CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1
CICLO 2019-1

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CIENCIA DE LA COMPUTACIÓN

Unidad 5: Punteros - Parte 2

http://bit.ly/2HRBWgq

Profesores:

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

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

Telegram:

1. Configurar tu cuenta

2. http://bit.ly/2TJnwBq

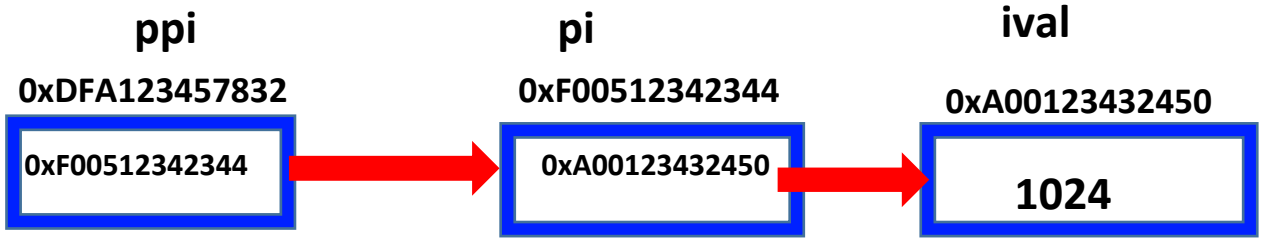
Logro de la sesión:

Al finalizar la sesión, los alumnos desarrollan sus programas utilizando punteros y asignación dinámica de memoria.

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 ;** // **pi** es un puntero a un **int**
pi = &ival;
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

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.
```

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);
}
```

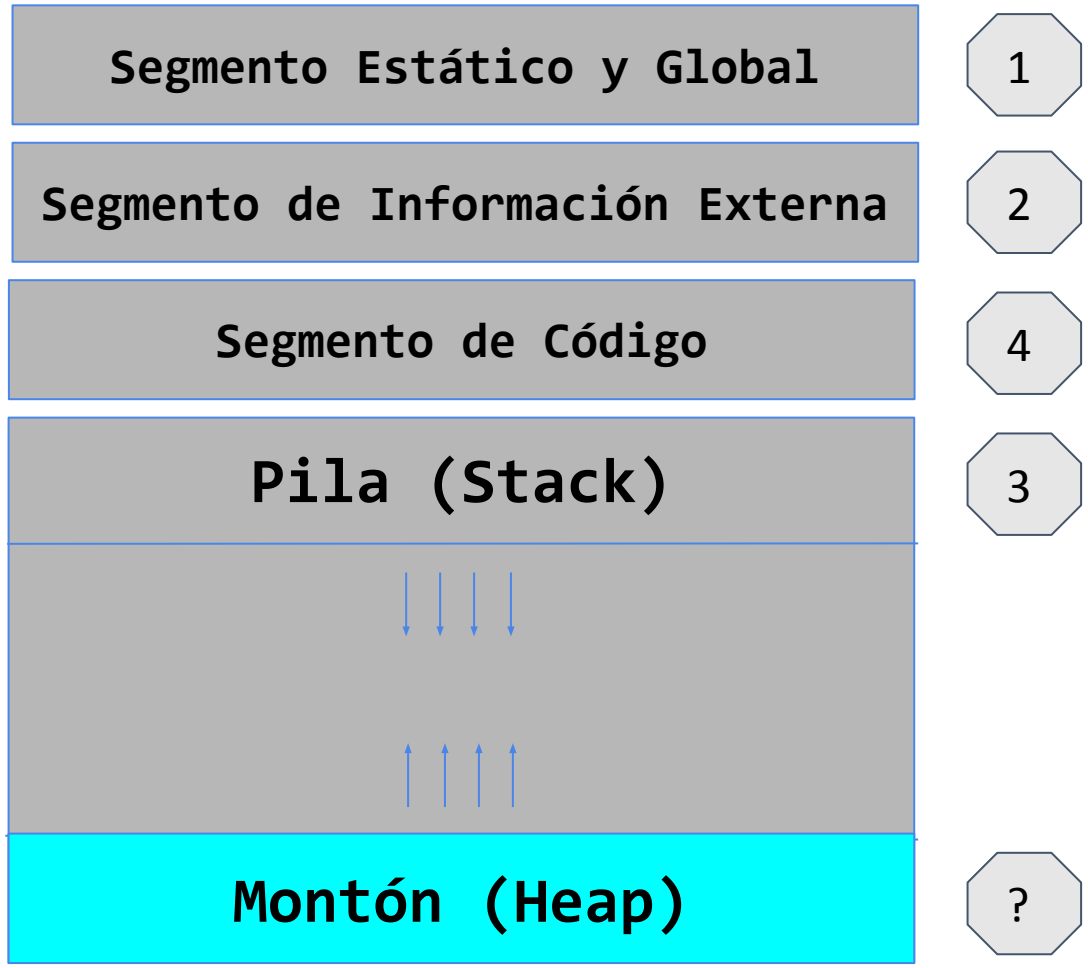
```
void f3(int * pn)
{
  ++*pn;
  pn = nullptr;
}
```

```
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(*pp);   // 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

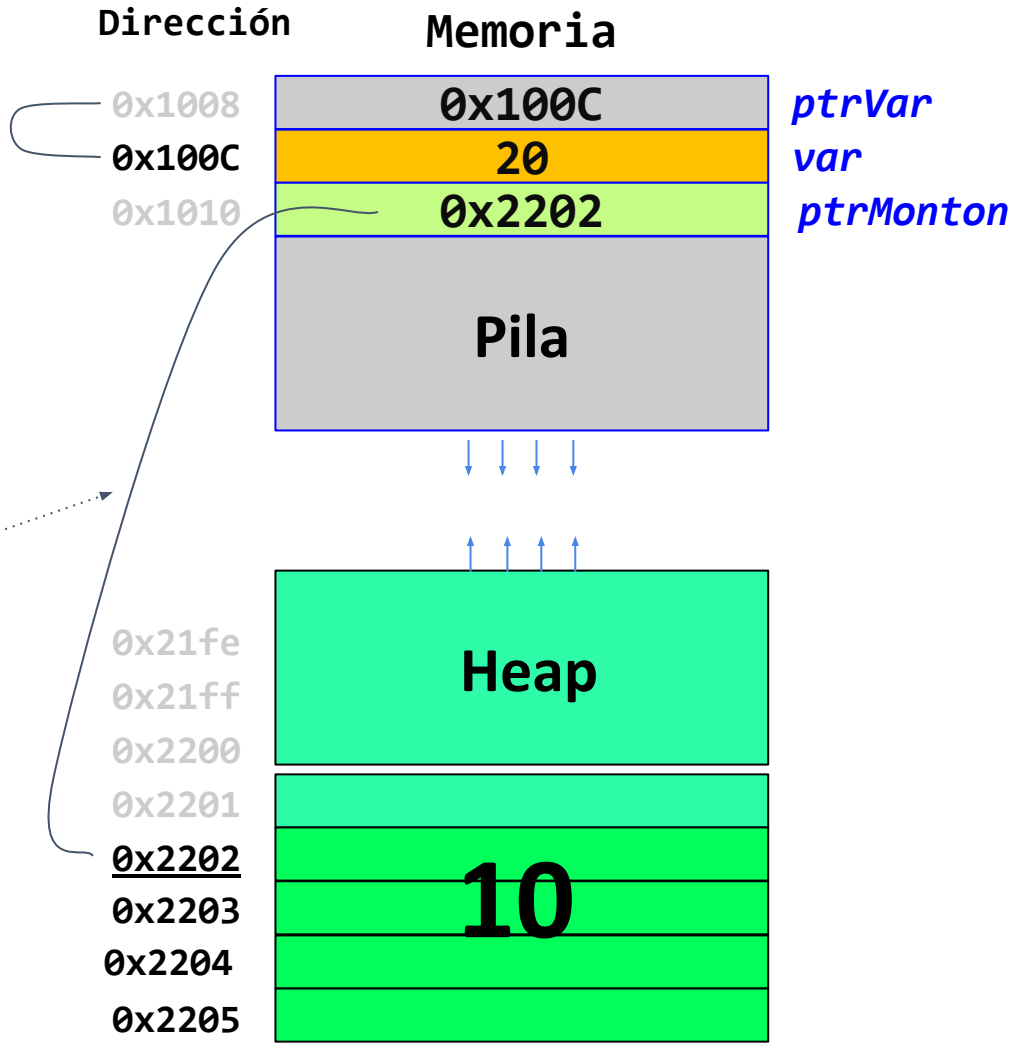
Liberando memoria dinámica:

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 almacenen utilizando memoria dinámica y luego halle la suma, la diferencia y el producto de estos números.

```
#include <iostream>
using namespace std;
int main()
{  double *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. Si se supone 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?

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 is 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};
```

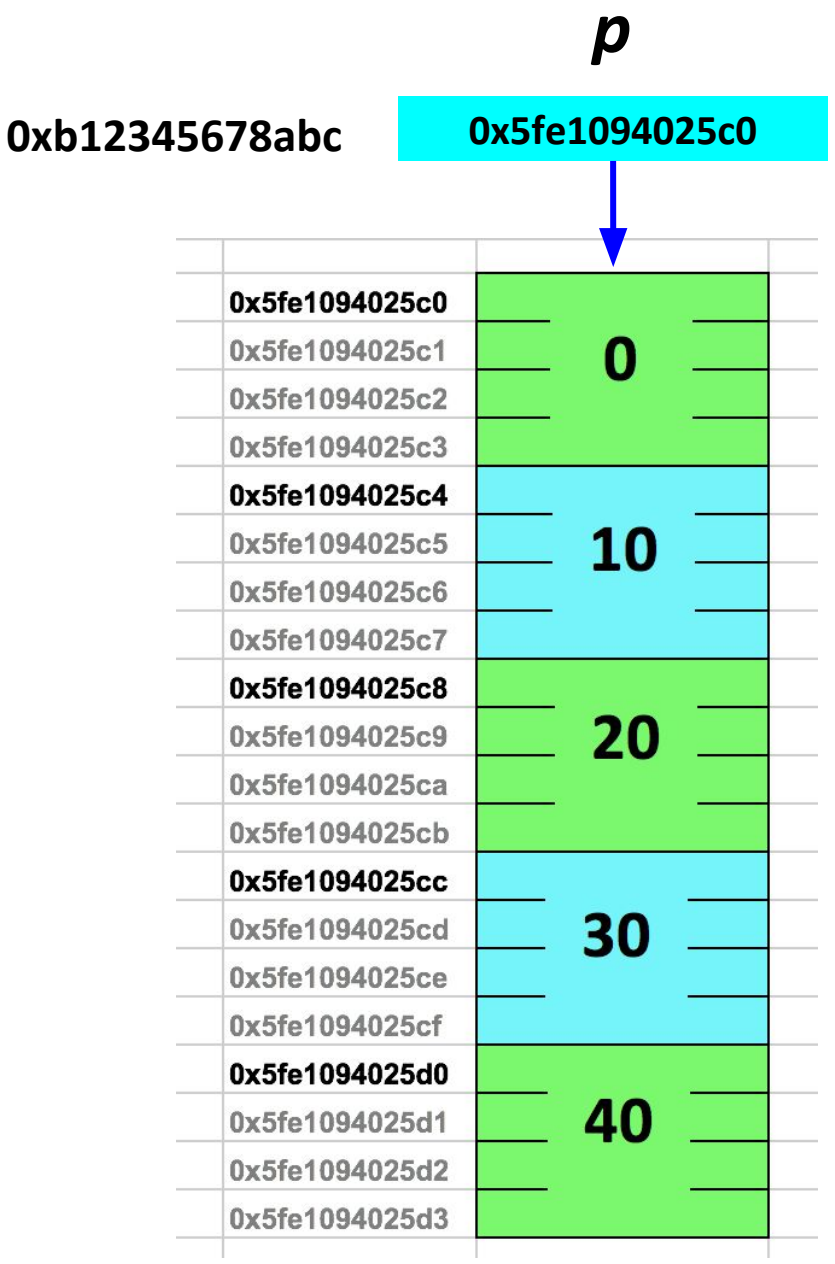
¿

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.**

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
```

En la solución se utiliza los siguientes archivos.

Main.cpp
UFunciones.h
UFunciones.cpp

Solución 1 - Usando array dinámicos

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 MULITPLOS_VERSION1_UFUNCIONES_H
#define MULITPLOS_VERSION1_UFUNCIONES_H

#include <iostream>
#include <cstddef>
#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 //MULITPLOS_VERSION1_UFUNCIONES_H
```

UFunciones.cpp -parte 1

```
#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];
}
```

UFunciones.cpp -parte 2

```
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

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;
}
```

Solución 3:
Usando vector y funciones Lambda

main.cpp

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

int main()
{unsigned long n=0;
 random_device rd;
 cout << "Numero de elementos : ";
 cin >> n;

vector<int> a(n);
 //--- llenamos el vector usando una función lambda
 for_each(begin(a),end(a), [&rd](int & item){
   item = rd()%1000;
 });
 cout << "\nVector \n\n";
 Imprimir(a);

vector<int> m5;
 int mul=5;
 //--- Se usa una funcion Lambda
 copy_if(begin(a), end(a),back_inserter(m5),[mul](int item){
    return item%mul==0;
 });

 cout << "\n\nVector con multiplos de 5 \n";
 Imprimir(m5);

 vector<int> m7;
 mul=7;
 //--- Se usa una función Lambda
 copy_if(begin(a), end(a),back_inserter(m7),[mul](int item){
    return item%mul==0;
 });

 cout << "\n\nVector con multiplos de 7 \n";
 Imprimir(m7);
 return 0;
}
```

UFunciones.h

```
#ifndef MULTIPLOS_VERSION3_UFUNCIONES_H
#define MULTIPLOS_VERSION3_UFUNCIONES_H

#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;

void  Imprimir(vector<int> v);


#endif //MULTIPLOS_VERSION3_UFUNCIONES_H
```

UFunciones.cpp

```
#include "UFunciones.h"

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

Solución 4:
Usando vector y funciones Lambda - 2

main.cpp

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

int main()
{ unsigned long n=0;
 random_device rd;

 cout << "Numero de elementos : ";
 cin >> n;
 vector<int> a(n);
 //--- llenamos el vector
 for_each(begin(a),end(a), [&rd](int & item){
    item = rd()%1000;
 });

 cout << "\nVector \n\n";
 Imprimir(a);

unsigned long mul=5;
 vector<int> m5= GeneraVectorconMultiplos(a, mul);
 cout << "\nVector con multiplos de 5 \n\n";
 Imprimir(m5);

 mul=7;
 vector<int> m7 =GeneraVectorconMultiplos(a, mul);
 cout << "\nVector con multiplos de 7 \n\n";
 Imprimir(m7);
 return 0;
}
```

UFunciones.h

```
#ifndef MULTIPLOS_VERSION4_UFUNCIONES_H
#define MULTIPLOS_VERSION4_UFUNCIONES_H

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

using namespace std;

void Imprimir(vector<int> v);
vector<int> GeneraVectorconMultiplos(vector<int> &a, unsigned long mul);

#endif //MULTIPLOS_VERSION4_UFUNCIONES_H
```

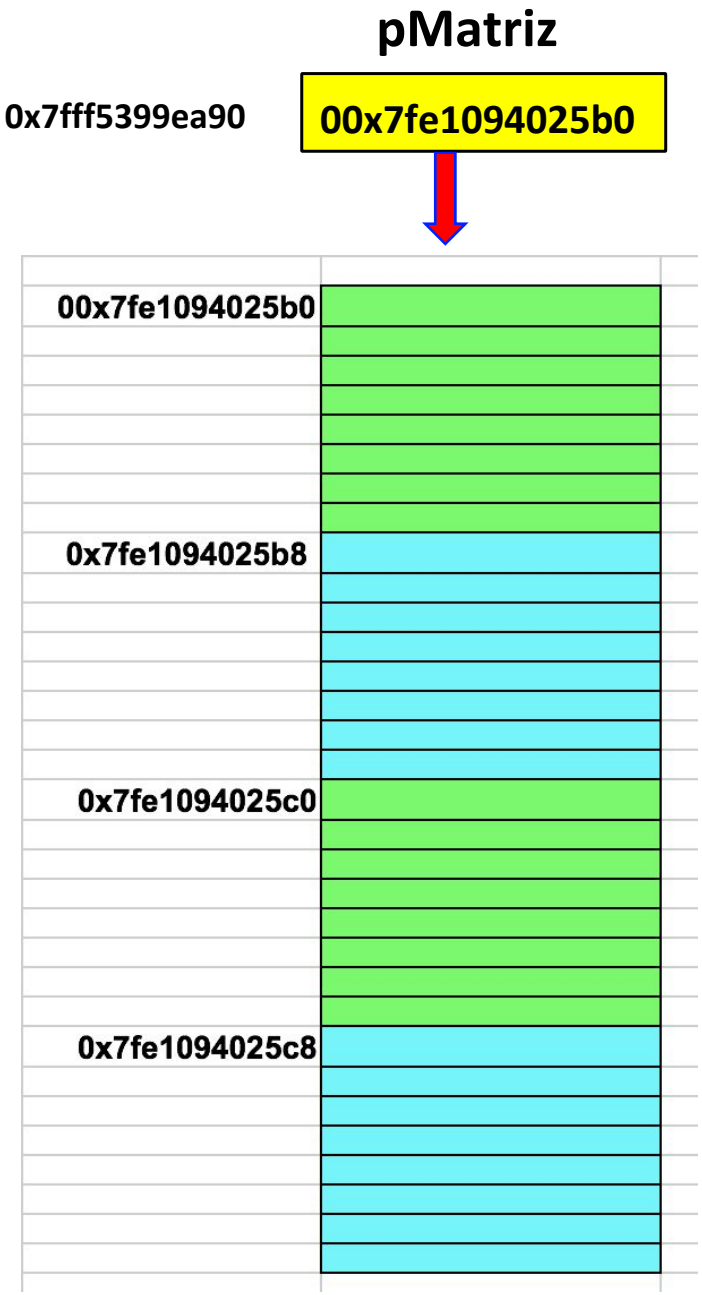
UFunciones.cpp

```
#include "UFunciones.h"

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

vector<int> GeneraVectorconMultiplos(vector<int> &a, unsigned long mul)
{//-------------------------------------------------------------------------------------------
 vector<int> vecmul;
 copy_if(begin(a), end(a), back_inserter(vecmul), [mul](int item) {
     return item%mul == 0;
 });
/*----También se puede resolver así: ----
for_each(begin(a),end(a),[&vecmul, mul](int item) {
 if(item%mul == 0)
   vecmul.push_back(item);
});
-------------------------------------------------------------*/
 return vecmul;
}
```

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

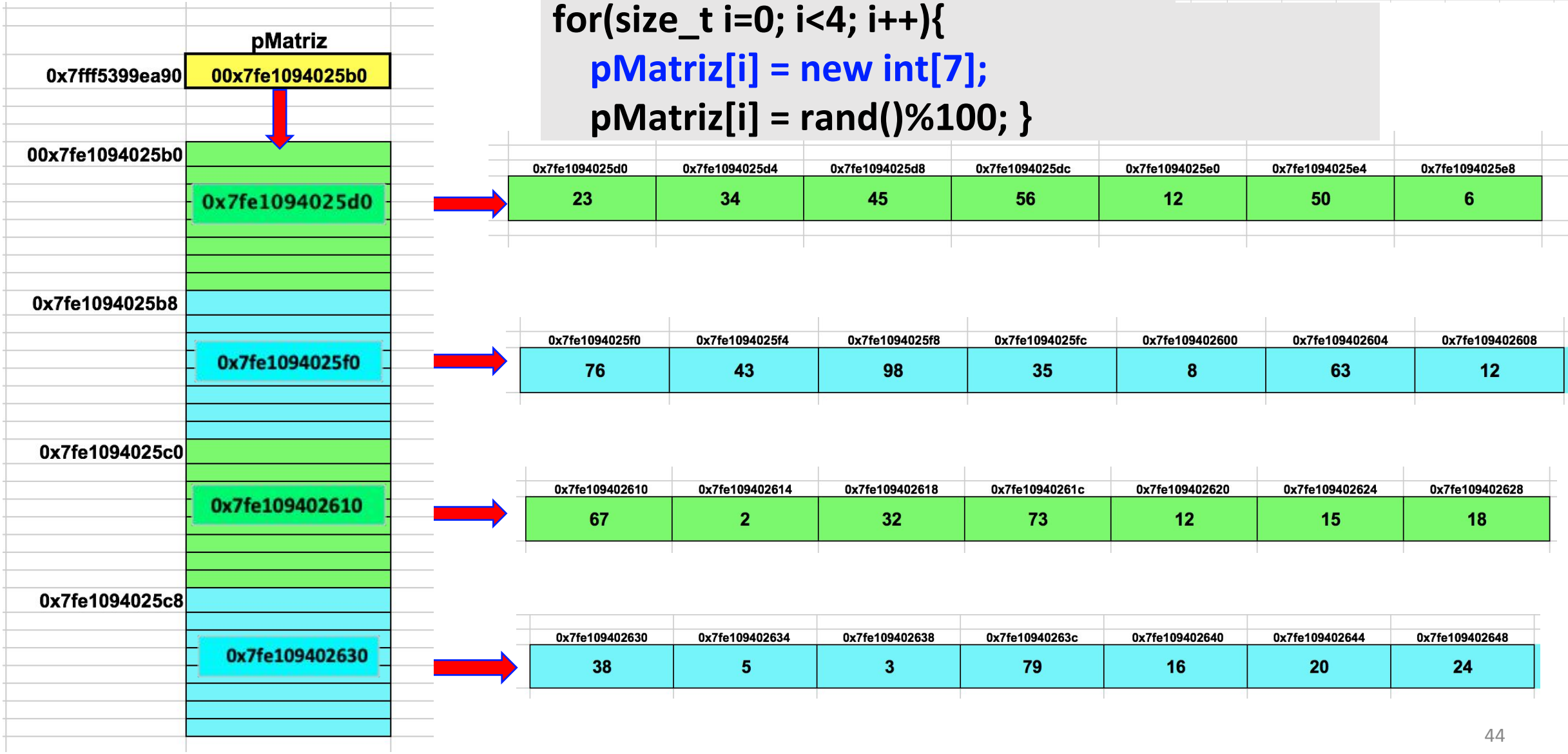



```
int **pMatriz= nullptr;
```

```
pMatriz = new int*[4];
```

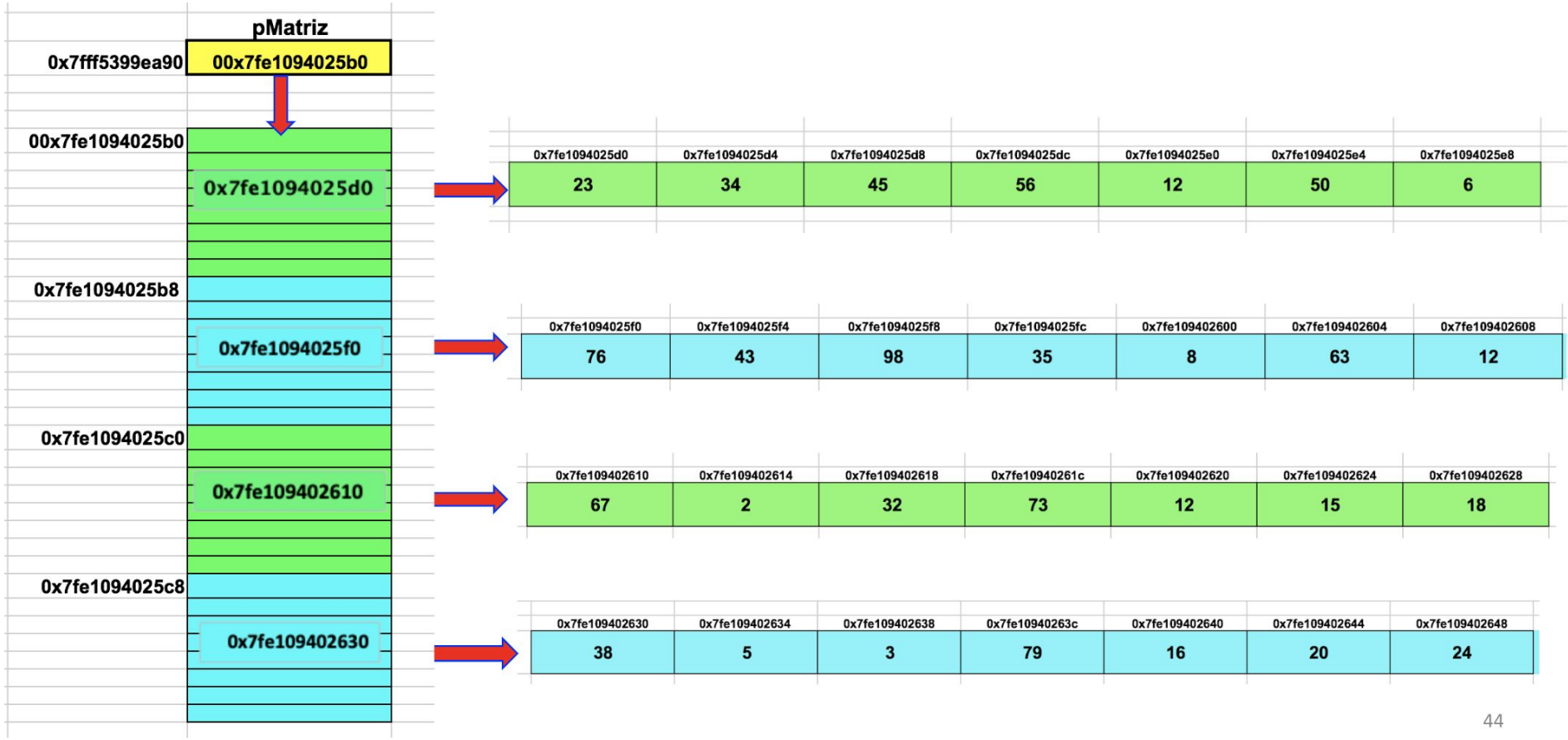
```
int **pMatriz= nullptr;
pMatriz = new int*[4];
for(size_t i=0; i<4; i++){
    pMatriz[i] = new int[7];
    pMatriz[i] = rand()%100; }
```

	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.

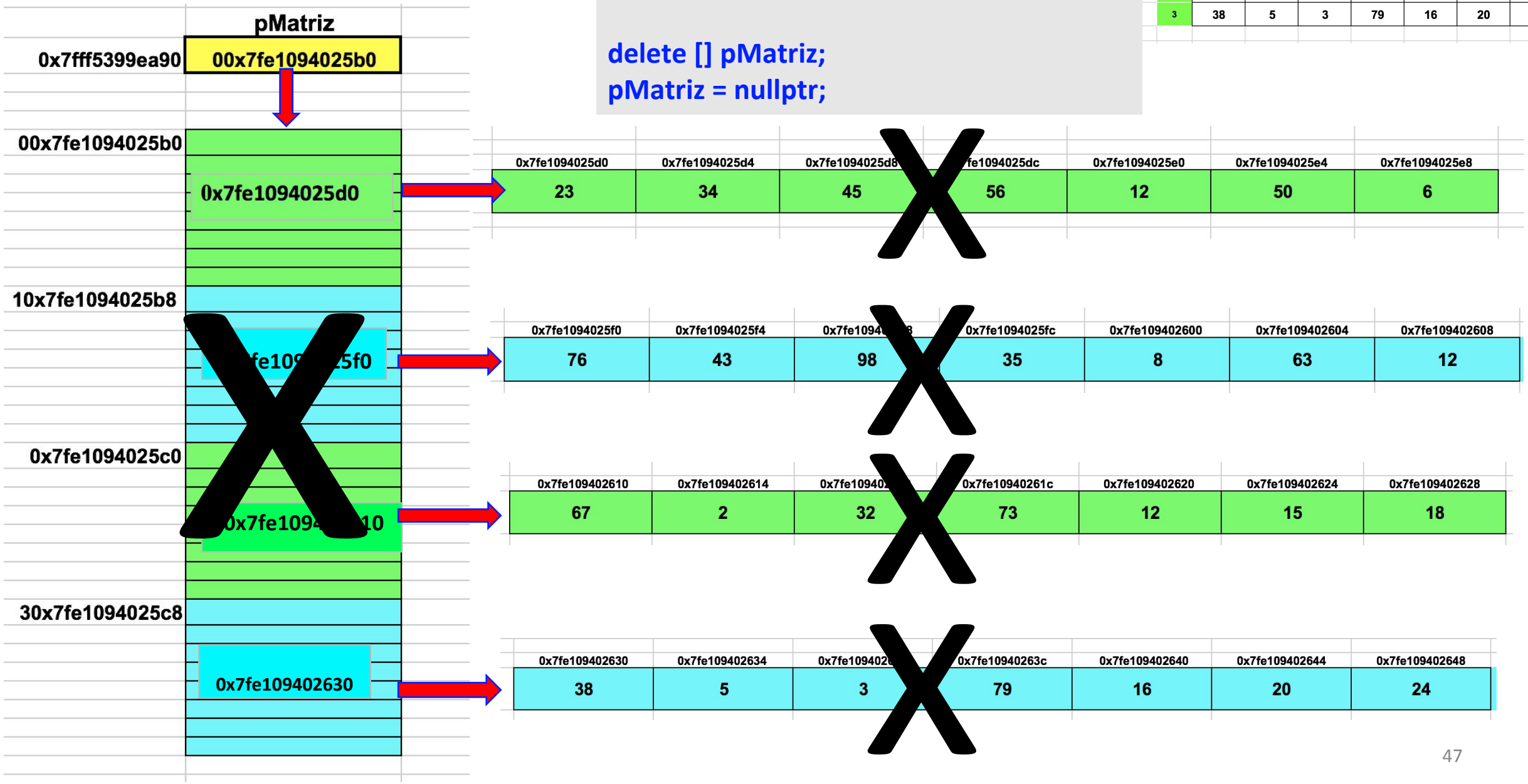
¿ Cuántos bytes en total ocupa la matriz?



Para liberar memoria

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