ARRAYS

CADA APARTADO PRINCIPAL CORRESPONDE CON EL NOMBRE DEL PROYECTO CON EJEMPLOS DE LO APUNTADO AQUÍ

1. INTRODUCCIÓN A LOS ARRAYS

- Los `arrays` son una forma de establecer colecciones en nuestros programas de C++.
- Hasta ahora estábamos trabajando con variables sueltas. Sin embargo en ocasiones nos puede interesar agrupar todas estas variables y manejarlas como una única entidad. Los `arrays` son los encargados de ellos.

2. DECLARACIÓN Y USO DE ARRAYS

- Los `arrays` permiten agrupar varias variables y manejarlas a la vez como si fuera una sola.
- Puedes referenciar cada una de las variables pertenecientes al `array` con `[]` y el índice, que es el número que se corresponde con la posición en la que se encuentra la variable en el `array`. La primera posición del `array` es siempre 0. Por lo que un `array` de tamaño 10 (el número de variables que contiene) tendrá las posiciones de las variables que irán desde 0 hasta 9.
- Ejemplo de `array`:
  - `int puntuaciones[10]; //Un array de tipo int de tamaño 10`
- Para declarar un `array` se debe especificar el tipo de dato, y el tamaño del `array`.
  En el ejemplo de arriba, tenemos un `array` de tipo `int` de tamaño 10.
- Cuando se declara un array de primeras no tiene ningún dato que nosotros le hayamos indicado, solo tiene información “basura” hasta que le pasemos nuestras propias variables.

- Los `arrays` tienen límites (*boundaries*), cuando especificas el tamaño de un `array`, este solo va a tener ese número de elementos, sin embargo si intentas leer o referenciar a un elemento fuera del `array` (por ejemplo la posición 12 de un `array` de tamaño 10) en C++ podrás hacerlo, pero el valor obtenido será información "basura" que tu no has especificado. Ya que estás intentando acceder a datos que no están alojados en la memoria. Y esto lógicamente puede dar lugar a errores en el programa o *crasheos*.
- Al igual que podemos acceder a una posición concreta del `array` con `[]` y el índice, también podemos recorrer un `array` mediante bucles, para leer todos los elementos del `array` y manejarlos u operar con todos a la vez. Lo normal es usar un `for` o un `for in range` para iterar sobre los `arrays`, más fácilmente.
- Al igual que podemos leer los elementos del `array`, también podemos escribir en ellos, es decir podemos asignar valores a cada variable del `array`. Y esto podemos hacerlo manualmente de uno en uno o con un bucle para escribir en todos los elementos del `array` a la vez mediante el uso del iterador del bucle como índice del `array`.
- Un `array` puede ser declarado e inicializado a la vez.
  En el caso de que declares un `array` de por ejemplo 5 de tamaño y solo inicialices los 3 primeros valores, los otros 2 se inicializarán automáticamente a 0.
  Puedes omitir el tamaño del `array` si lo inicializas a la vez, ya que el compilador deducirá el tamaño en base al número de valores que inicialices. Es decir un `array` con 6 valores inicializados hará que el compilador piense que el tamaño del `array` es de 6 como es lógico.
  Si haces un `array` constate (`const`) , no podrás hacer operaciones de escritura en él.
- Si los índices no te hacen falta y solo te importan los valores puedes usar un `for in range` para facilitar el código y la tarea.
- Los `arrays` almacenan variables del mismo tipo de dato. **NO PUEDES** rellenar un `array` con variables que tengan distintos tipos de dato, **TODAS** las variables deben de tener el mismo.
- También puedes realizar operaciones con los elementos de un `array` al recorrerlo.

3. TAMAÑO DE UN ARRAY

- Para saber el tamaño de un array en tiempo de ejecución existe la función `std::size(nombreArray)`
- Sin embargo está función fue implementada en C++17, hasta entonces para saber el tamaño de un `array` había que hacer un par de cosas:
  - Sabiendo que los elementos de un `array` tienen que tener siempre el mismo tipo de dato y que este tipo de dato tenía que ser el mismo que el del propio `array` al declararlo, lo que se hacía era dividir la suma del tamaño en *bytes* de todos los elementos del `array`, entre el tamaño (también en *bytes*) de cualquiera de los elementos del `array`.
    Para hacer eso se usa `sizeof()`
    - ✓ 1 - Haces `sizeof(nombreArray)` para saber el tamaño en bytes de la suma de todos los elementos del `array`
    - ✓ 2 - Haces `sizeof(nombreArray[0])` para saber el tamaño en bytes de cualquier elemento del `array`, en este caso el primero de ellos
    - ✓ 3 - Divides ambos resultados y obtienes el número de elementos del `array`
- Si usas un `for in range` no te hace falta saber el tamaño del `array`, ya que la forma de iterar sobre el `array` es diferente y va a funcionar perfectamente, pero perderás la información de los índices de los elementos.
- En definitiva tienes 2 opciones:
  - Usar un for normal con `std::size()` para saber el tamaño del `array`.
  - Usar un `for in range` y perder la información de los índices.

4. ARRAYS DE CARACTERES

- Puedes declarar un `array` de caracteres como cualquier otro `array` indicando que el `array` es de tipo `char`, para inicializar el `array`, los elementos deben de estar separados por comas, y cada carácter debe ir entrecomillado con comilla simple.

- Al igual que con los `arrays` de `integers`, puedes recorrerlos mediante bucles y modificarlos los elementos del `array`.
- Una característica de los `arrays` de caracteres que los diferencia de los `arrays` de `int` es que puedes imprimirlos directamente por pantalla sin necesidad de recorrerlos con un bucle, sin embargo esto puede conllevar a resultados no deseados, ya que `std::cout()` no sabe cuándo parar de imprimir, puesto que las cadenas de C deben acabar en `NULL`, por lo que para imprimir únicamente el `array` de caracteres formado por los elementos del `array` sin basura extra, hay que añadir un elemento extra al final del `array` mediante el carácter de terminación `NULL \0`.
- Para saber porque los `arrays` de `int` no se pueden mostrar por pantalla sin recorrerlos con un bucle, ver el siguiente apartado.
- Como usar `arrays` de caracteres no es muy productivo, y es bastante tedioso, C++ nos permite definir `strings` de C literales creando un `array` de un único elemento, con la palabra o la oración que deseemos entrecomillada con comilla doble. Sin necesidad de añadir el carácter de terminación, cuando C++ detecta las comillas dobles lo añade automáticamente.

5. LÍMITES DE UN ARRAY

- Como hemos visto previamente, cuando asignas espacio en memoria para un `array`, el PC te otorga varias localizaciones de memoria para almacenar la información que desees, estas "localizaciones" serán del tamaño que hayas especificado al crear el `array` (el compilador también puede deducir el tamaño del `array` sin necesidad de especificarlo como hemos visto).
- Por ejemplo, supongamos que tenemos 10 localizaciones de memoria (de la 0 a la 9) y nuestro `array` tiene 4 elementos y se encuentra en las posiciones 2 a 5. Podremos modificar datos del `array` en las posiciones 2, 3, 4 y 5, sin embargo si intentamos modificar un elemento que se encuentre en una posición de memoria anterior o posterior a nuestro `array` (0, 1, 6, 7, 8 y 9), puede que esta se trate de una posición de memoria que pertenezca a otro programa, que esté siendo usada por el sistema

operativo, etc. Por lo que puedes liarla muy fácilmente si intentas trabajar con datos que estén fuera de los límites del `array`.

Fuera del array	Fuera del array	En el array	En el array	En el array	En el array	Fuera del array	Fuera del array	Fuera del array	Fuera del array

- C++ permite la lectura de elementos fuera de los límites del `array`, sin embargo a pesar de que el programa se compile, puede que leas basura o que estés accediendo a una posición de memoria restringida haciendo que el programa finalice automáticamente y *crashee*.

  C++ también permite la escritura fuera de los límites del `array`, sin embargo a pesar de que el programa se compile, en ningún momento eres el propietario de la posición de memoria en la que estás escribiendo, por lo que otros programas cambiarán la posición de memoria y si intentas leer esta nueva posición de memoria tampoco vas a conseguir lo esperado. También puede pasar que modifiques datos pertenecientes a otros programas, ya puedes imaginar lo que puede llegar a pasar.
- En resumen, es importante no salirse de los límites del `array`, para estar en una “a salvo” todo el tiempo.