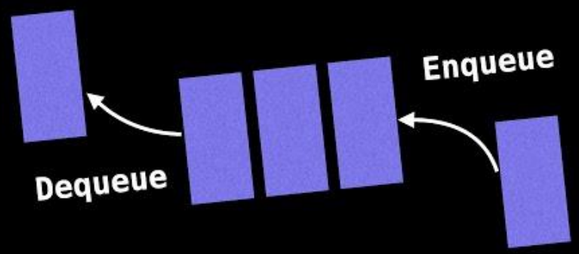
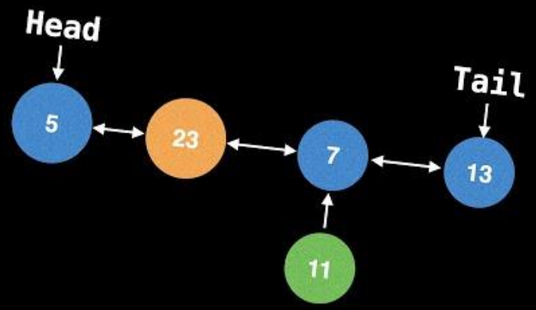
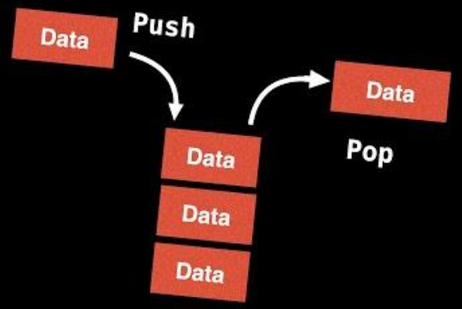
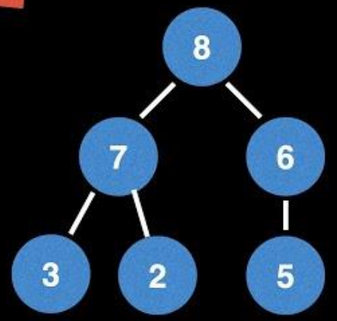
Welcome to

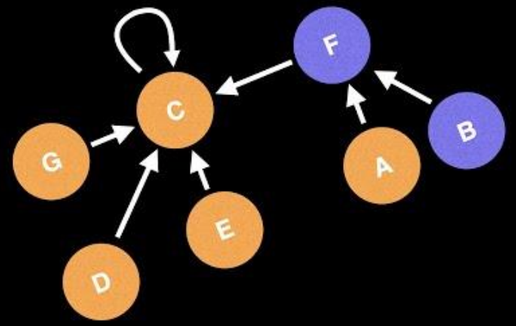




Data Structures







CS2100

Heider Sanchez E., Dr
hsanchez@utec.edu.pe

Qué veremos en este curso...

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

- Algoritmos para resolver problemas de manera eficiente
- Estructuras de datos para almacenar y organizar datos eficientemente
- Análisis de requerimientos para uso adecuado de distintas estructuras de datos

Qué NO veremos en este curso...

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

- No se enseñará C++, sin embargo este lenguaje se usará para las prácticas.
- POO se asume como algo estudiado en cursos previos.
- La STL y estructuras básicas se entienden como ya vistas en cursos previos.

Tutorial rápido de STL C++

Lista de temas

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

1. Sorts, listas e iteradores
2. Teoría de grafos
3. Matrices esparzas
4. Árboles equilibrados
5. Tries

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Laboratorio

1. Si el entregable no compila será calificado sobre 11
2. Tratar de evitar warnings en los proyectos
3. Tratar que pasen todos los test de prueba.
4. No olviden subir su código a master

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Sistema de evaluación

EVALUACIÓN	TEORÍA (T)	LABORATORIO (L)
* La ponderación de la evaluación se hará si ambas partes están aprobadas	Examen **E1** (25%) Examen **E2** (25%)	Evaluación Continua **C1** (20%) Evaluación Continua **C2** (20%) Proyecto **P1** (5%) Proyecto **P2** (5%)
	50%	50%
	100%	

Evaluación Continua:

- Tareas en Github Classroom (grupales/individuales)
- Practicas calificadas (individuales)

Comunicación

Discord:

Ingresar a:

https://discord.gg/tE2DZRTs

desde tu celular o laptop

Tu NickName debe ser nombre y apellidos

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

Herramientas

1. IDE: CLion, Visual Code

2. Git + Git Flow + Fork

3. Karma Git Commits
    a. *http://karma-runner.github.io/4.0/dev/git-commit-msg.html*

Tarea

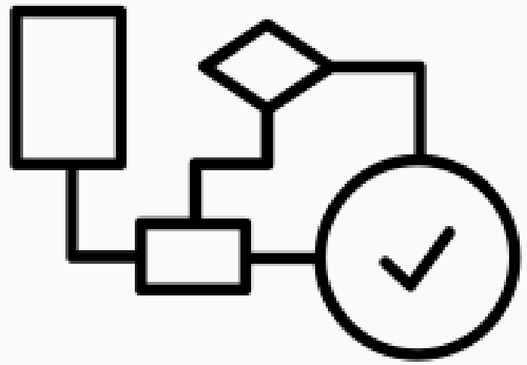
1. Crear una cuenta en github (https://github.com/)
2. Revisar: clone/add/commit/push/pull
3. Hacer push de un main.cpp (Hello world!)
4. Revisar branch/checkout/merge
5. Crear una rama develop, modificar el main.cpp
6. Hacer un commit y merge con master

Qué es un algoritmo?

➔ Es un conjunto ordenado de pasos o instrucciones, los cuales son realizados para resolver un problema.

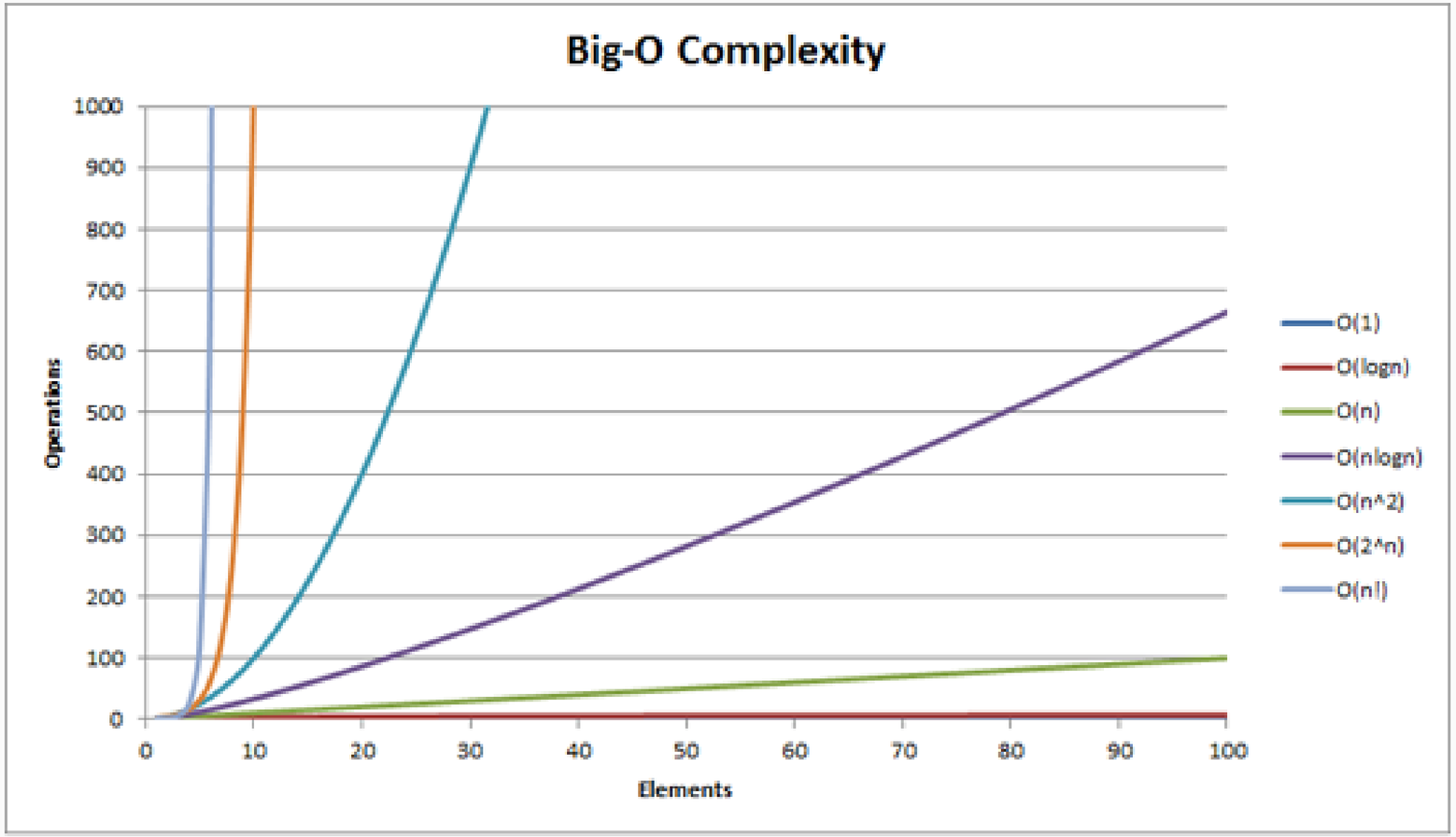
➔ Un algoritmo siempre debe producir un resultado, por ello se debe hacer de forma racional y con un objetivo en mente.

E.g. Una receta de cocina, la manera en la que se muestran los resultados de Google, etc.



Qué es la notación big O?

Provee un límite superior a la tasa de crecimiento de una función



Qué es la notación big O?

- T = a = **O(1)**
- T = an + b = **O(n)**
- T = $an^2 + bn + c$ = **$O(n^2)$**

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65,536
5	32	160	1024	32,768	4,294,967,296

En qué casos sería O(log n)?

Ordenamiento

Qué ejemplos de ordenamiento se les ocurren?
Ordenar a personas en base a la edad, ordenar entradas de un blog en base a fecha, etc.

Qué algoritmos de ordenamiento conocen?
Bubble sort, selection sort, insert sort, quick sort, merge sort, heap sort, counting sort, etc.

En qué nos basamos para elegir un algoritmo de ordenamiento para un proyecto?
Algunos son más rápidos pero usan muchos recursos, otros lentos pero ligeros y usan pocos recursos.
Esto los hace un gran caso de estudio para comparar.

Vamos a recordar un poco… Punteros

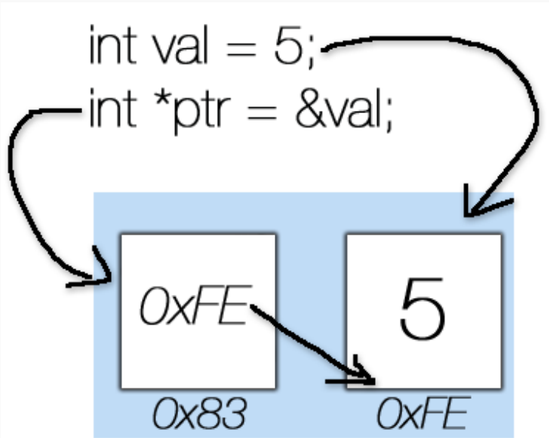
Qué es un puntero?
Un puntero es una variable la cual almacena la dirección de otra variable. Para declarar un puntero se utiliza la siguiente nomenclatura: *<tipo> *<nombre>;*

Dada la tabla de la derecha, cuánto espacio ocupa un puntero en memoria?
Un puntero ocupa 4 bytes en arquitecturas de 32 bits y 8 bytes en arquitecturas de 64 bits. No depende del tipo de dato.

Type Name	32-bit Size	64-bit Size
`char`	1 byte	1 byte
`short`	2 bytes	2 bytes
`int`	4 bytes	4 bytes
`long`	4 bytes	8 bytes
`long long`	8 bytes	8 bytes

Punteros

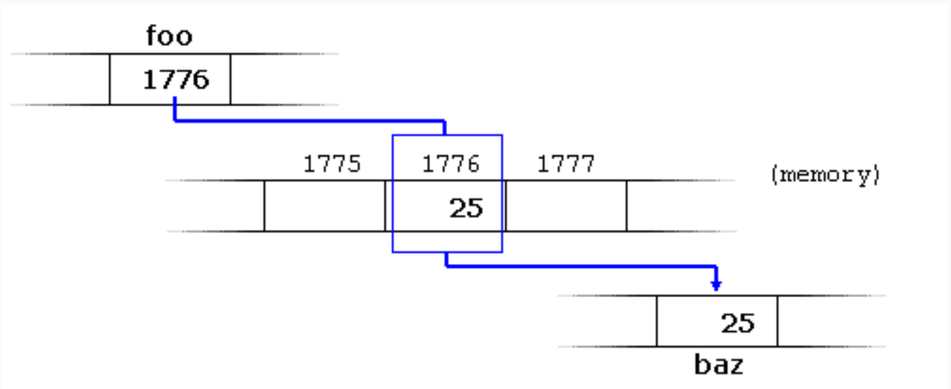
Una variable que almacena la dirección de otra variable es un puntero. Por tanto se suele decir que un puntero **“apunta a”** la variable la cual su dirección está almacenada



Operadores importantes:

Dereference (*):

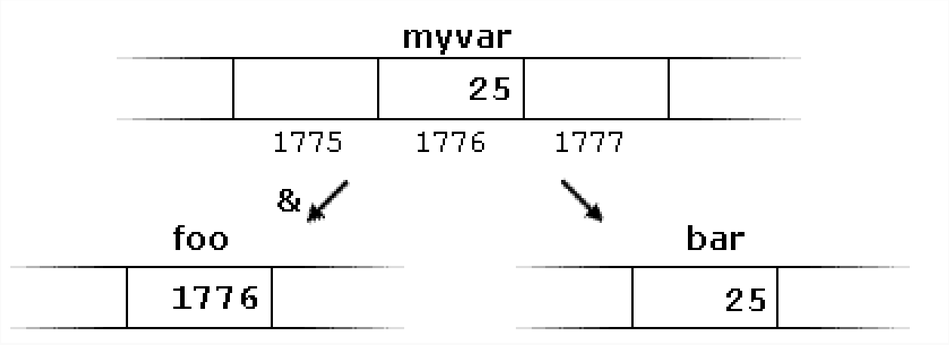
a. Se utiliza para definir al puntero, y va de lado de la variable. E.g. int *a, *b, *c;
b. A su vez, permite acceder al contenido a la cual apunta. E.g. int d = *a;



Punteros

Address-of (&):

a. Obtiene la dirección de una variable de cualquier tipo. E.g. int *pointer = &variable;



Por qué el tipo de dato debe ser conocido al momento de definir un puntero?
Se debe a una de las características de *Dereference*, ya que podemos obtener el valor al cual se apunta.

Recuerden que la verdadera dirección solo se puede conocer en tiempo de ejecución

Punteros

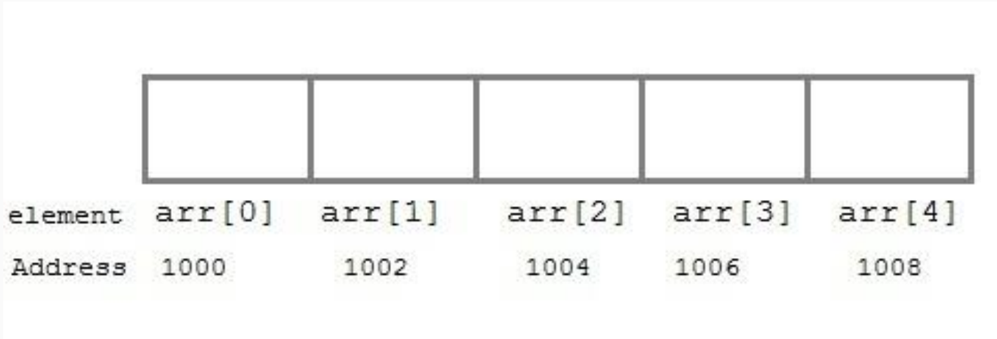
Cuando un array es declarado, se separa suficiente memoria para almacenar todos los elementos, dado:

```
int arr[5] = {1, 2, 4, 8, 16};
cout << arr << endl;
```

Qué imprimirá el código anterior?
1000, ya que *arr = &arr[0]*

y esto?

```
cout << *arr + 2 << endl;
// Sería 3!
```



Punteros

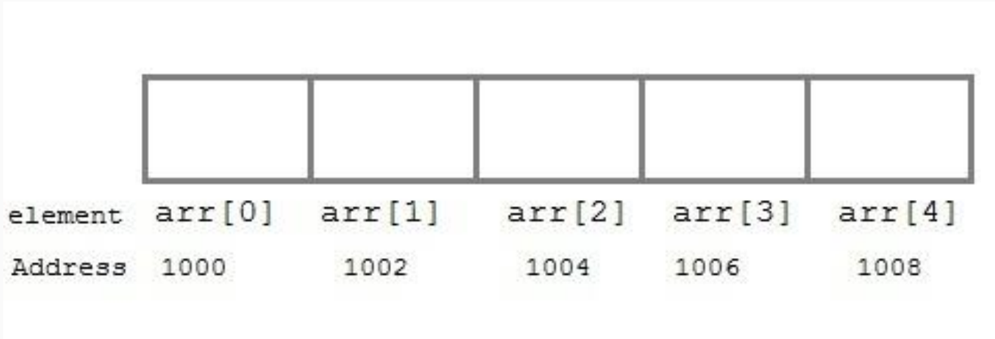
int arr[5] = {1, 2, 4, 8, 16};

Bien, entonces continuando que daría esto?

cout << *(arr + 2) << endl;
// Sería 4, perfecto

Finalmente, qué tal esto?

cout << *(arr + 12) << endl;
// 21908, daría basura probablemente



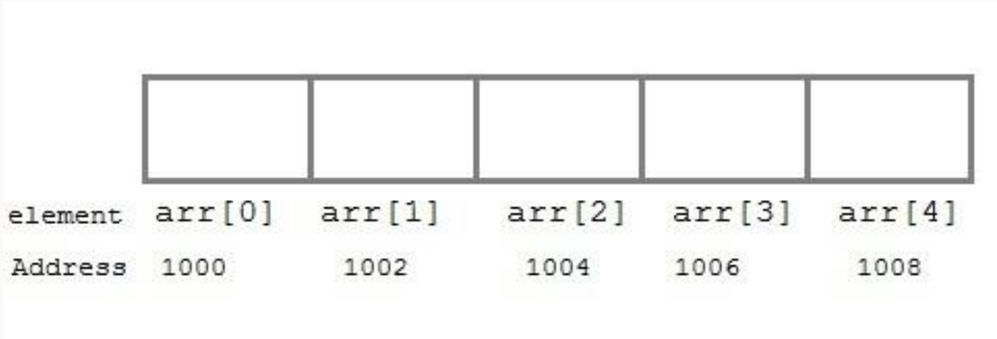
Punteros

Recuerden que un puntero y un arreglo son equivalentes en ciertas situaciones y tienen propiedades similares

int arr[5], *ptr;
ptr = arr; // Válido

Sin embargo, un puntero puede ser asignado a diferentes direcciones mientras que un arreglo siempre va a representar el mismo bloque.

arr = ptr; // Esto nunca será válido

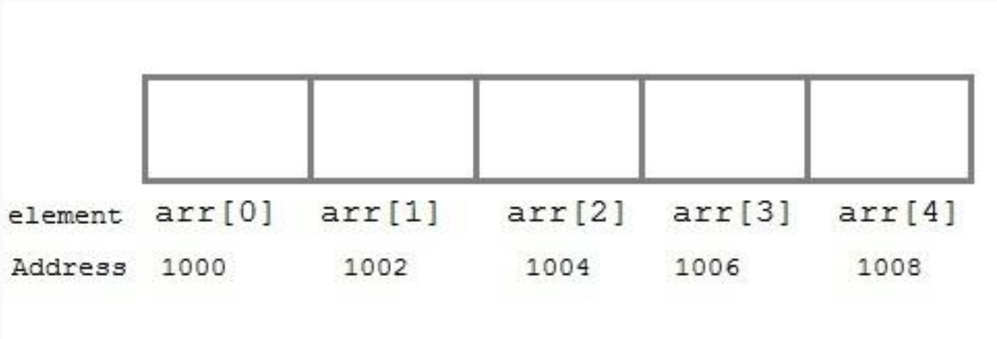


Punteros

En el caso de un array de varias dimensiones, como arr[i][j], podríamos tratarlo de la siguiente manera: *(*(a + i) + j)

Ahora, que imprimiría el siguiente código?

```
int number = 17, *ptr;
ptr = &number;
cout << *ptr << endl;
```



perfecto, 17. Y esto?

```
int number = 13, *ptr;
*ptr = number;
cout << *ptr << endl;
```

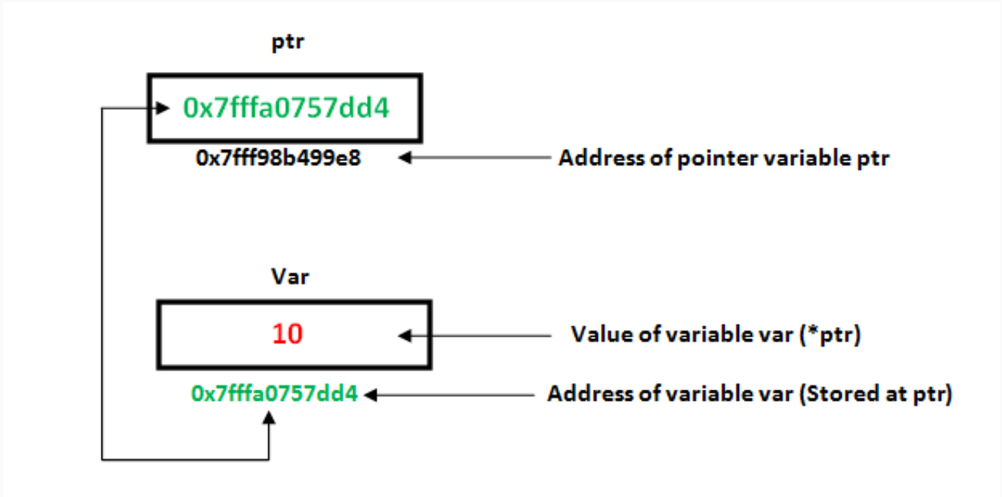
Error de memoria, porque el puntero no ha sido inicializado

Punteros

Cómo se pueden inicializar los punteros?

- int *ptr = new int;
  *ptr = 5;

- int variable = 5;
  int *ptr = &variable;



El siguiente código estaría correcto?

int *ptr = &5;

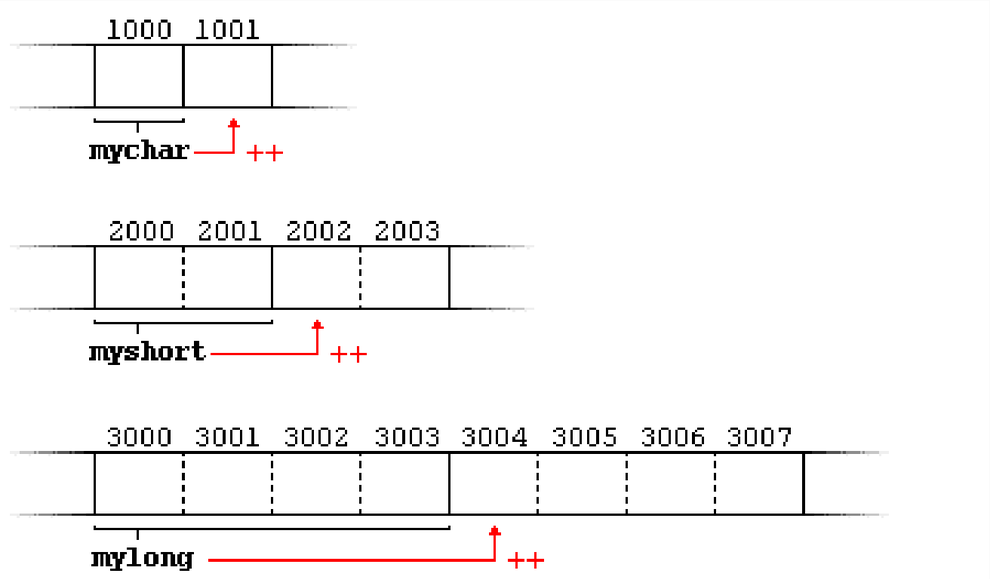
// No, ya que 5 no es una variable

Punteros

Ya vimos un par de operaciones aritméticas sobre arreglos, me podrían decir, qué sucede aquí?

```
char *mychar;
short *myshort;
long *mylong;

++mychar;
++myshort;
++mylong;
```



Punteros

Dadas las siguientes equivalencias:

- *p++ = *(p++)
- *++p = *(++p)
- ++*p = ++(*p)

Siempre recuerden:

i++ = incrementa pero retorna original
++i = incrementa y retorna nuevo valor

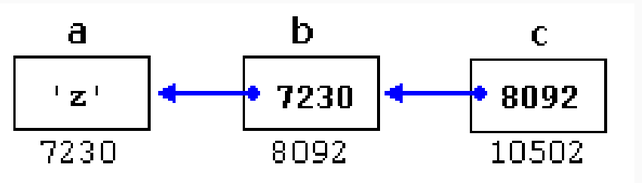
Qué imprimirá?

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;

cout << *ptr << endl;          1
cout << *ptr++ << endl;        1
cout << *++ptr << endl;        3
cout << ++*ptr << endl;        4
cout << (*ptr)++ << endl;      4
```

Punteros

Los punteros también pueden apuntar a punteros, sólo se tiene que agregar un * en la declaración: int ** ptr;



Punteros a void, son un tipo especial de punteros donde *void* representa la ausencia de tipo. Por tanto, puede ser usado para cualquier tipo con algunas restricciones en cuanto a sus operadores

```
int number = 5;
void *ptr = &number;
cout << *(int*)ptr << endl;
```

Recuerda siempre definir tus punteros a nullptr o NULL, cuando estén vacíos, de otra forma va a apuntar a basura

```
int *ptr = nullptr;
int * ptr = NULL;
```

Finalmente, un puntero también puede apuntar a funciones! Sería bueno que lo revisen

NULL is a "manifest constant" (a `#define` of C) that's actually an integer that can be assigned to a pointer because of an implicit conversion.

nullptr is a keyword representing a value of self-defined type, that can convert into a pointer, but not into integers.

```
1  int i = NULL; // OK
2  int i = nullptr; // error - not a integer convertible value
3  int* p = NULL; //ok - int converted into pointer
4  int* p = nullptr; // ok
```