Programación Orientada a Objetos

Teoría 01 - 18/08/2011 - Pablo Novara

Programación Orientada a Objetos

- Web: http://e-fich.unl.edu.ar (clave: POO2011)

- Software a utilizar:
  - Unidades 1 a 8: Cualquier *cosa* para **C++** que medianamente cumpla con el standard
    - MinGW Developer Studio, DevCpp, CodeBlocks
    - Microsoft Visual Studio, Borland C++ Builder,
    - Anjuta, Kdevelop, Emacs, Bloc de notas, etc...

  Sugerencia: ZinjaI – http://zinjai.sourceforge.net

- Unidad 9 y Proyecto Final: ¡¿Proyecto Final?!

Programación Orientada a Objetos

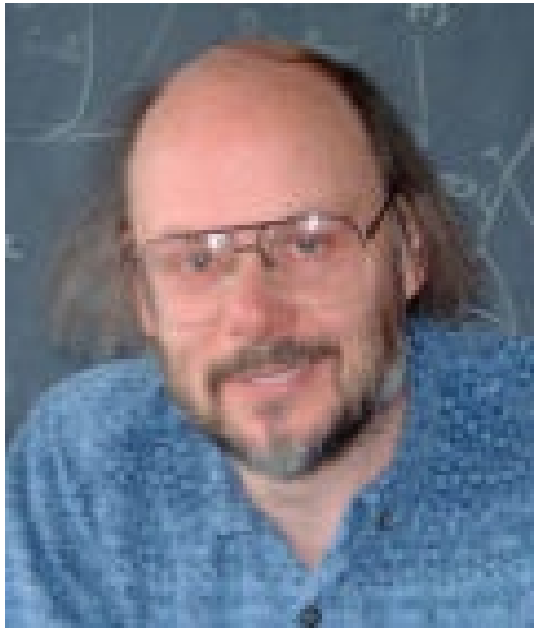
- Evaluación: 2 parciales + Proyecto Final

- Cómo Regularizar
  - Promedio 50 en parciales + 80% de asistencia
  - Deberán rendir final escrito y presentar PF

- Cómo Promocionar
  - Promedio 80 en parciales + 80% de asistencia
  - Deben presentar igualmente PF en fecha de final

- Ambos parciales valen lo mismo
- Se puede recuperar un parcial (también para promoción)

"C++ hace que sea más difícil dispararte en el pie, pero cuando lo hacés te vuela la pierna completa"



"C++ hace que sea más difícil dispararte en el pie, pero cuando lo hacés te vuela la pierna completa"

Bjarne Stroustrup, *creador* de C++

1- Punteros
2- Introducción a la POO
3- Relaciones entre clases
4- Sobrecarga de Operadores
4.5- Implementación de pilas y colas
Primer Parcial

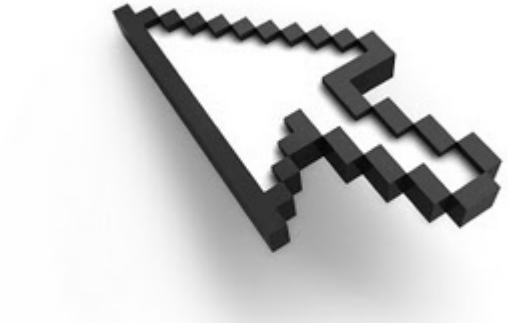
5- Objetos String
6- Flujos de I/O (archivos)
7- Programación Genérica (Templates)
8- Biblioteca STL
Segundo Parcial

9- Diseño de Interfaces Gráficas
10- Proyecto Final

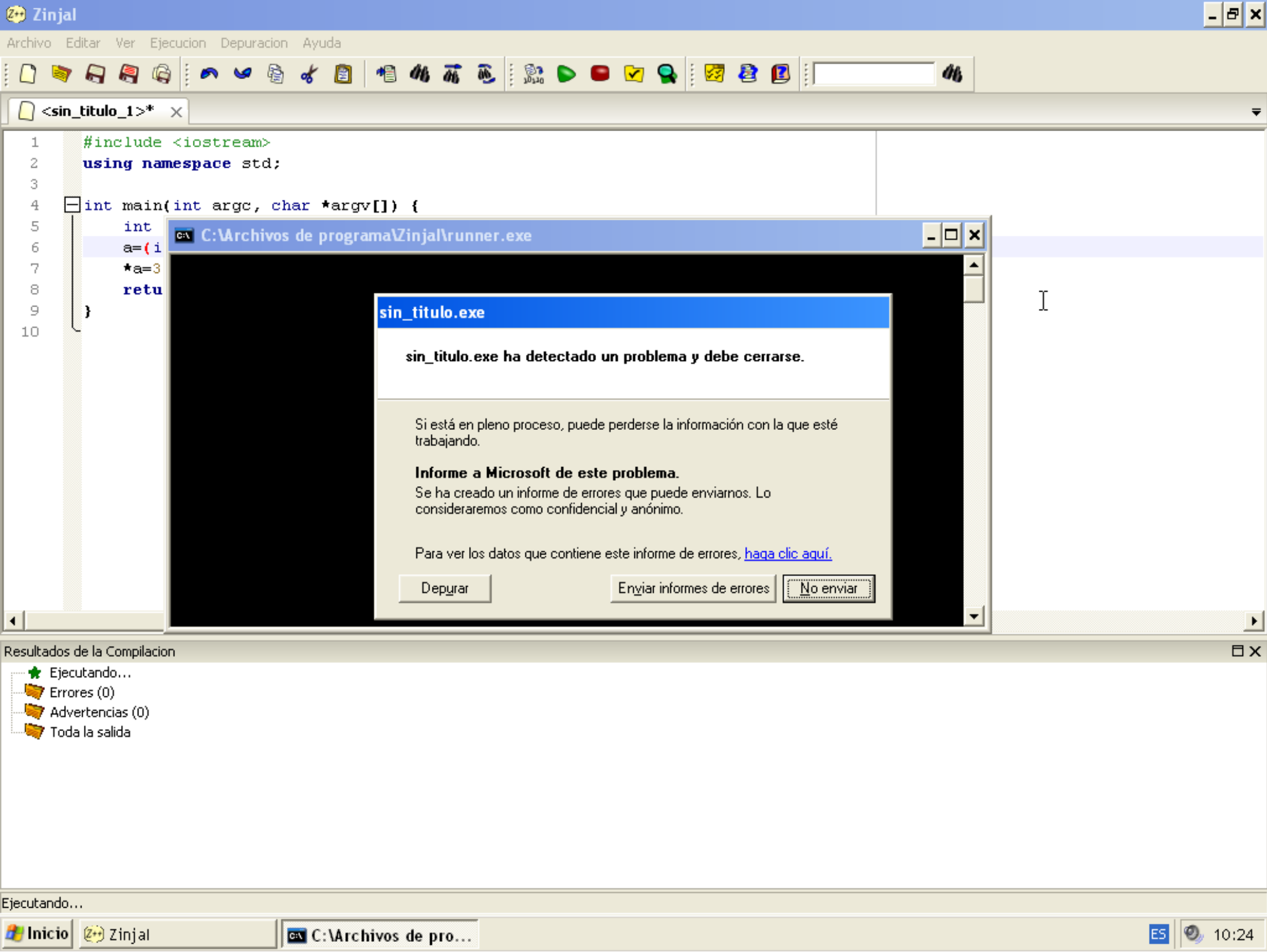
Programación Orientada a Objetos

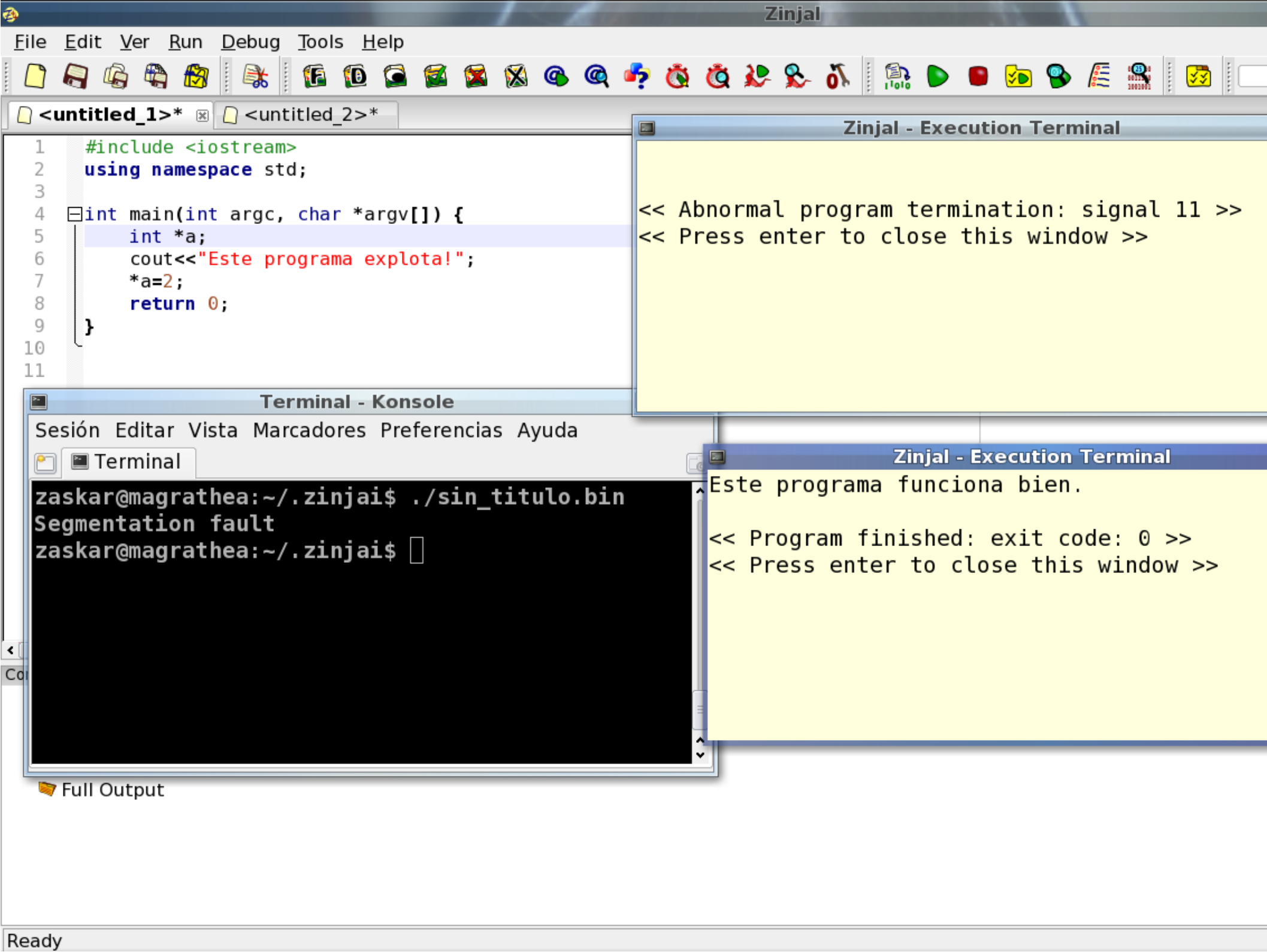
Unidad 1: Punteros

Teoría 01 - 18/08/2011 - Pablo Novara

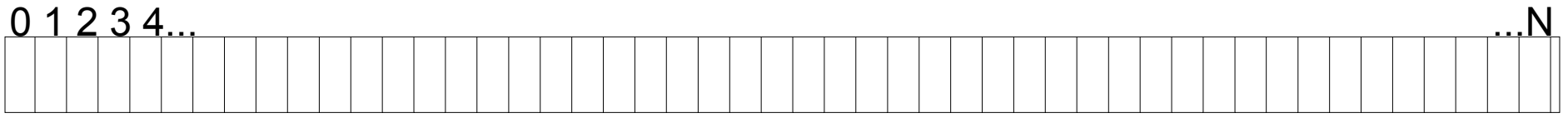



Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) +
00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) +
00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will
lose any unsaved information in all applications.
Press any key to continue


Zinjal
Archivo Editar Ver Ejecucion Depuracion Ayuda
<sin_titulo_1>*
1 #include <iostream>
2 using namespace std;
3
4 int main(int argc, char *argv[]) {
5 int
6 a=(i
7 *a=3
8 retu
9 }
10
C:\Archivos de programa\Zinjal\runner.exe
sin_titulo.exe
sin_titulo.exe ha detectado un problema y debe cerrarse.
Si está en pleno proceso, puede perderse la información con la que esté trabajando.
Informe a Microsoft de este problema.
Se ha creado un informe de errores que puede enviarnos. Lo consideraremos como confidencial y anónimo.
Para ver los datos que contiene este informe de errores, haga clic aquí.
Depurar
Enviar informes de errores
No enviar
Resultados de la Compilacion
Ejecutando...
Errores (0)
Advertencias (0)
Toda la salida
Ejecutando...
Inicio
Zinjal
C:\Archivos de pro...
ES
10:24


Zinjal
File Edit Ver Run Debug Tools Help
<untitled_1>*
<untitled_2>*
#include <iostream>
using namespace std;
int main(int argc, char *argv[]) {
int *a;
cout<<"Este programa explota!";
*a=2;
return 0;
}
Zinjal - Execution Terminal
<< Abnormal program termination: signal 11 >>
<< Press enter to close this window >>
Terminal - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
Terminal
zaskar@magrathea:~/.zinjai$ ./sin_titulo.bin
Segmentation fault
zaskar@magrathea:~/.zinjai$
Zinjal - Execution Terminal
Este programa funciona bien.
<< Program finished: exit code: 0 >>
<< Press enter to close this window >>
Full Output
Ready

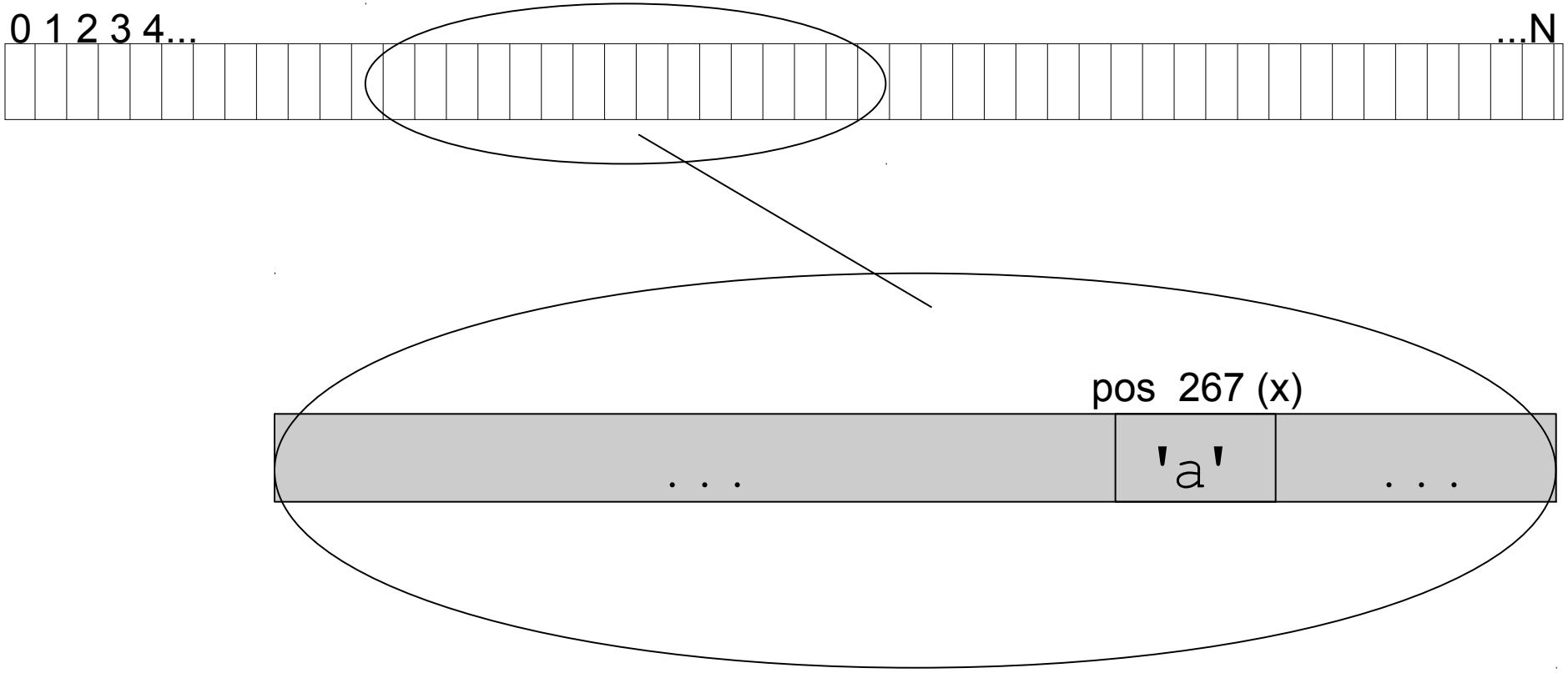
¿Que #@!*~# es un puntero?



¿Que #@!*~# es un puntero?

```
char x='a';
```

Variable tipo char: pedacito de memoria que contiene un caracter

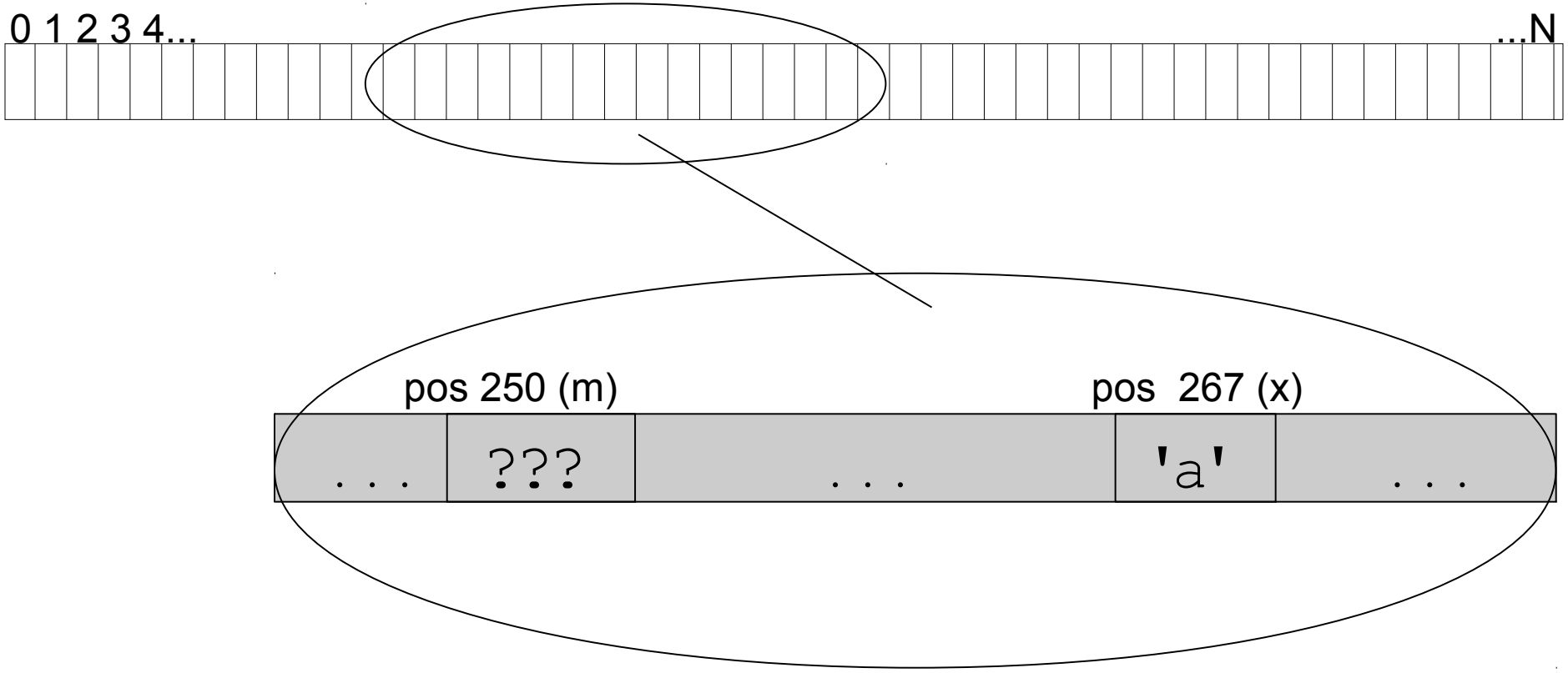


¿Que #@!*~# es un puntero?

```
char x='a';
char *m;
```

Variable tipo char: pedacito de memoria que contiene un caracter

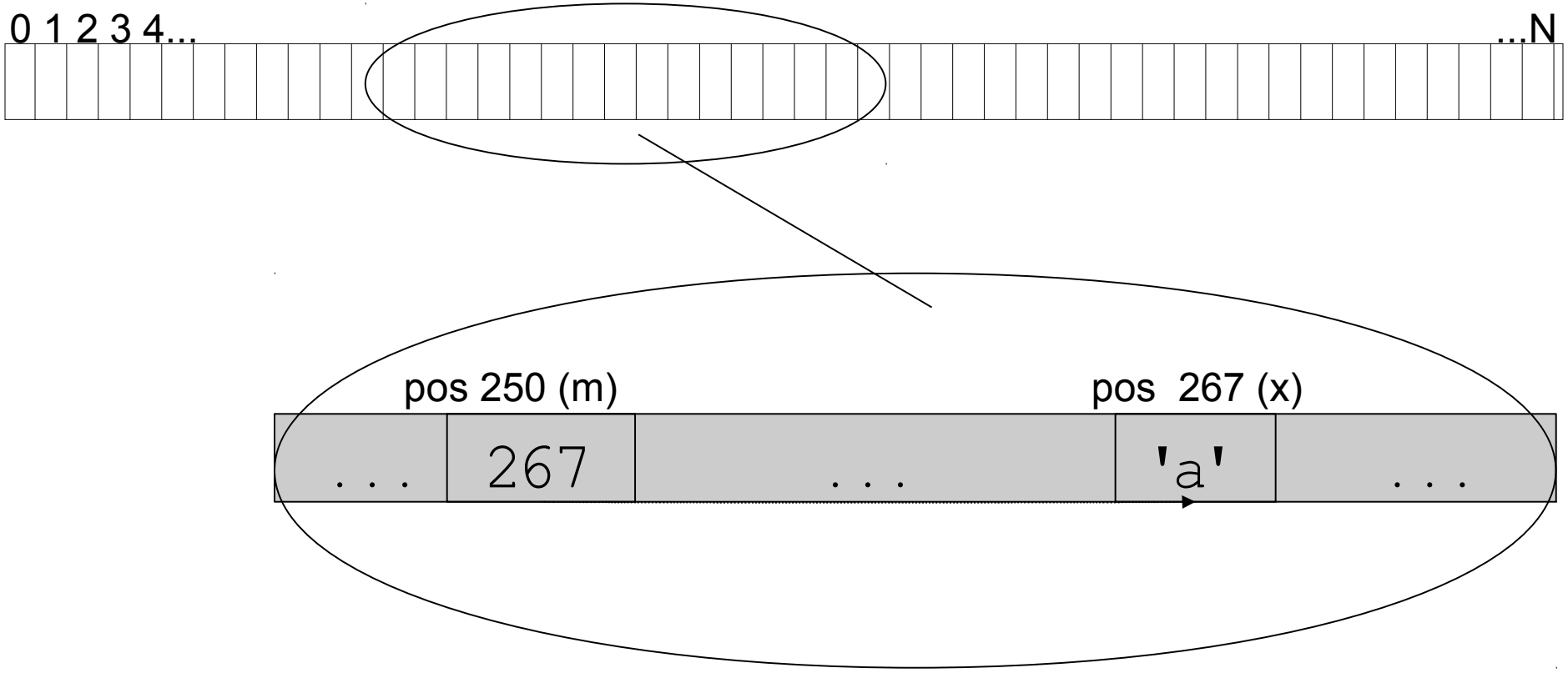
Puntero a char: contiene la dirección de memoria donde se guarda un caracter



¿Que #@!*~# es un puntero?

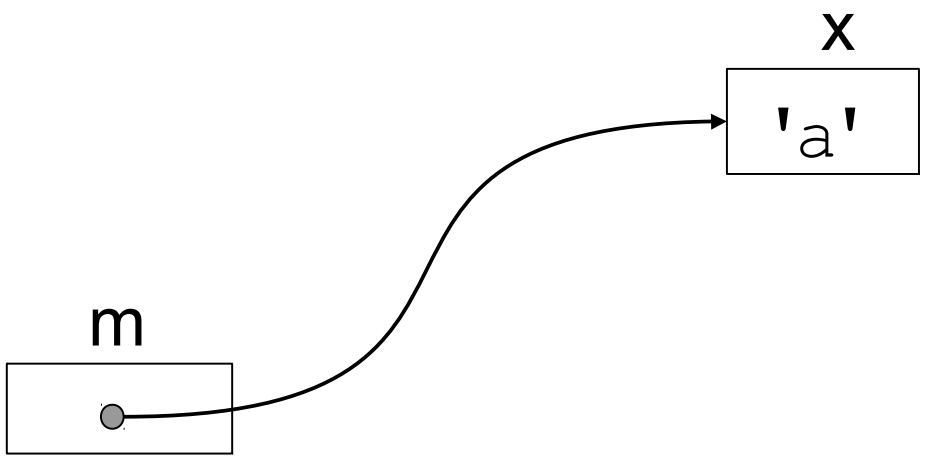
```
char x='a';
char *m;
m = &x;
```

Puntero a char: contiene la dirección de memoria donde se guarda un caracter



¿Que #@!*~# es un puntero?

```
char *m;
```



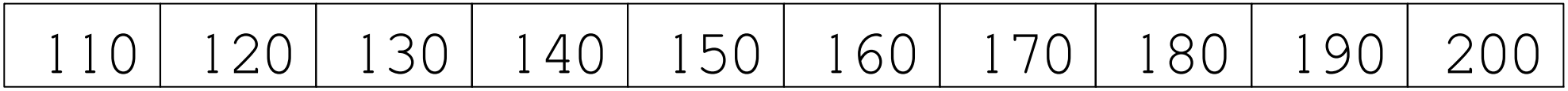
Notación y Operadores: Lo mismo pero distinto

- En una declaración
  - *: declara puntero (int *x;) (SÓLO PUNTERO!)
  - &: declara alias (int &y;)

- En una expresión:
  - * = desreferencia un puntero (cout<<(*x);)
  - & = referencia una variable (x=&y;)
  - + y -: aritmetica de punteros (¿arit-qué?)

Punteros y Arreglos

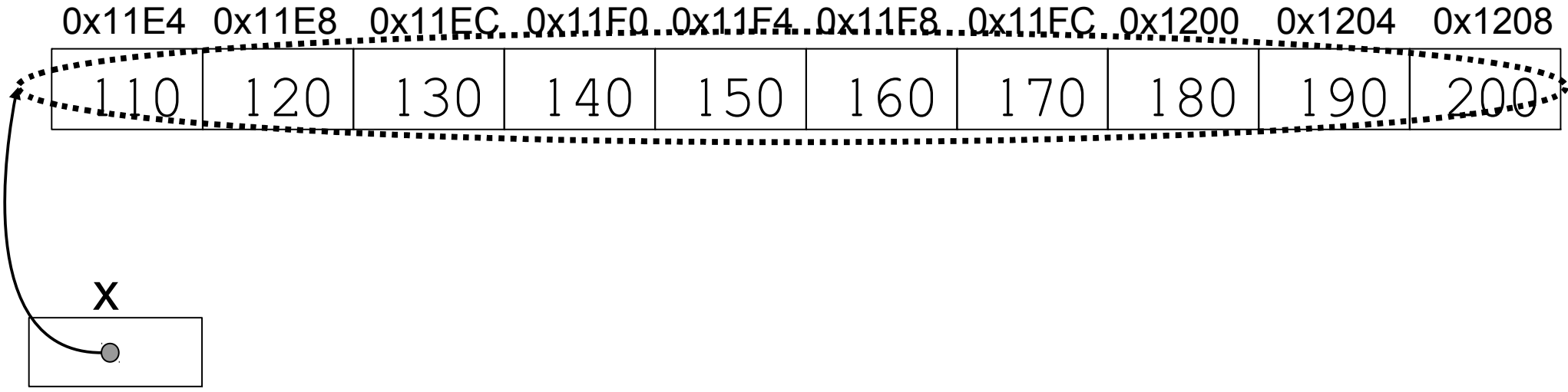
```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

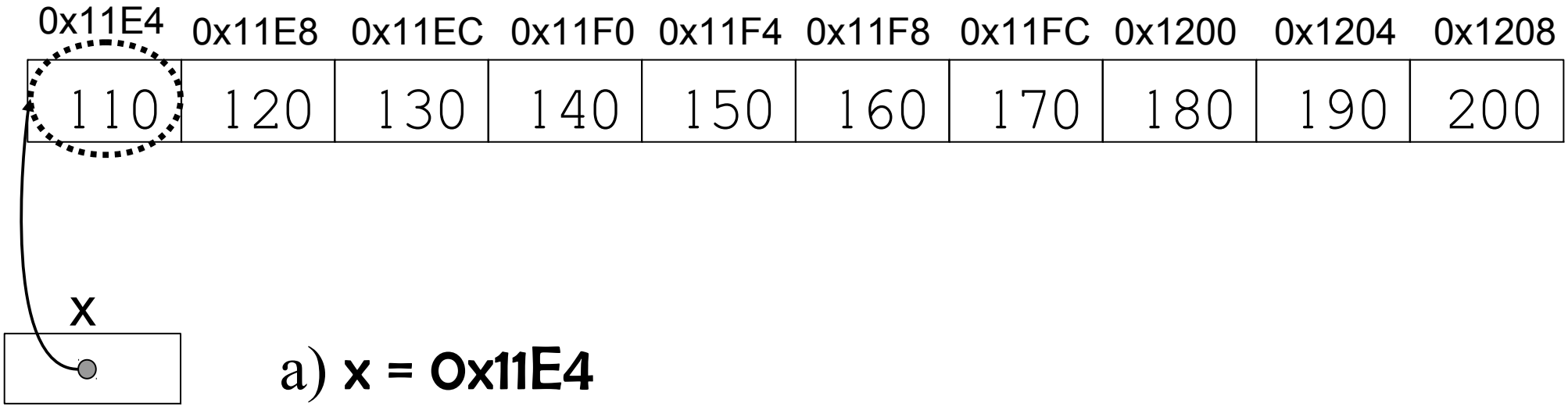
El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:

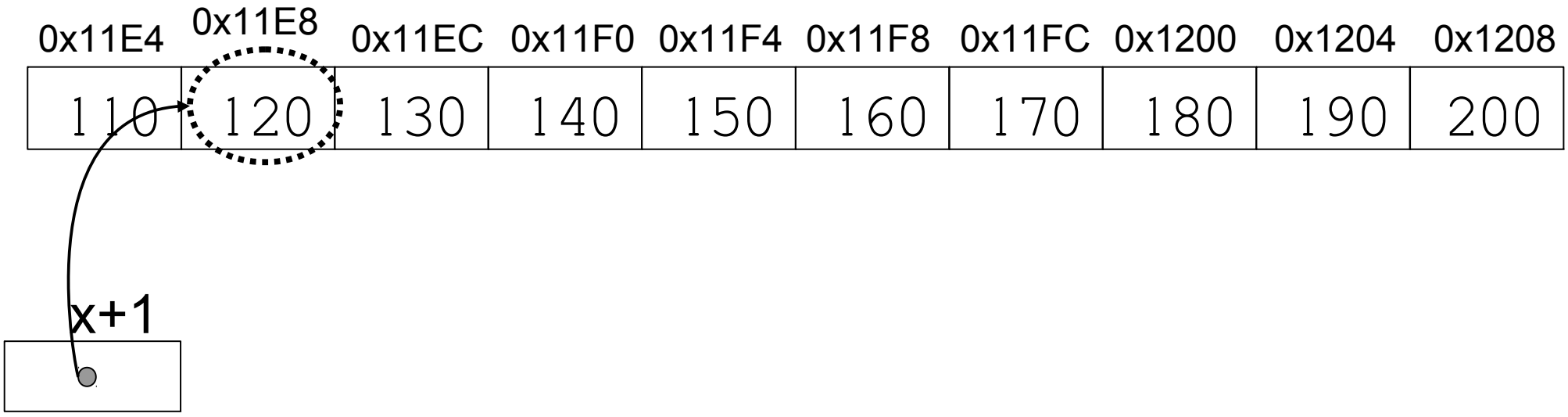


a) **x = 0x11E4**

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

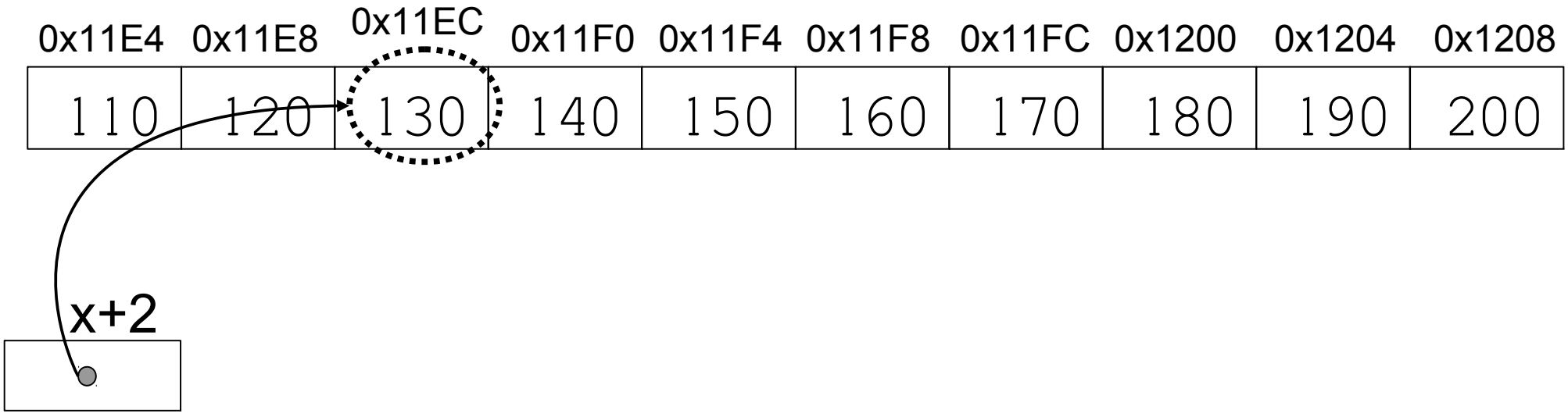
El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

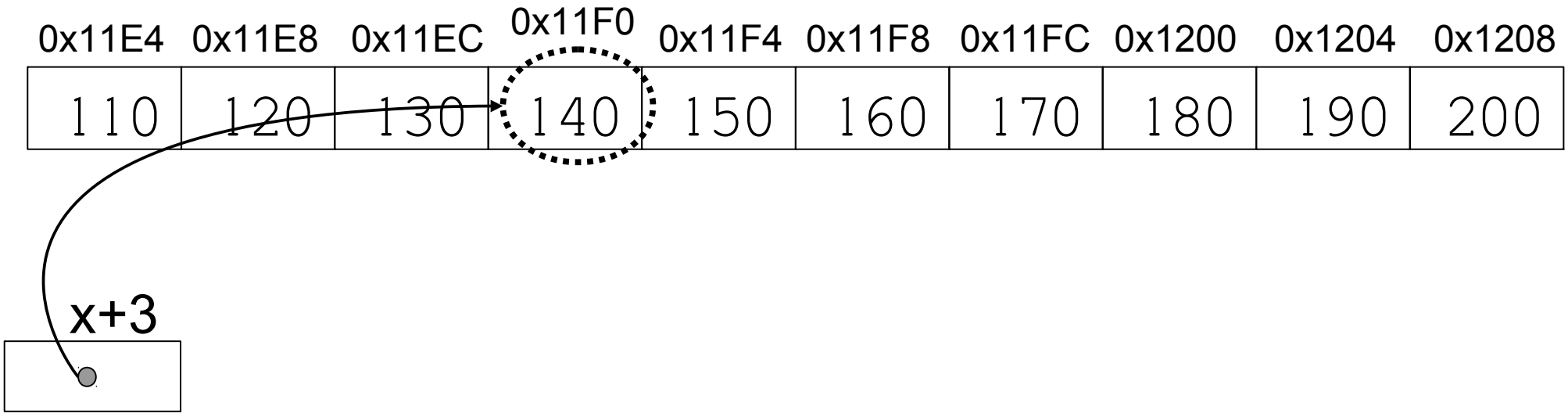
El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

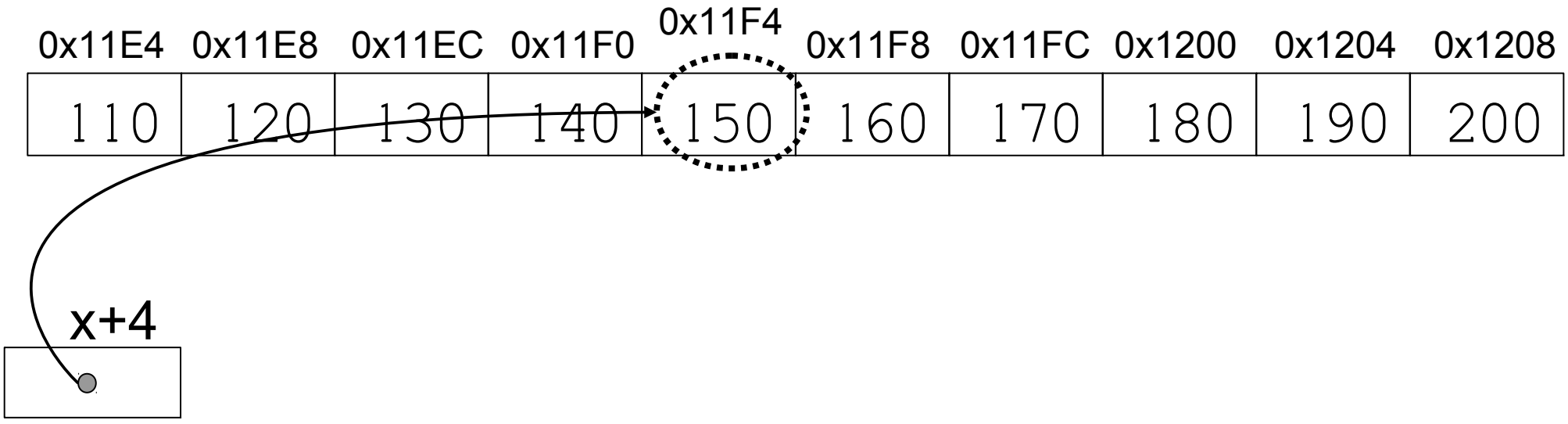
El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

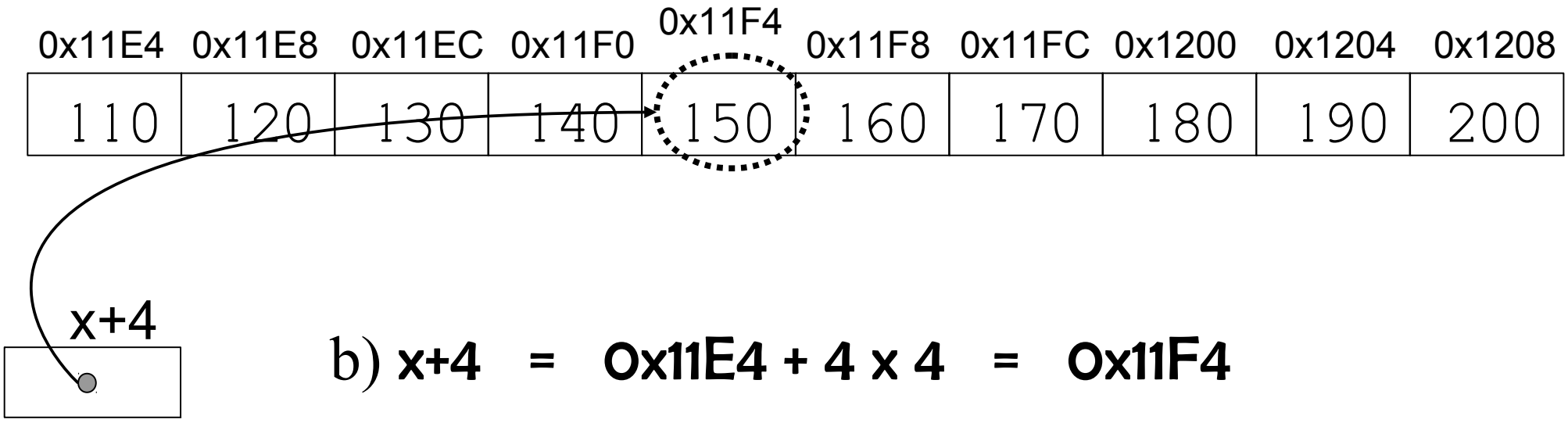
El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:

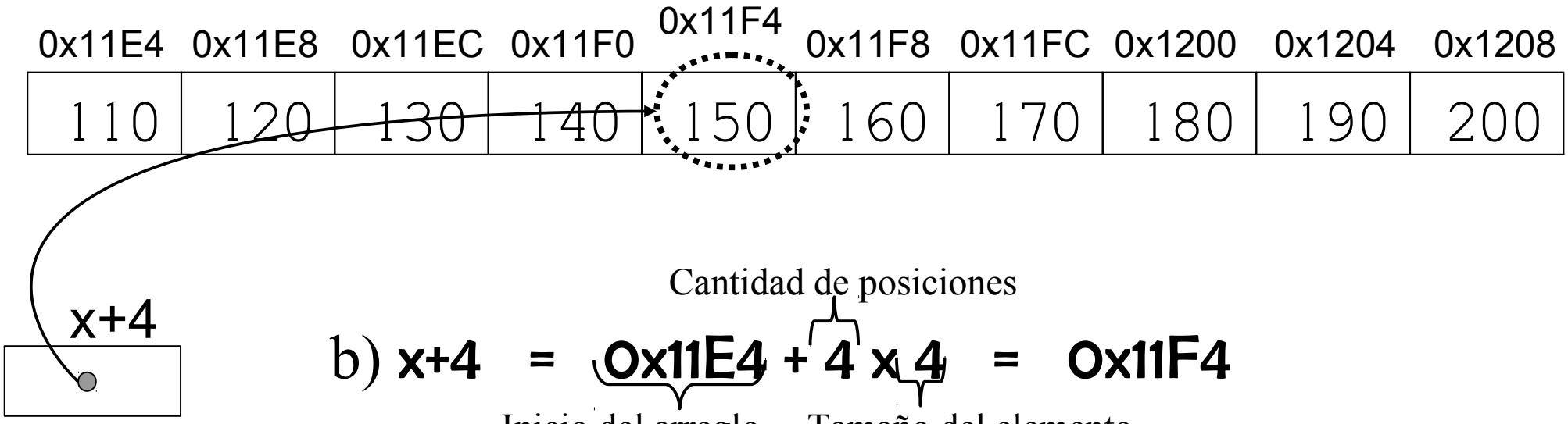


b) **x+4 = 0x11E4 + 4 x 4 = 0x11F4**

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



b) **x+4 = 0x11E4 + 4 x 4 = 0x11F4**

Cantidad de posiciones

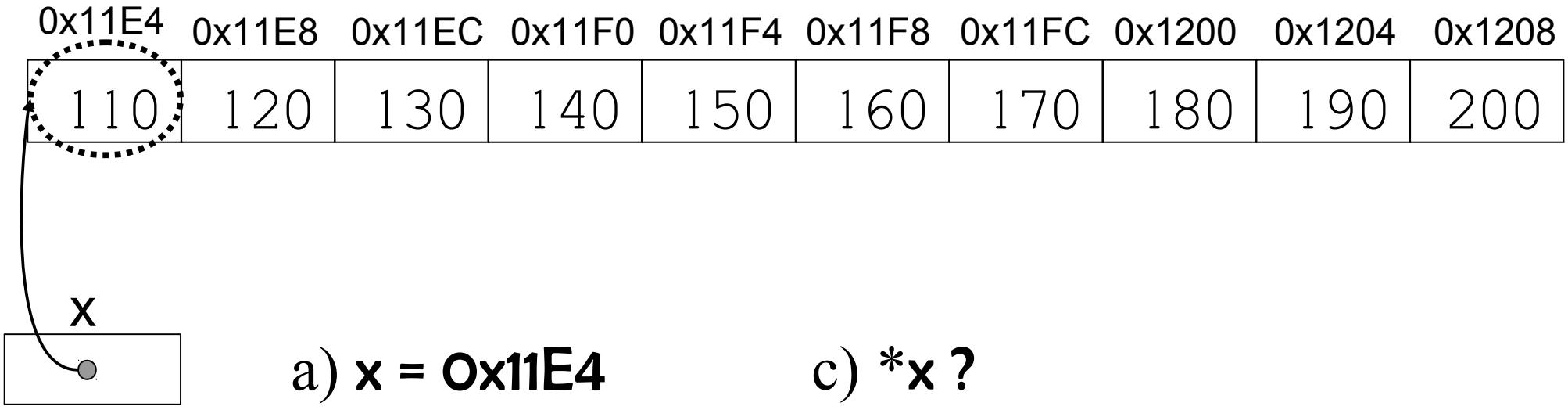
Inicio del arreglo

Tamaño del elemento (entero = 4 bytes)

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



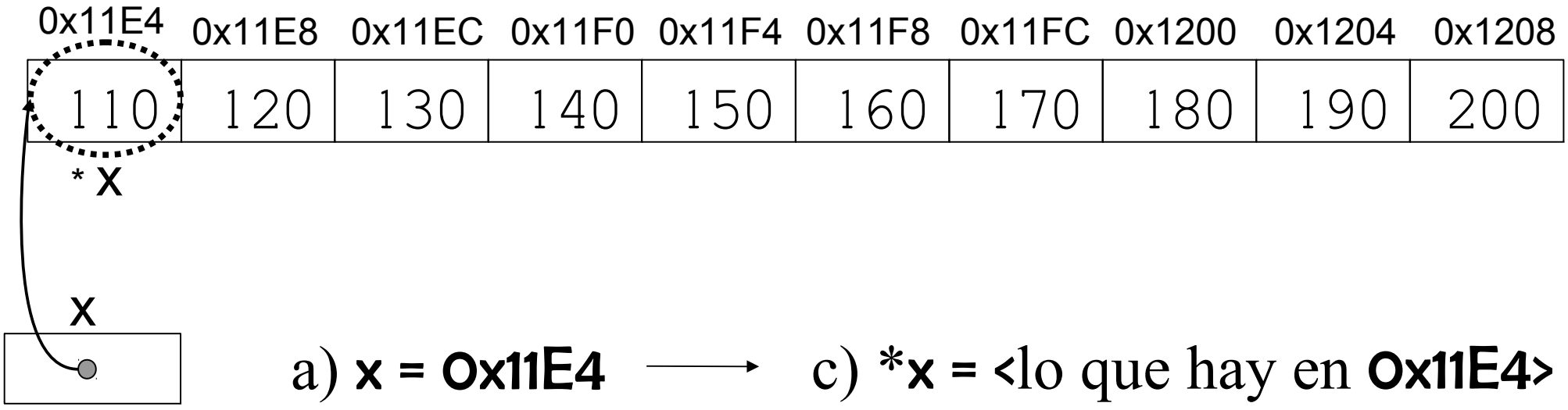
a) **x = 0x11E4**

c) ***x ?**

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:

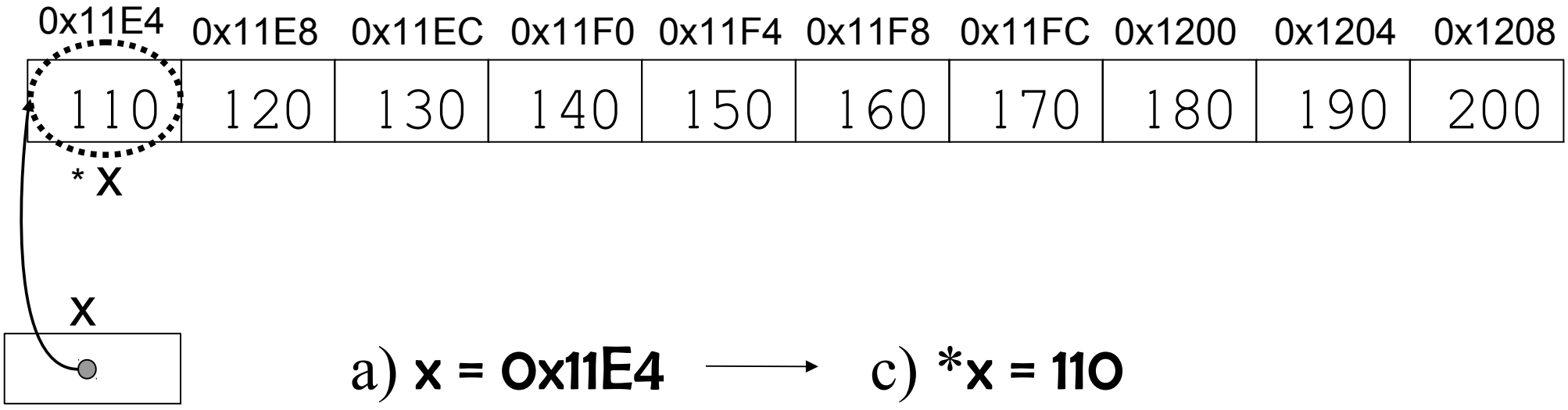


a) **x = 0x11E4** ⟶ c) ***x** = <lo que hay en **0x11E4**>

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:

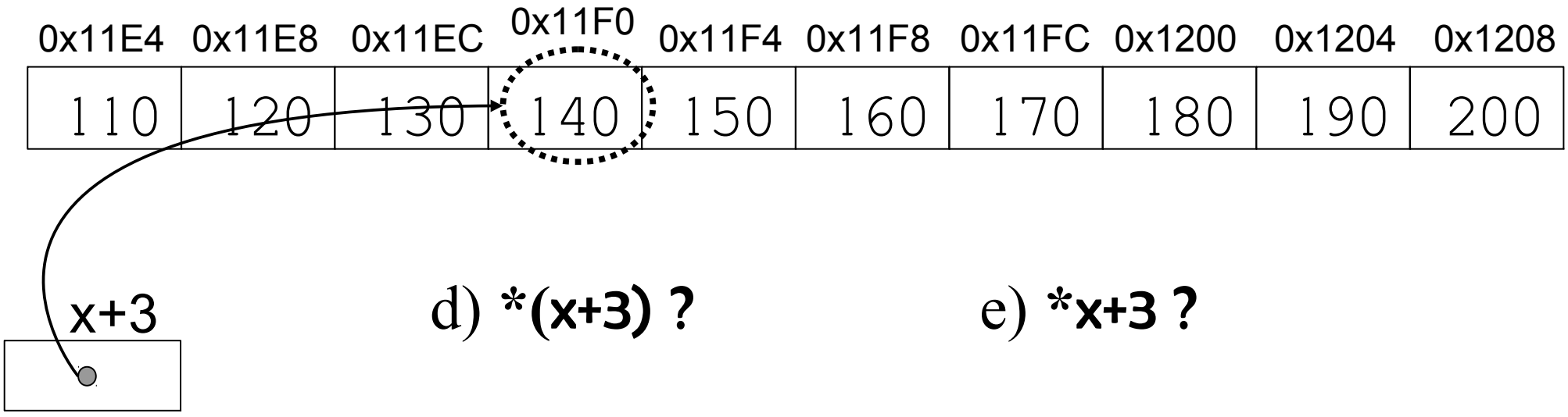


a) **x = 0x11E4** ⟶ c) ***x = 110**

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



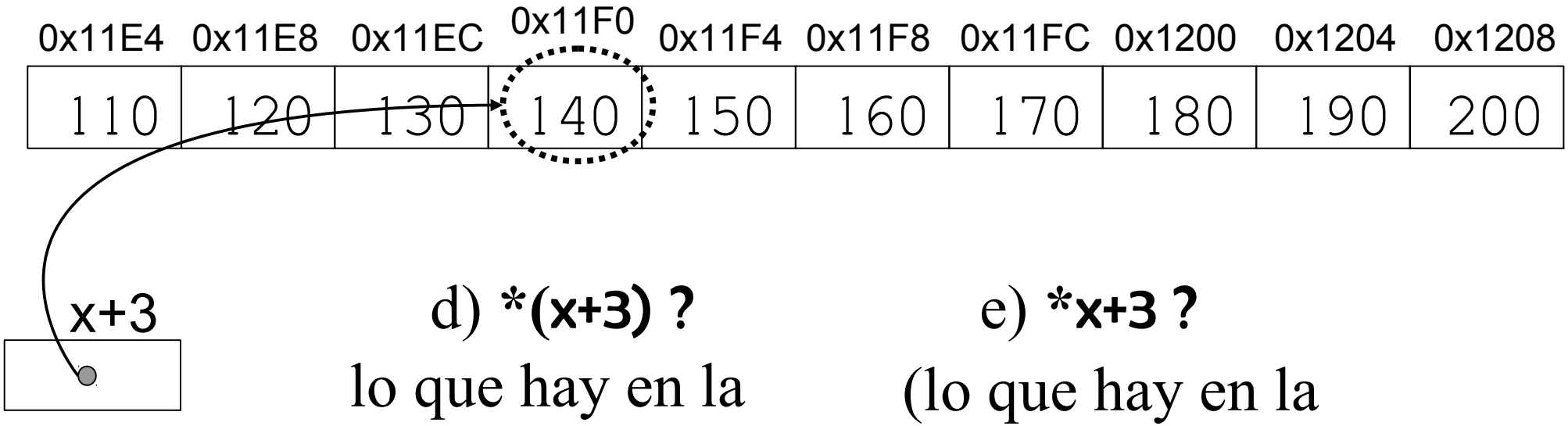
d) ***(x+3) ?**

e) ***x+3 ?**

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



d) ***(x+3) ?**

lo que hay en la direccion (x+3)

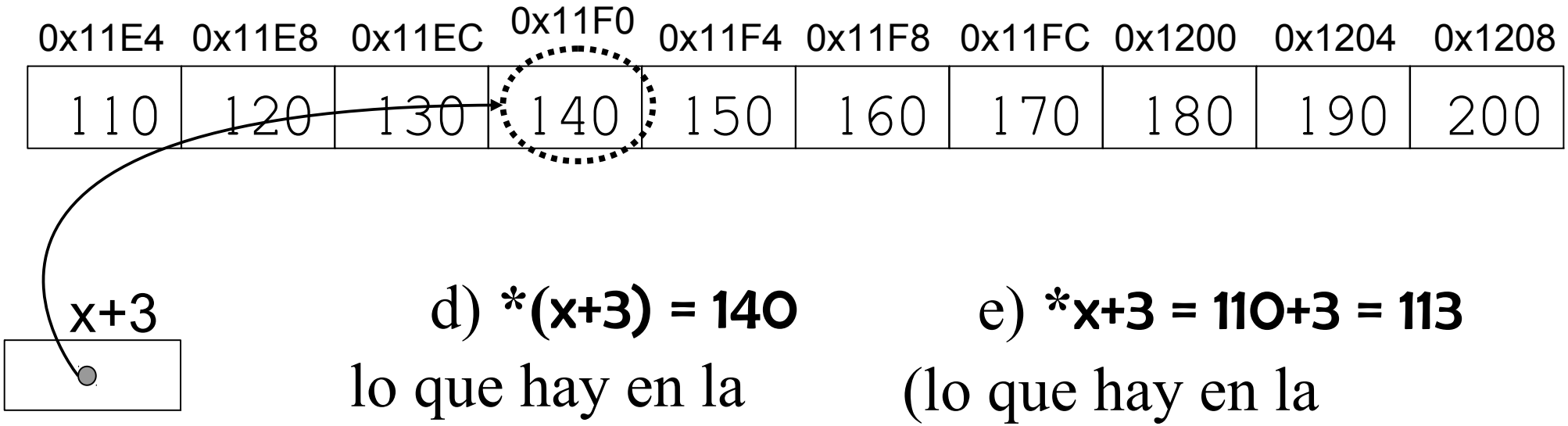
e) ***x+3 ?**

(lo que hay en la direccion x) + 3

Punteros y Arreglos

```
int x[10]={110, 120, 130, 140, 150,
160, 170, 180, 190, 200};
```

El elemento inicial **x[0]** se ubica en la dirección de memoria **000011E4**:



d) ***(x+3) = 140**

lo que hay en la direccion (x+3)

e) ***x+3 = 110+3 = 113**

(lo que hay en la direccion x) + 3

Punteros void

- Los punteros *void* pueden apuntar a cualquier tipo de dato.
- Su única limitación es que el dato apuntado no puede ser referenciado directamente (no se puede usar el operador de referencia * sobre ellos), dado que la longitud del dato apuntado es indeterminada.
- Por lo anterior, siempre se debe recurrir a la conversión de tipos (*type casting*) o a asignaciones para transformar el puntero *void* en un puntero de un tipo de datos concreto.

Ej:

```
void *memcpy (
    void *destino,
    const void *origen,
    int tamanio);
```

Estructuras dinámicas

Resuelven los siguientes problemas:

- Crear variables cuyo tamaño se desconoce en tiempo de compilación.
- Datos que no perduran durante toda la ejecución del programa.

Ej:

```
int a[10]; // vector estático
int *b=new int[10]; // vector dinámico
```

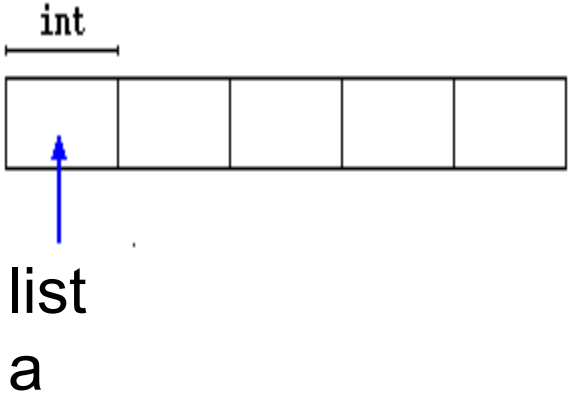
Operadores new y new[]

Para solicitar memoria dinámica **new**. Va seguido por un tipo de dato y opcionalmente el número de elementos entre corchetes `[]`. Retorna un puntero que apunta al comienzo del nuevo bloque de memoria asignado durante la ejecución del programa. Su forma es:

```
puntero = new tipo
o
puntero = new tipo [elementos]
```

Por ejemplo:

```
int *uno;
uno = new int;
int *lista;
lista= new int [5];
```



Operadores delete y delete[]

- Una vez que no se necista más hacer uso de una estructura dinámica es posible liberar los recursos que ella invlolucra y ponerlos disponibles para las aplicaciones que se están ejecutando.

```
delete puntero;
delete [] puntero;

Ej:
char *aux=new char[n];
...
delete []aux;
```

Funciones que devuelven arreglos

```
include <iostream>
using namespace std;
int *func(int a[]);

int main( ) {
    int *p;
    p=func(x);
    for (int i=0; i<10; i++)
        cout<<i<<"   "<<p[i]<<endl;
    delete []p;
    return 0;
}

int *func(int a[]) {
     int *q= new int[10];
     for (int i=10;i<4;i++)
         q[i]=rand();
     return q;
}
```

Funciones malloc y free

Existe dos funciones para reservar memoria heredadas de C:

```
int *p1 = new int[10];
int *p2 = (int*)malloc(10*sizeof(int));
... /*se operan exactamente igual*/ ...
delete []p1;
free(p2);
```

Diferencias:

malloc y free no reconocen el tipo de dato

new y delete *construyen* y *destruyen* los *objetos*

Funciones que devuelven arreglos

```
include <iostream>
using namespace std;
int *func(int a[]);

int main( ) {
    int *p;
    p=func(x);
    for (int i=0; i<10; i++)
        cout<<i<<"   "<<p[i]<<endl;
    delete []p;
    return 0;
}

int *func(int a[]) {
     int *q= new int[10];
     for (int i=10;i<4;i++)
         q[i]=rand();
     return q;
}
```

Punteros a estructuras

```
struct pelicula {
            char titulo [50];
            int anio; };
```

```
pelicula peli;
pelicula *ppeli = &peli;
```

Las distintas notaciones para acceder a miembros de un struct:

`peli.titulo:` elemento titulo de la struct **peli**

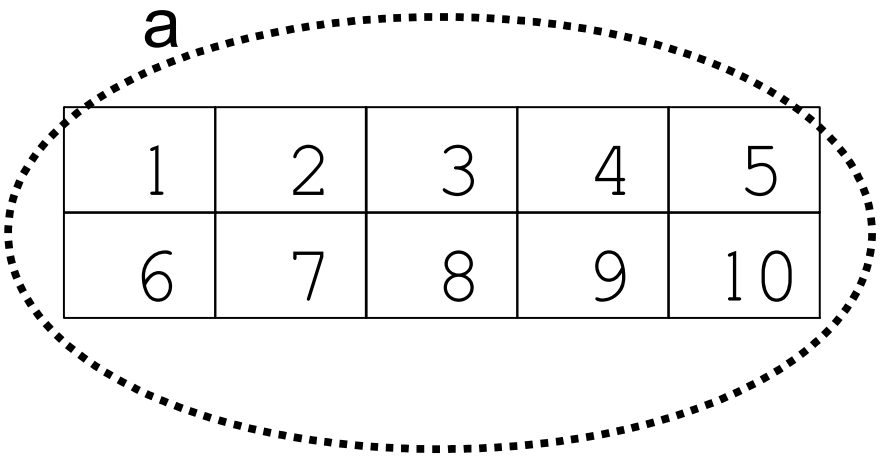
`ppeli->titulo:` elemento titulo del struct apuntado por **ppeli**

`(*ppeli).titulo`: idem anterior

`*peli.titulo:` valor apuntado por el puntero titulo del struct **peli**

Punteros y Matrices

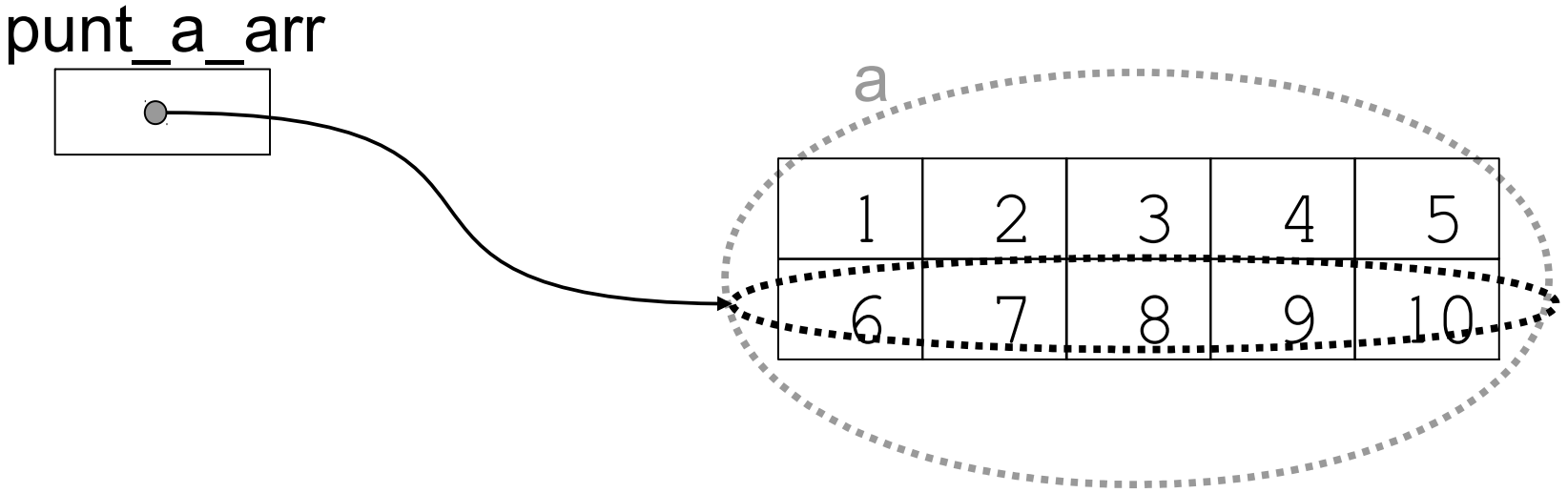
```
int a[2][5] = { {1,2,3,4,5} , {6,7,8,9,10} };
```



Punteros y Matrices

```
int a[2][5] = { {1,2,3,4,5} , {6,7,8,9,10} };

int (*punt_a_arr)[5] = &(a[1]);
```

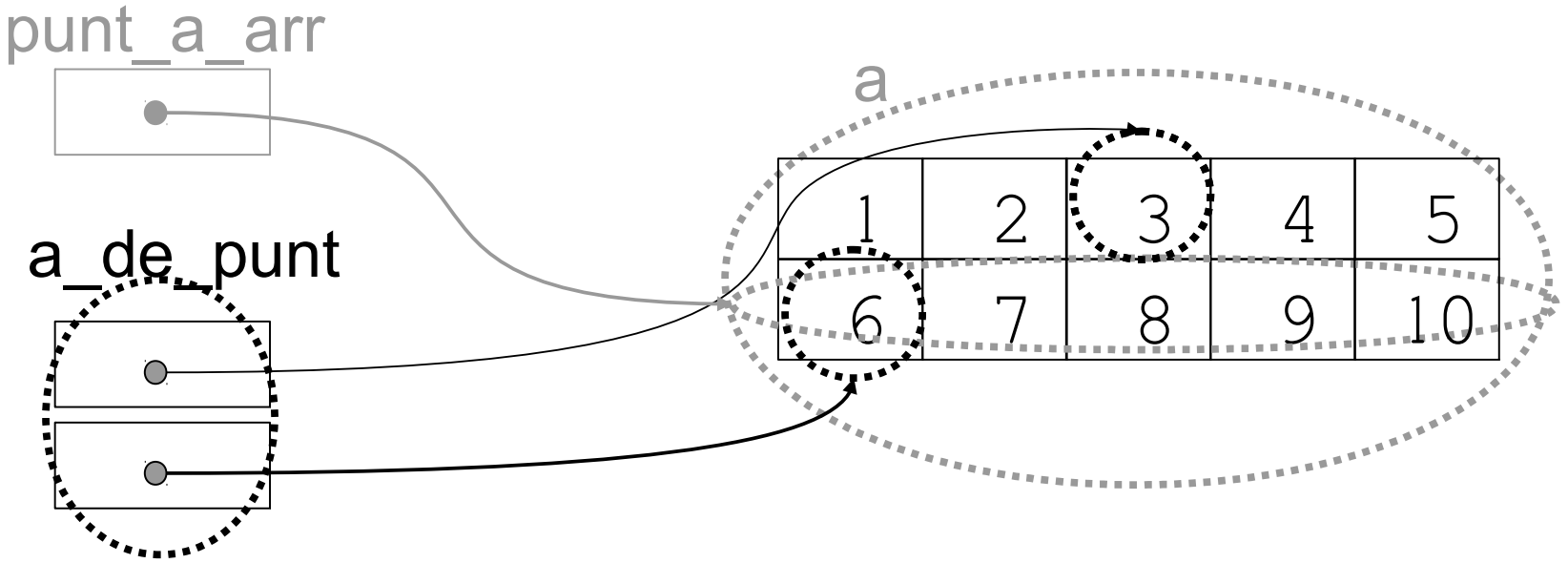


Punteros y Matrices

```
int a[2][5] = { {1,2,3,4,5} , {6,7,8,9,10} };

int (*punt_a_arr)[5] = &(a[1]);

int *a_de_punt[2] = { &(a[0][2]) , &(a[1][0]) };
```

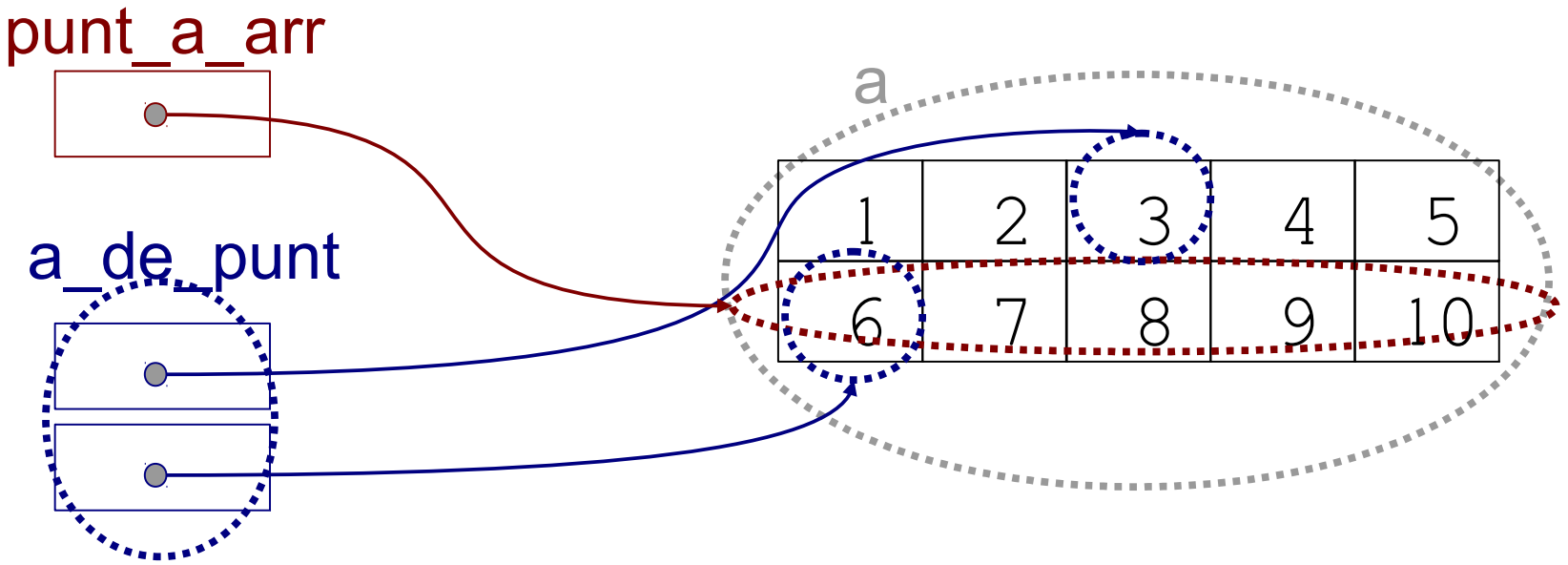


Punteros y Matrices

```
int a[2][5] = { {1,2,3,4,5} , {6,7,8,9,10} };

int (*punt_a_arr)[5] = &(a[1]);

int *a_de_punt[2]={&(a[0][2]),&(a[1][0])};
```

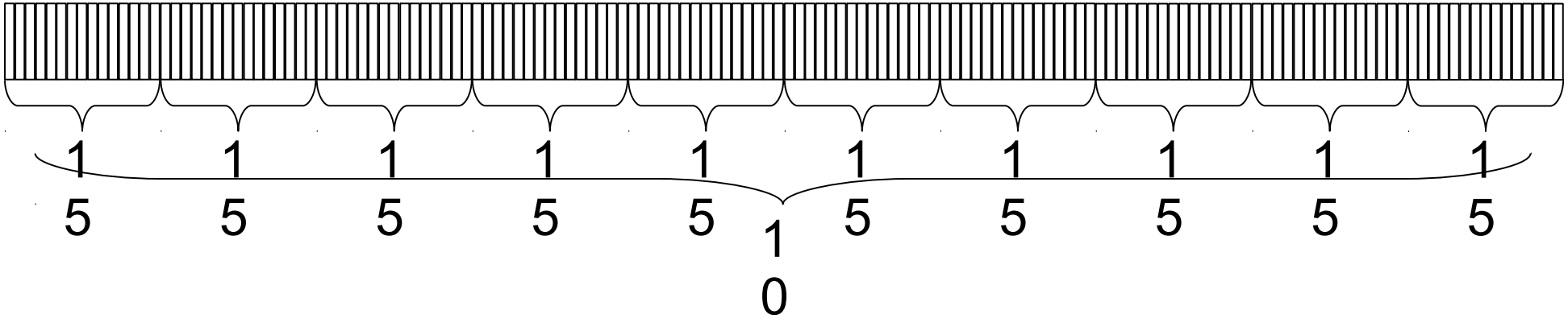


2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.

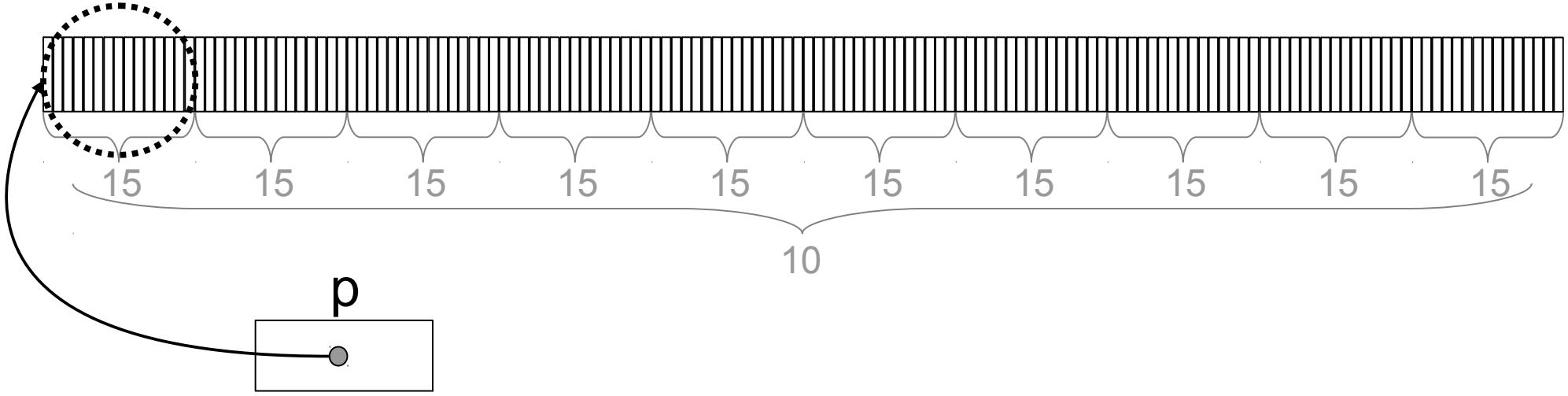
2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.



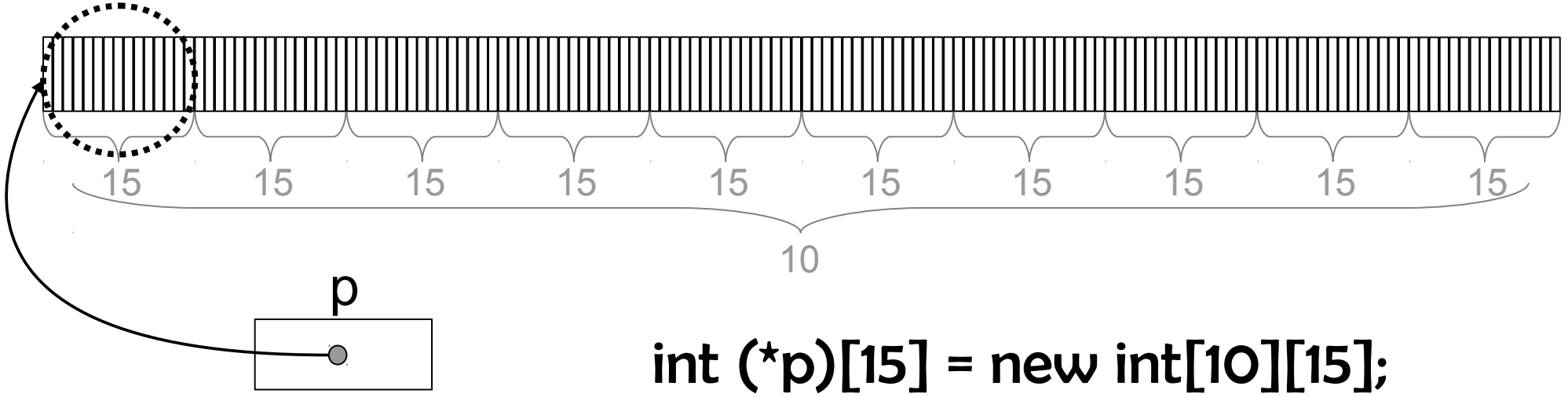
2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.



2. Usando la sintaxis de C++ escriba el código que crea necesario para:

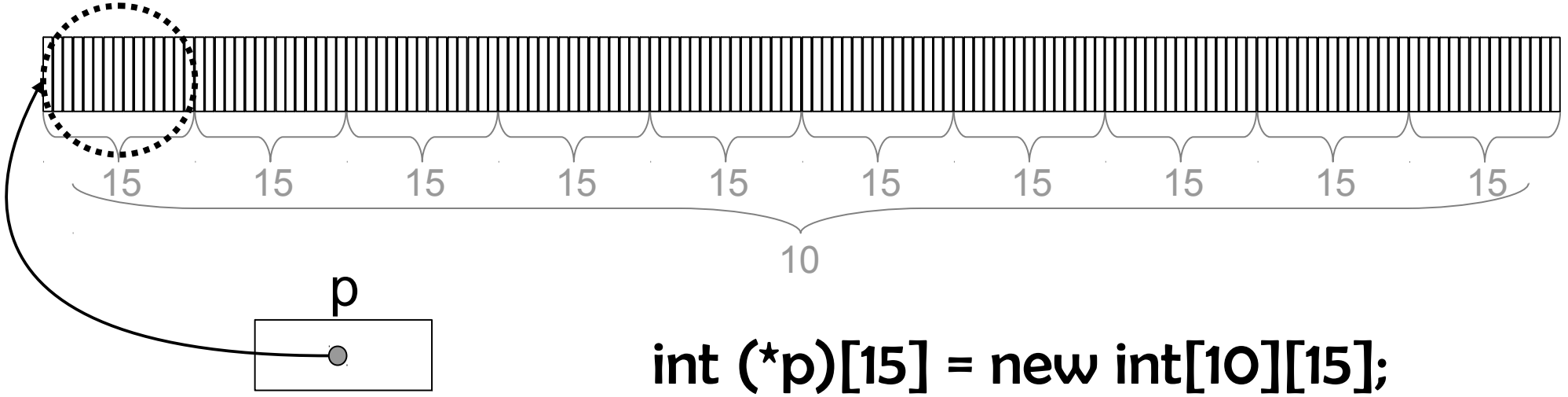
b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.



int (*p)[15] = new int[10][15];

2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.

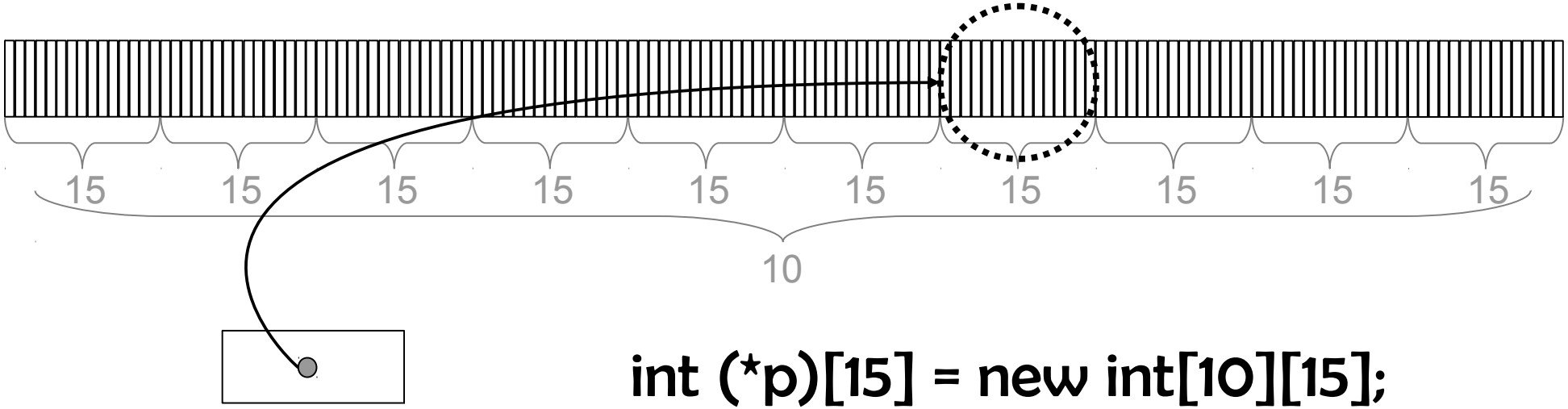



```
int (*p)[15] = new int[10][15];

p[6][2] = 5;
```

2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.



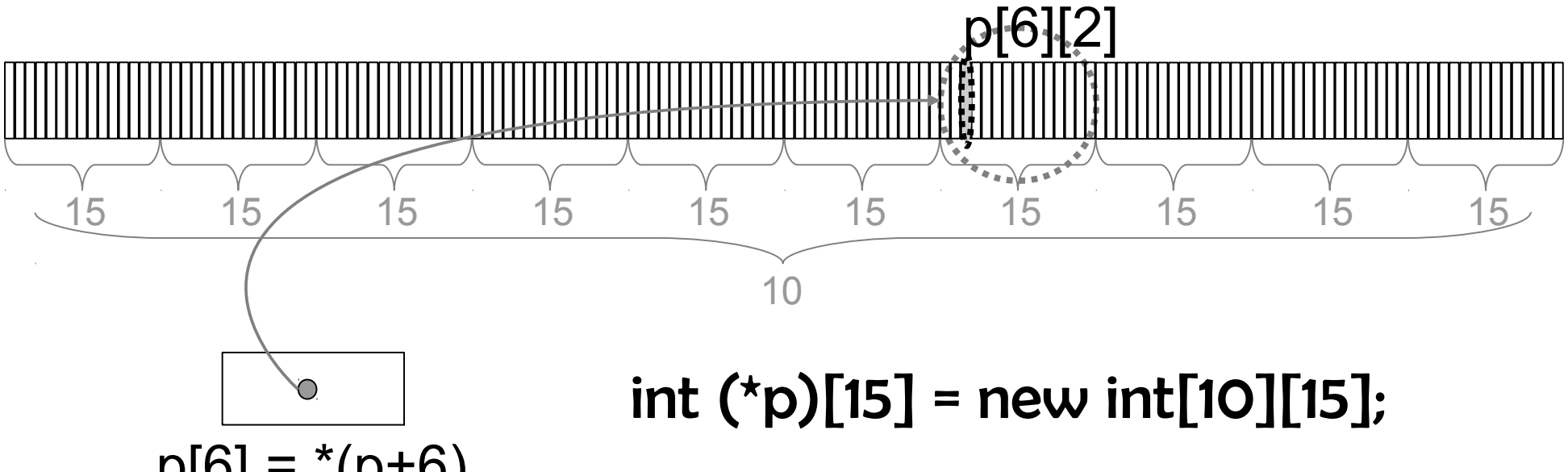
p[6] = *(p+6)

int (*p)[15] = new int[10][15];

p[6][2] = 5;

2. Usando la sintaxis de C++ escriba el código que crea necesario para:

b) Declarar un puntero a una lista de 10 arreglos contiguos de 15 elementos enteros cada uno. Reserve la memoria necesaria.



int (*p)[15] = new int[10][15];

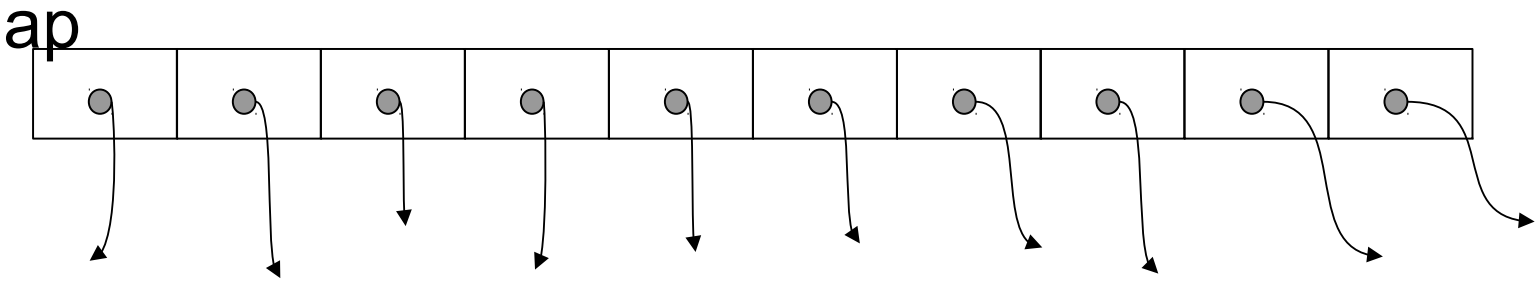
p[6][2] = 5;

2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.

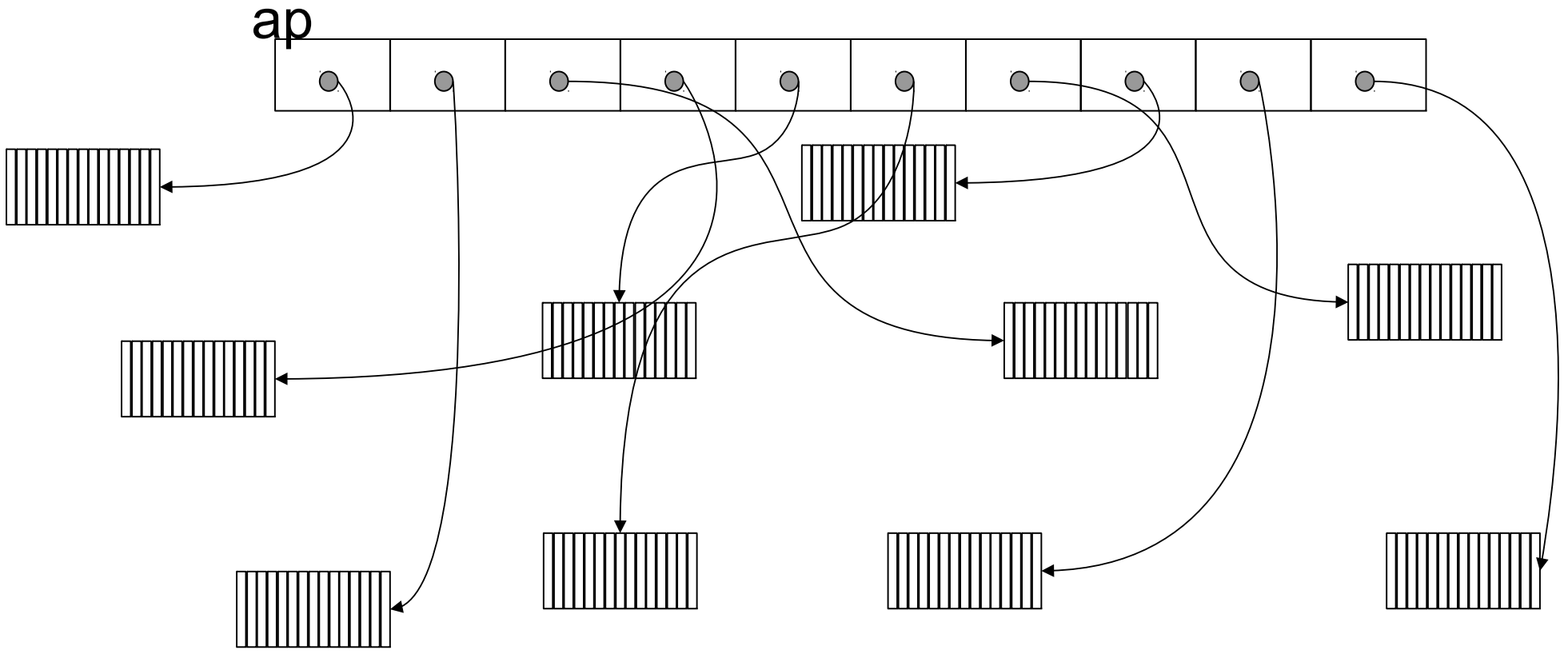
2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.



2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.



2. Usando la sintaxis de C++ escriba el código que crea necesario para:

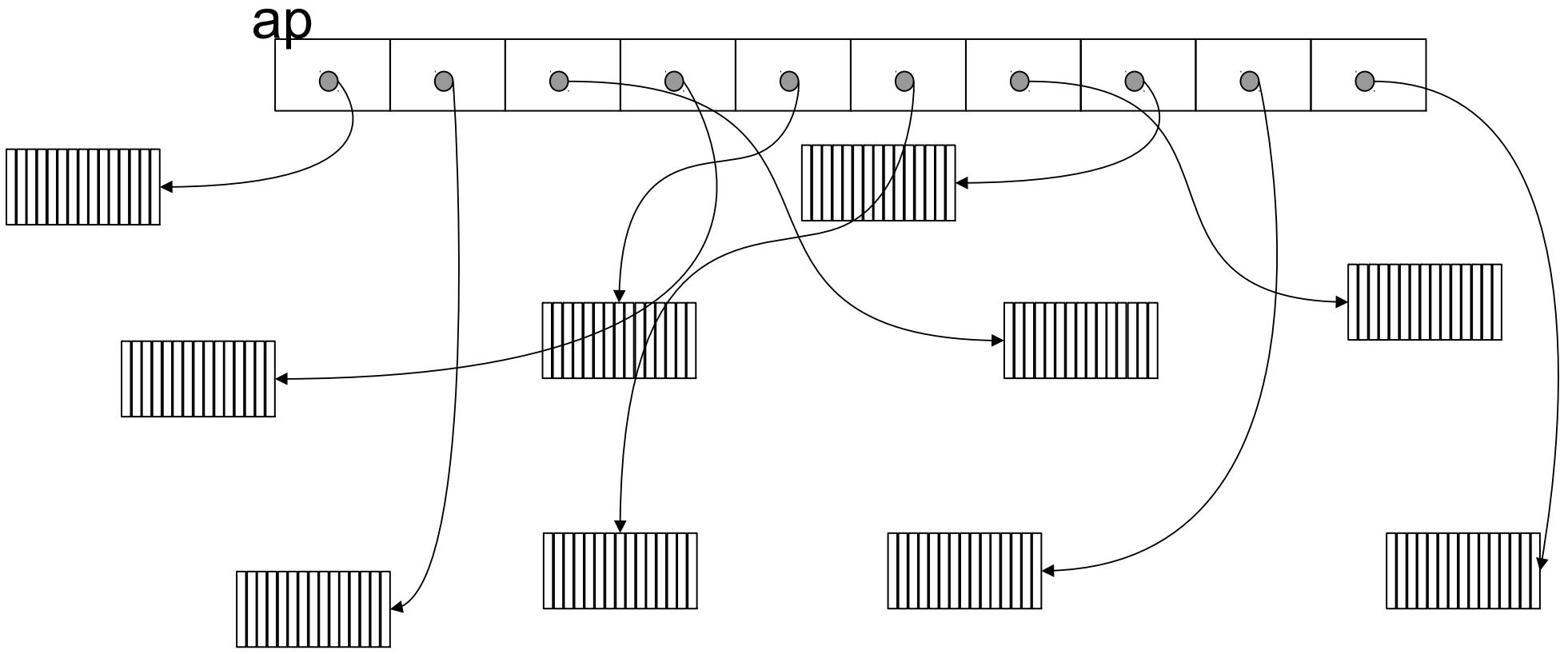
c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.

```
float *ap[10];

for (int i=0; i<10; i++)
    ap[i] = new float[15];
```

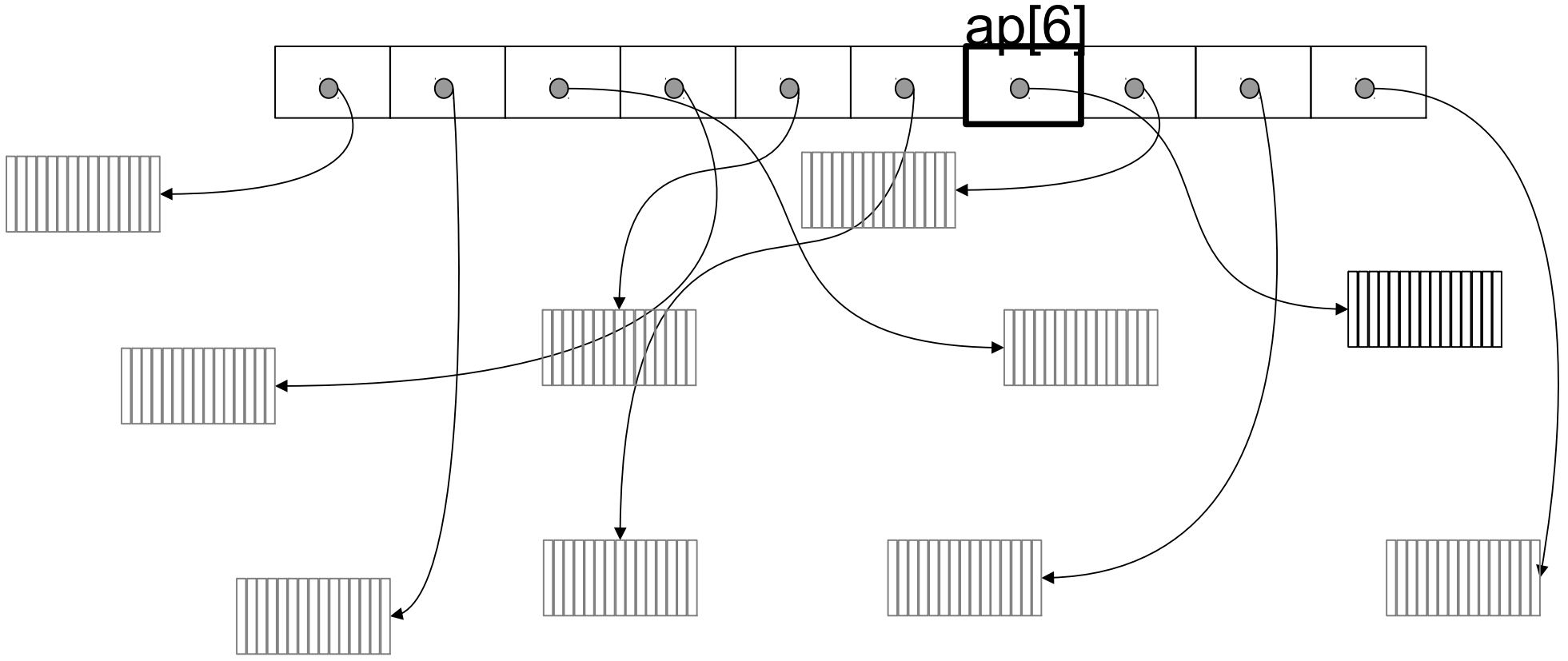
2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.



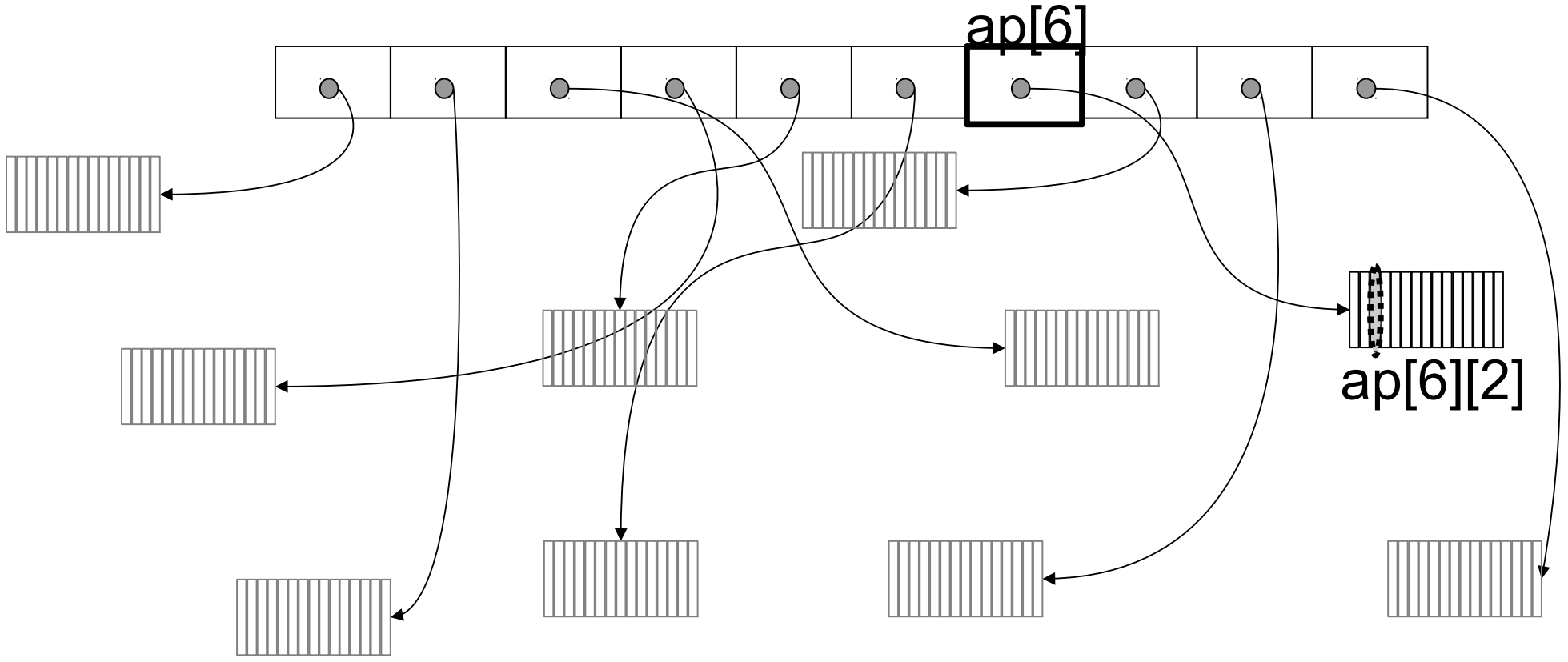
2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.



2. Usando la sintaxis de C++ escriba el código que crea necesario para:

c) Declarar un arreglo de punteros para representar una matriz de 10x15 elementos flotantes.



Punteros a funciones

- Es posible pasar una función como parámetro a otra función.

```
int suma (int a, int b)   { return (a+b); }
int resta (int a, int b)    { return (a-b); }

int operacion (int x, int y,
            int (*func_a_llamar)(int,int) ) {
    int g;
    g = (*func_a_llamar)(x,y);
    return (g);
}

int main () {
   int m,n;
   m = operacion (7, 5, &suma);
   n = operacion (20, m, &resta);
   cout <<n; return 0;
}
```

Punteros a funciones

```
void sort(pelicula *desde, pelicula hasta,
                                    bool (*menor)(pelicua,pelicula);

struct pelicula { char nombre[50]; int anio; };

bool compara_por_nombre(const pelicula p1, const pelicula p2) {
    return strcmp(p1.nombre,p2.nombre)<0;
}

bool compara_por_anio(const pelicula p1, const pelicula p2) {
    return p1.anio<p2.anio;
}

int main(int argc, char *argv[]) {
    pelicula A[50];
    ...
    sort(A,A+50,compara_por_nombre);
    ...
    sort(A,A+50,compara_por_anio);
    ...
    return 0;
}
```