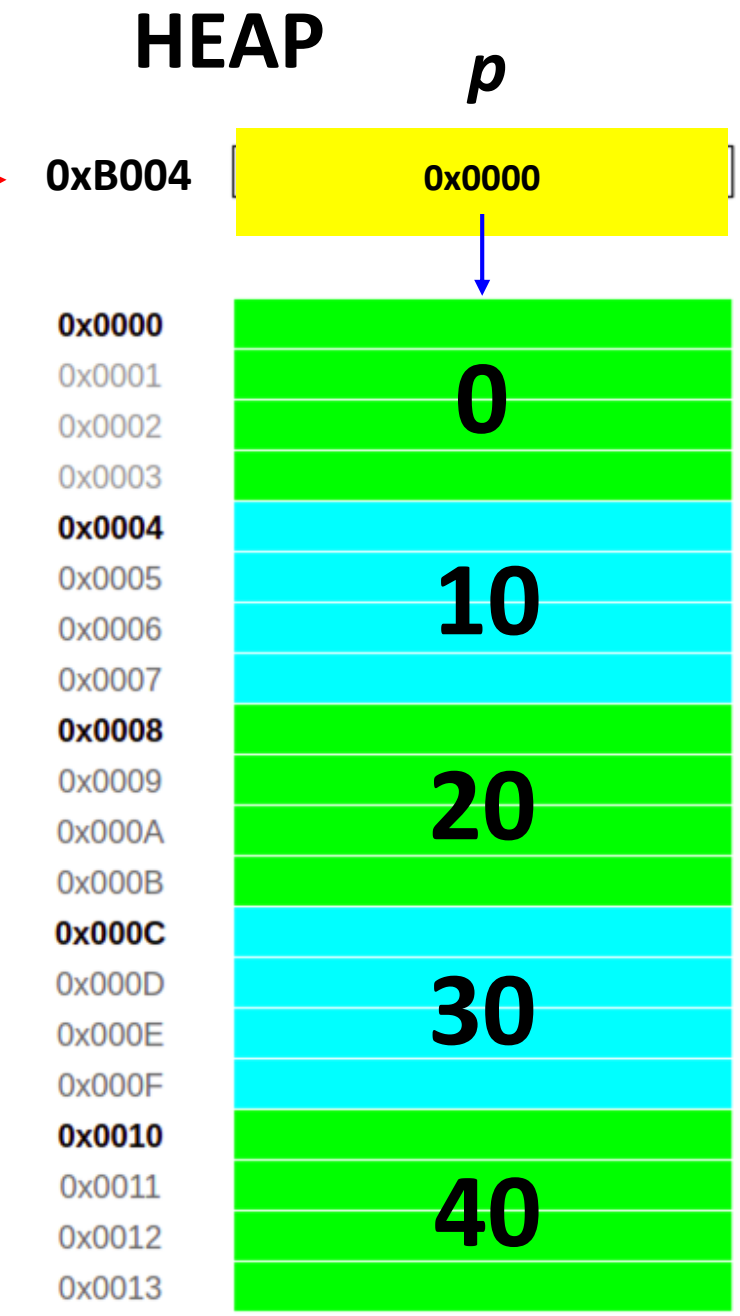
¿

Para liberar el espacio de memoria:

```
delete [] pia;
```

Se crea un array de 5 elementos en el heap:

```
int *p;

p = new int[5];

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

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



Ejemplo 1:

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 array dinámicos, el primero con los múltiplos de 5 y el segundo con los múltiplos de 7 que tenga el primer array.**

Solución 1 – Solo un archivo. Main.cpp

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

void Imprimir(int *pArreglo, size_t n);

int main()
{ size_t numDatos;
  int *pA,*pMultiplode5, *pMultiplosde7;
  size_t contMulde5=0, contMulde7=0;

  srand(time(nullptr));
  cout <<"Numero de elementos : ";
  cin >> numDatos;

  pA = new int[numDatos];
  for(size_t i=0; i<numDatos; i++)
  {
    pA[i] = rand()%1000;
    if(pA[i]%5==0)
      contMulde5++;
    if(pA[i]%7==0)
      contMulde7++;
  }
```

```
25      //-- se crea los otros dos arrays
26      pMultiplode5 = new int[contMulde5];
27      pMultiplosde7= new int[contMulde7];
28      //-- se llenan los datos segun sea el caso
29      size_t indiceMul5=0, indiceMul7=0;
30      for(size_t i=0; i<numDatos; i++)
31      {
32        if(pA[i]%5==0)
33          pMultiplode5[indiceMul5++]=pA[i];
34        if(pA[i]%7==0)
35          pMultiplosde7[indiceMul7++]=pA[i];
36      }
37      cout<<"\n\nArray con datos\n";
38      Imprimir(pA,numDatos);
39      cout<<"\nArray con los multiplos de 5\n";
40      Imprimir(pMultiplode5,contMulde5);
41      cout<<"\nArray con los multiplos de 7\n";
42      Imprimir(pMultiplosde7,contMulde7);
43      delete [] pA;
44      delete [] pMultiplode5;
45      delete [] pMultiplosde7;
46      return 0;
47  }
```

```
48
49  void Imprimir(int *pArreglo, size_t n)
50  {
51    for(size_t i=0; i<n; i++)
52      cout << setw(5) << pArreglo[i];
53  }
54
```

```
Numero de elementos : 25


Array con datos
  117   37  215  334  643  638  138  987  861  638   93  567  397  142   72  866  537  941  263  951  240  882  612  344  503
Array con los multiplos de 5
  215  240
Array con los multiplos de 7
  987  861  567  882
```

Solucion 2.
Utilizando funciones

Archivos:

Main.cpp
Array.h
Array.cpp

Main.cpp

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

int main()
{ size_t numDatos;
  int *pA,*pMultiplode5, *pMultiplosde7;
  size_t contMulde5=0, contMulde7=0;

  srand(time(nullptr));
  cout <<"Numero de elementos : ";
  cin >> numDatos;

  pA = CreayLlenaDatos(numDatos, contMulde5, contMulde7);
  pMultiplode5=CrearArrayconMultiplos(pA, numDatos, 5, contMulde5);
  pMultiplosde7=CrearArrayconMultiplos(pA, numDatos, 7,contMulde7);

  cout<<"\n\nArray con datos\n";
  Imprimir(pA,numDatos);
  cout<<"\nArray con los multiplos de 5\n";
  Imprimir(pMultiplode5,contMulde5);
  cout<<"\nArray con los multiplos de 7\n";
  Imprimir(pMultiplosde7,contMulde7);
  Eliminar(pA);
  Eliminar(pMultiplode5);
  Eliminar(pMultiplosde7);
  return 0;
}
```

Array.h

```
1   //
2   // Created by Maria Hilda Bermejo on 9/24/18.
3   //
4
5   #ifndef ARRAY_DINAMICO_MULTIPLOS_ARRAYS_H
6   #define ARRAY_DINAMICO_MULTIPLOS_ARRAYS_H
7
8   #include <cstddef>
9   #include <cstdlib>
10  #include <iomanip>
11
12  int * CreayLlenaDatos(size_t numDatos, size_t & contMulde5, size_t &contMulde7);
13  int * CrearArrayconMultiplos(int *A, size_t num, int multiplo, size_t contMultiplos);
14  void Imprimir(int *pArreglo, size_t n);
15  void  Eliminar(int * &pA);
16
17
18  #endif //ARRAY_DINAMICO_MULTIPLOS_ARRAYS_H
19
```

Array.cpp

```cpp
//
// Created by Maria Hilda Bermejo on 9/24/18.
//

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


int * CreayLlenaDatos(size_t numDatos, size_t & contMulde5, size_t &contMulde7)
//-----------------------------------------------------------------------
{int *pA;

  pA = new int[numDatos];
  for(size_t i=0; i<numDatos; i++)
  {
    pA[i] = rand()%1000;
    if(pA[i]%5==0)
      contMulde5++;
    if(pA[i]%7==0)
      contMulde7++;
  }
  return pA;  //-- devuelve el puntero que apunta al inicio del array
}

```

Array.cpp

```
26  int * CrearArrayconMultiplos(int *A, size_t num, int multiplo, size_t contMultiplos)
27  //-------------------------------------------------------------------------------
28  { //--- A es el puntero al array con todos los numeros
29    //---  num es la cantidad de numeros en el array
30    //--- multiplo, indica cuales multiplos se va a escoger para formar el nuevo array
31    //--- contMultiplos es la cantidad de multiplos que hay en el array original
32    int *pAM;
33    pAM = new int[contMultiplos];
34    //-- se llenan los datos segun sea el caso
35    size_t indice=0;
36    for(size_t i=0; i<num; i++)
37    {
38      if(A[i]%multiplo==0)
39        pAM[indice++]=A[i];
40    }
41    return pAM;  //-- devuelve el array con los multiplos
42  }
43
44  void Imprimir(int *pArreglo, size_t n)
45  //-------------------------------
46  { for(size_t i=0; i<n; i++)
47      cout << setw(5) << pArreglo[i];
48  }
49
50  void  Eliminar(int * &pA)
51  //--------------------
52  {
53    delete []pA;
54    pA= nullptr;  //-- se esta lacrando el puntero.
55  }
```

Matrices dinámicas

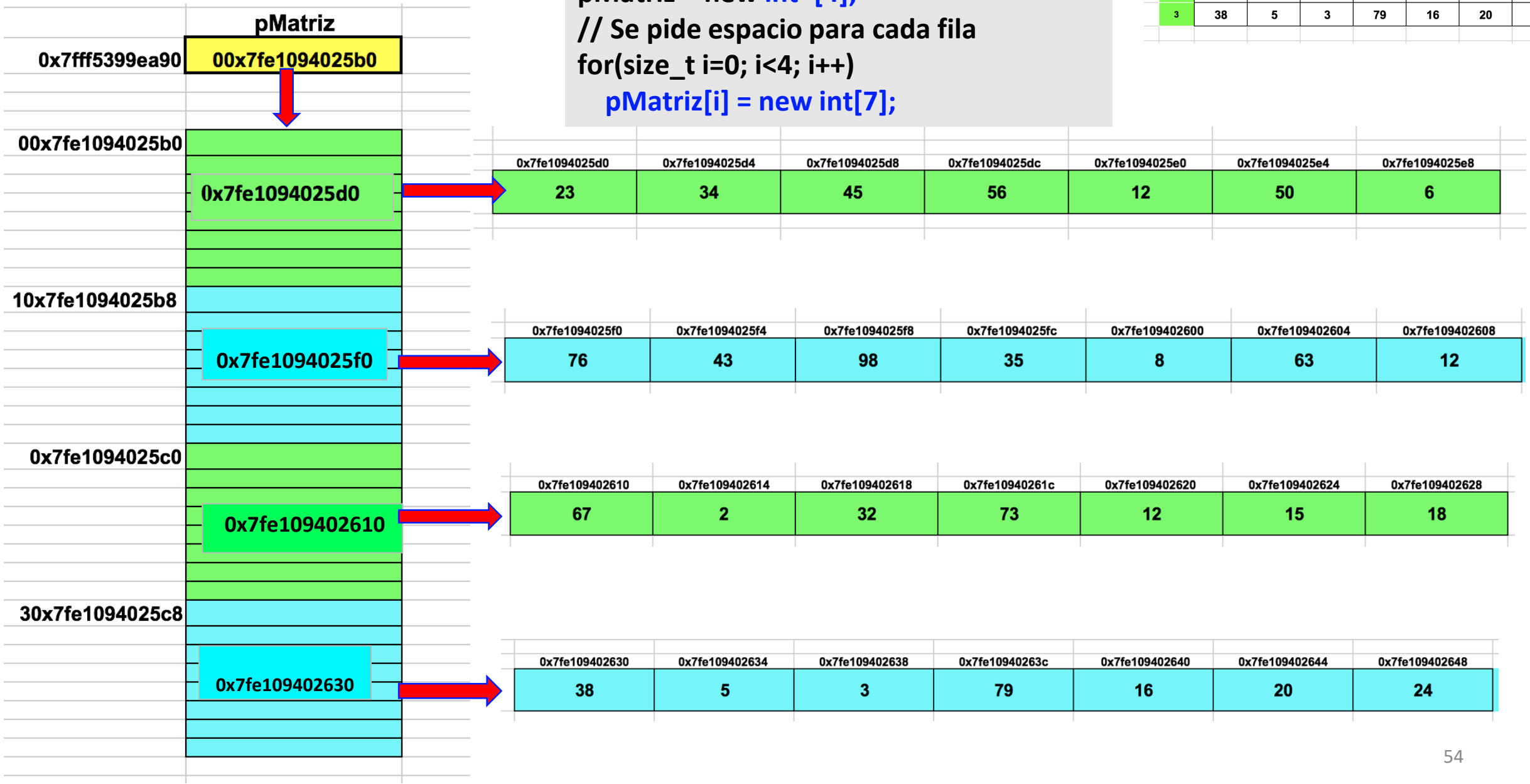
Se crea una matriz de 4 x 7 en el heap:

	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

4x7

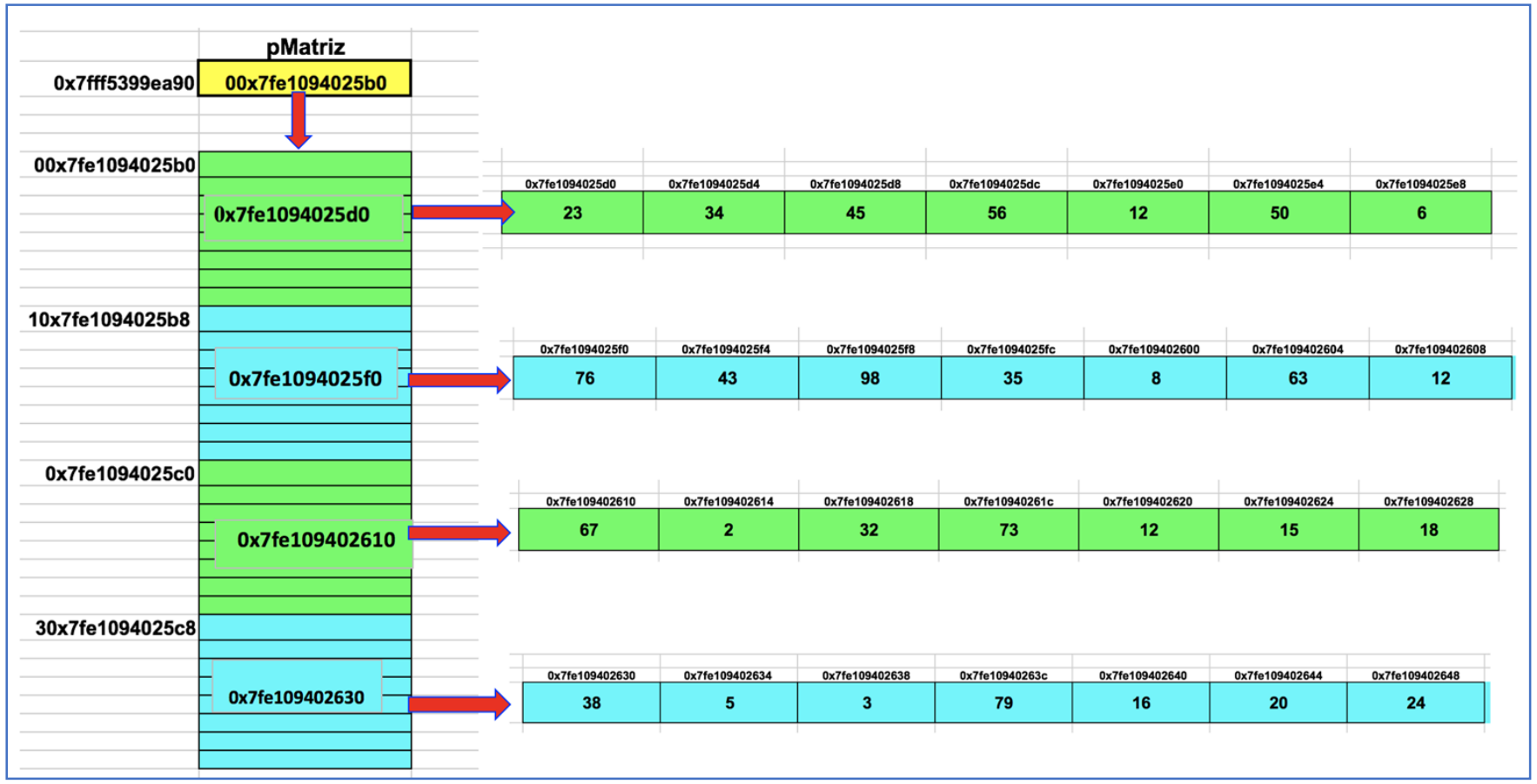
```
int **pMatriz= nullptr;
pMatriz = new int*[4];
// Se pide espacio para cada fila
for(size_t i=0; i<4; i++)
    pMatriz[i] = new int[7];
```

	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.

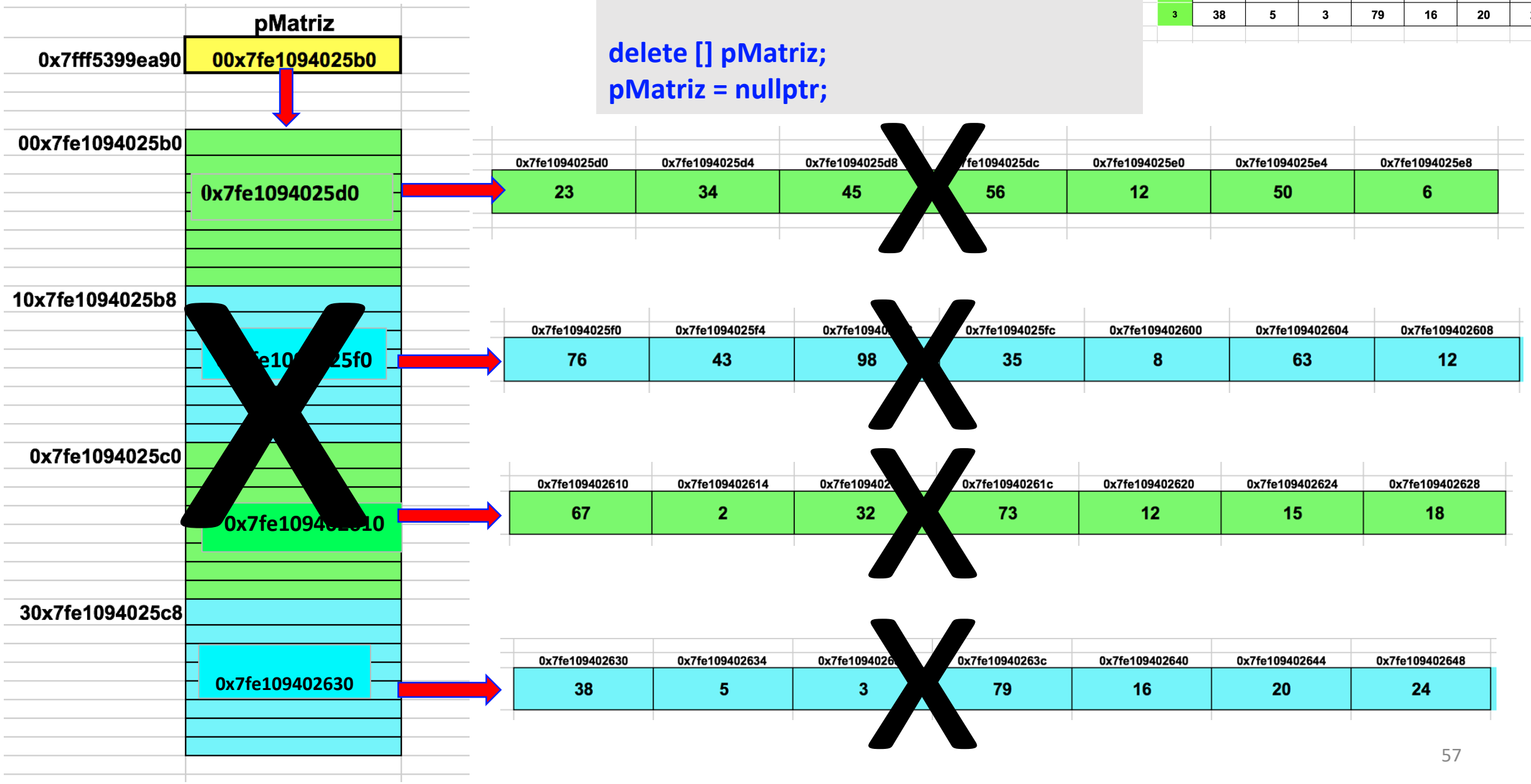
¿ Cuántos bytes en total ocupa la matriz?



Para liberar memoria

```
for(size_t i=0; i<4; 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



CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1
CICLO 2018-2

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Unidad 6: Punteros

http://bit.ly/2p3fgiD

Profesores:

Ernesto Cuadros-Vargas, PhD. *ecuadros@utec.edu.pe*

María Hilda Bermejo, M. Sc. *mbermejo@utec.edu.pe*

El Código de los programas lo pueden ubicar en este

http://bit.ly/2QYctEc