|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **NOMBRE DE LA PRÁCTICA** | **Prácticas de C ++ Apuntadores** | | | **No.** | | **14** | |
| **ASIGNATURA:** | **Métodos númericos** | **CARRERA:** | **INGENIERIA EN SISTEMAS COMPUTACIONALES** | | **PLAN:** | | **ISIC-2010-224** |

**I. COMPETENCIA(S) ESPECÍFICA(S): Aprender c++**

**II. MATERIAL EMPLEADO:**

* Laptop con Linix Lite
* Editor de texto Visual Studio Code
* Terminal

**III.**

**DESARROLLO DE LA PRÁCTICA:**

**Lámina 14 Apuntadores**

* Importar la biblioteca del estándar de entrada y salida



* Declararemos el método principal de tipo entero



* Declarar dos variables de tipo entero una con nombre x con el valor de 17 y otra y sin valor inicial



* Declarar un apuntador llamado p de tipo entero



* El apuntador p guardaremos la dirección en memoria de x



* Imprimir el valor de x y como argumento el contenido de donde apunta p



* La variable y va a ser igual a el contenido de la dirección a donde apunta + 3 osea 20



* Por último para comprobar imprimiremos el valor de y que es 20 calculado con anterioridad



* Guardamos el código con el nombre de apuntadores1.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejemplo 2**

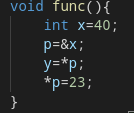
* Importar la biblioteca del estándar de entrada y salida



* Declarar dos variables globales de tipo entero que una será entera “y” y otra apuntador “p”



* Una función sin tipo de dato, tendrá una variable x con valor 40 que será apuntada su dirección por p y “y” apuntara a p, la variable p tenia el valor de 40 pero se remplaza con 23



* Declarar el método principal



* Llamar a la función creada anteriormente



* Recordemos que p tenia el contenido de 23 almacenado, y ahora y toma ese valor



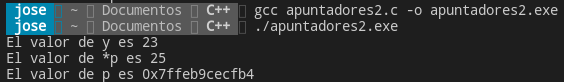
* P tenia el contenido de 23 y ahora cambiara su contenido a 25



* Imprimir el valor de y, p y la dirección de p a donde esta apuntando con valor hexadecimal



* Guardamos el código con el nombre de apuntadores2.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo

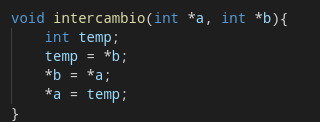


**Ejercicio 3**

* Importar la biblioteca del estándar de entrada y salida



* Declarar la función que recibirá por parámetros dos apuntadores de tipo entero, en la función que servirá como un respaldo llamado temp, temp tomara el valor del contenido de b, y b tomara el de a, por último a tomara el contenido de temp que era el de b



* Declarar el método principal



* Declarar dos variables de tipo entero x con valor de 2 y otra y que toma el de 5



* Se imprime el valor inicial de estas variables



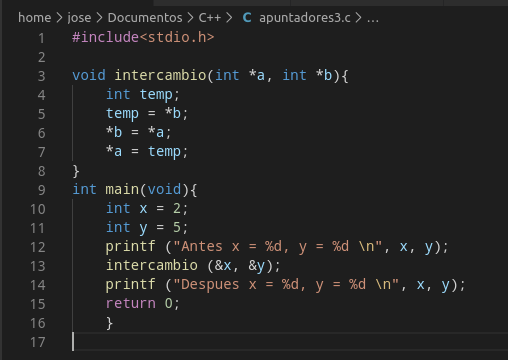
* Se llamara al la función intercambio que se le pasara como argumentos la dirección de las variables antes creadas, y se hará el intercambio de contenidos



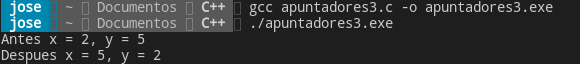
* Por último se hará la impresión de las variables pero con contenidos intercambiados



* Código completo



* Guardamos el código con el nombre de apuntadores3.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejemplo 4 Función size of**

* Importar la biblioteca del estándar de entrada y salida



* Declarar el método principal



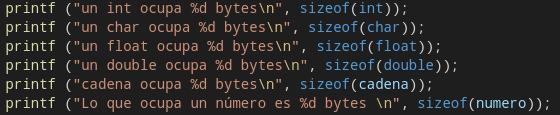
* Declarar dos arreglos con las mismas posiciones pero con distintos números de datos



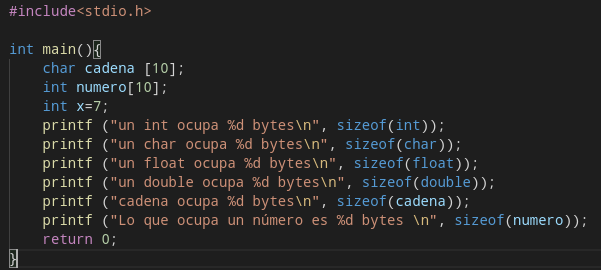
* Declarar una variables de tipo entera con un valor de 7 aunque el valor no es importante ahorita porque solo importa el tipo de dato



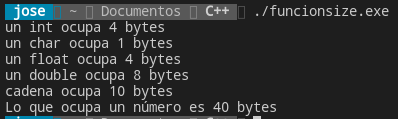
* Imprimir el valos de cada tipo, la función sizeof te muestra como entero cuantos bytes ocupa cada tipo de dato, como mi computadora es de x64 se descubrirá que valor adquiere, el char toma 1 byte asi que un arreglo de 10 posiciones tomara 10 bytes, y el arreglo de ints tomara 40 bytes ya que un int ocupa 4 de los mismos



* Código completo:



* Guardamos el código con el nombre de funcionsize.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejemplo 5 Función malloc**

* Antes de empezar yo veo esta función como el recepcionista de un restaurante que el usuario le dirá cuantas personas quiere (cuantos espacios en memoria) y el recepcionista dice si si tienes ese espacio en el restaurante (espacio en memoria) y si los hay los dejara pasar (asignara la memoria correspondiente)
* Primero importaremos las bibliotecas necesaria que es el estándar de entrada y salida y el estándar de bibliotecas



* Declarar el método principal



* Después declarar dos variables una que se usara para moverse entre posiciones y otra que almacenara el número de posiciones que ingresara el usuario



* Un apuntador de tipo carácter



* Pedir al usuario el número de la longitud de cadena



* Aquí el apuntador tomara con la función malloc los espacios necesarios en memoria para almacenar el número ingresado por el usuario más 1 (esto es una buena práctica), que como son caracteres tomaran un byte



* Si la memoria no tiene espacio necesario para almacenar lo pedido por la función malloc el programa habrá terminado



* Iniciar un ciclo for que que se repetirá desde 0 a el número ingresado por el usuario



* En las posiciones del buffer se guardara una letra del abecedario de forma “semi-aleatoria”



* Se llama a la posición inicial del buffer



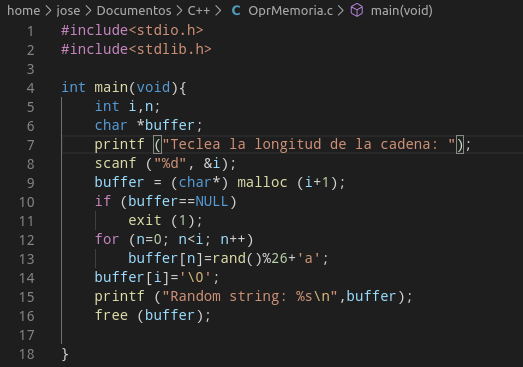
* Luego se imprimirá lo que tiene en esa posición de memoria del buffer



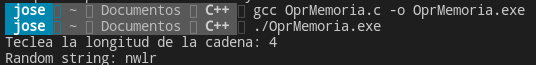
* Por último se liberara la memoria ocupada del buffer



* Código completo



* Guardamos el código con el nombre de OprMemoria.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejercicio 1**

1. Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.
2. Llena todos los elementos del arreglo con datos ingresados por el usuario.
3. Muestra los valores

* Primero importaremos las bibliotecas necesaria que es el estándar de entrada y salida y el estándar de bibliotecas



* Declarar el método principal



* Después declarar dos variables una que se usara para moverse entre posiciones y otra que almacenara el número de posiciones que ingresara el usuario



* Un apuntador de tipo int (que será el arreglo)



* Pedir al usuario el número de la longitud de el arreglo entero



* Aquí el apuntador tomara con la función malloc los espacios necesarios en memoria para almacenar el número ingresado por el usuario, que como son enteros tomaran 4 bytes



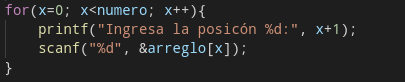
* Si la memoria no tiene espacio necesario para almacenar lo pedido por la función malloc el programa habrá terminado



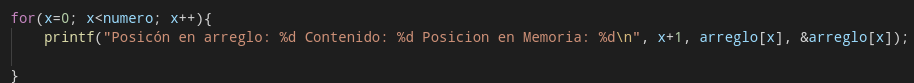
* Iniciar un ciclo for que que se repetirá desde 0 a el número ingresado por el usuario



* En el ciclo hira aumentando la posición de memoria y el usuario hira ingresando los números que pidió, así se recorrerá en este caso



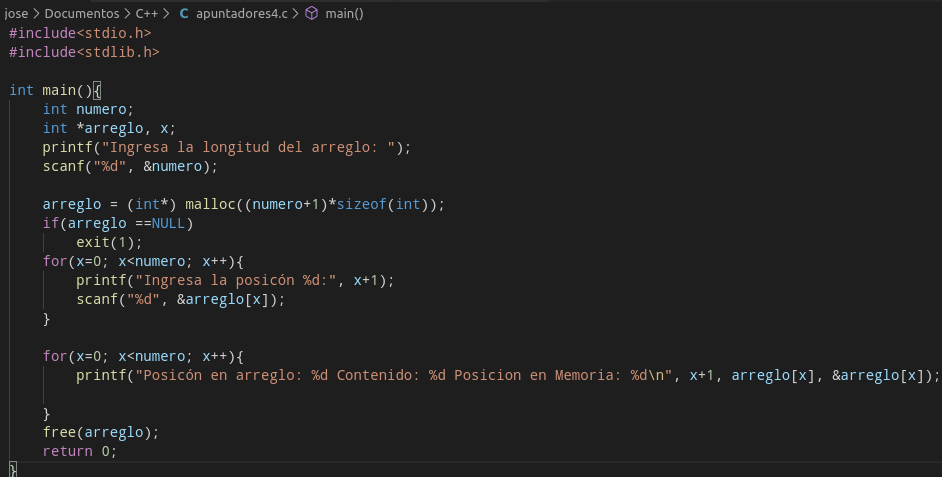
* Después se hará otro ciclo para mostrar los número ingresados con su posición en memoria



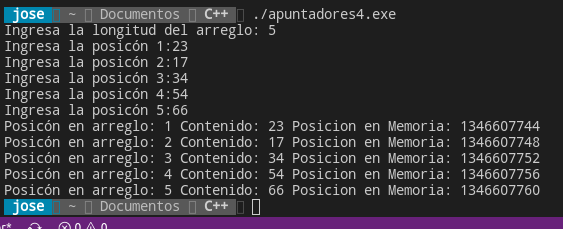
* Por último se liberara el arreglo



* Código completo



* Guardamos el código con el nombre de apuntadores4.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejemplo 6 Direcciones del arreglo**

* Importar la biblioteca del estándar de entrada y salida



* Declarar un arreglo entero de diez posiciones y una variable entera



* Declarar un arreglo de punto flotante de 10 posiciones



* Declarar el método principal de tipo entero



* Después imprimir un mensaje que funcionara como titulo de nuestra tabla



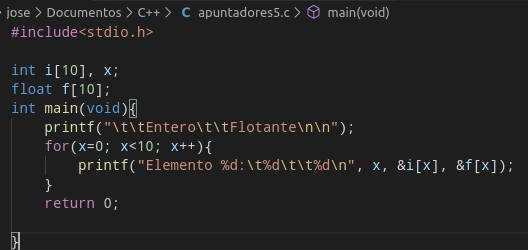
* Declararemos un ciclo for para recorrer las posiciones de los dos arreglos, para esto la variable x será el contador



* Imprimir la posición en donde esta y la posición en memoria de los dos arreglos

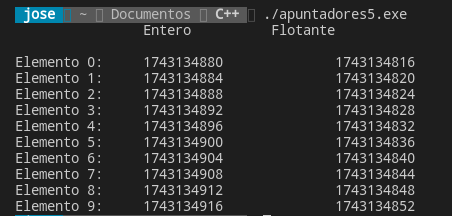


* Código completo



* Guardamos el código con el nombre de apuntadores5.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo

Como vemos las direcciones de los arreglos son contiguas



**Ejemplo 7 arreglo con memoria dinámica**

* Primero importaremos las bibliotecas necesaria que es el estándar de entrada y salida y el estándar de bibliotecas



* Declarar el método principal



* Después declarar dos variables una que se usara para moverse entre posiciones y otra que almacenara el número de posiciones que ingresara el usuario



* Declarar dos punteros de tipo entero, uno será utilizado para pedir el espacio en memoria y otro será el respaldo



* Pedir el número al usuario por teclado y almacenarlo en n



* En el apuntador buffer hará la petición de memoria del número que ingreso el usuario y multiplicado por el tipo entero (4 bytes)



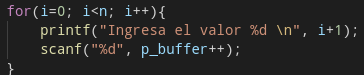
* Hará la comprobación si tiene la suficiente memoria y si no saldrá del programa



* Hacer el respaldo de buffer en p\_buffer



* Se inicializara un ciclo for para recorrer posiciones y se pedirá los números al usuario



* Se volverá a la primera posición de la memoria dinamica



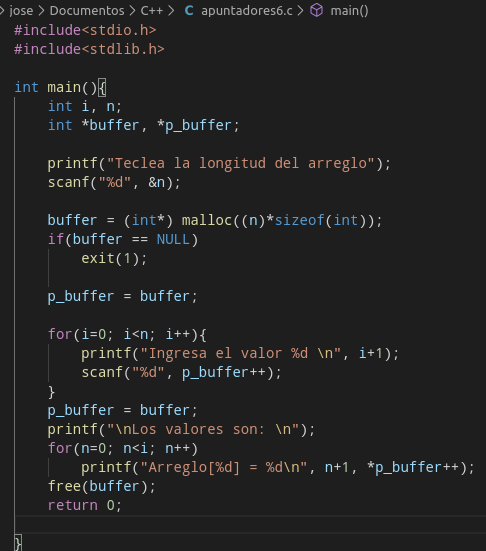
* Se inicializara otro ciclo for para imprimir los datos obtenidos recorriendo la memoria



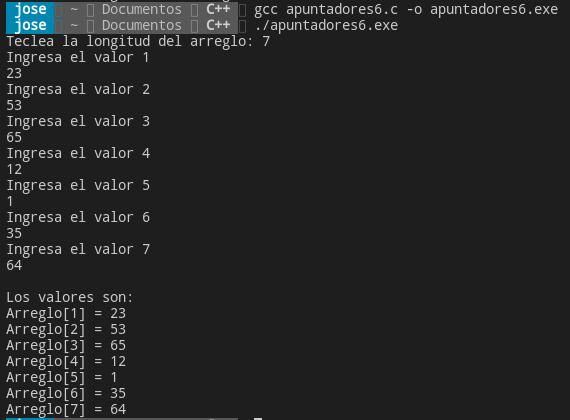
* Liberar lo almacenado en la dirección de memoria buffer



* Código completo



* Guardamos el código con el nombre de apuntadores6.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**Ejercicio 2**

1. Crea un arreglo de tipo char de tamaño x, en donde x es ingresado por teclado.
2. Llena elemento por elemento del arreglo con letras ingresados por el usuario.
3. Muestra el arreglo impreso en forma inversa.
4. Todo debe ser manejado con apuntadores**.**

* Primero importaremos las bibliotecas necesaria que es el estándar de entrada y salida y el estándar de bibliotecas



* Declarar el método principal



* Después declarar dos variables una que se usara para moverse entre posiciones y otra que almacenara el número de posiciones que ingresara el usuario



* Declarar dos punteros de tipo char, uno será utilizado para pedir el espacio en memoria y otro será el respaldo



* Pedir el número al usuario por teclado y almacenarlo en numero



* En el apuntador apuntador1 hará la petición con la función malloc de memoria del número que ingreso el usuario y multiplicado por el tipo carácter (1 byte)



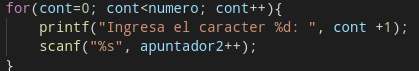
* Hará la comprobación si tiene la suficiente memoria y si no saldrá del programa



* Hacer el respaldo de apuntador1 en apuntador2



* Se inicializara un ciclo for para recorrer posiciones y se pedirá los caracteres al usuario que pidió



* Se volverá a la primera posición de la memoria dinámica



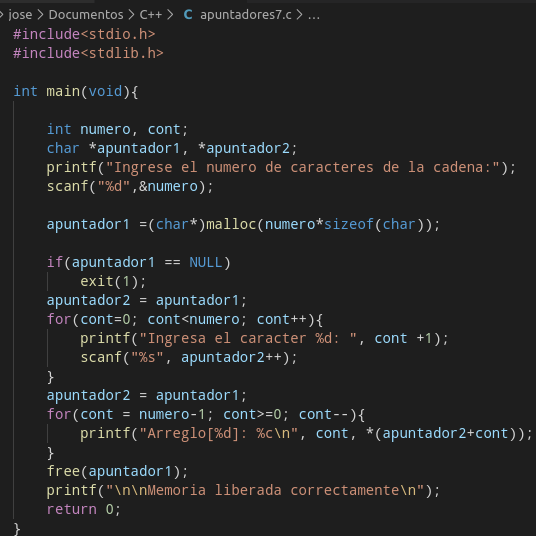
* Se inicializara otro ciclo for para imprimir de forma inversa los datos obtenidos recorriendo la memoria



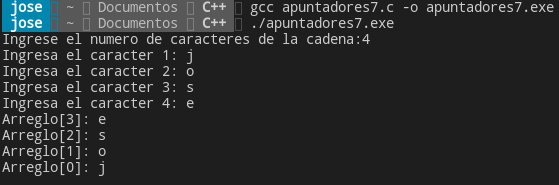
* Liberar lo almacenado en la dirección de memoria apuntador1



* Código completo



* Guardamos el código con el nombre de apuntadores7.c, lo compilaremos y creare un .exe con el mismo nombre, después ejecutarlo



**IV. Conclusión**

Después de haber realizado los ejemplos y ejercicios propuestos en la lámina 14 de apuntadores llego a la conclusión que este es el mayor plus de los lenguajes de bajo nivel ya que permiten y no simulan comunicarse con la memoria del ordenador, haciendo que el proceso de asignación de memoria sea mucho más rápido y eficiente, aunque aún falta ver muchas más instrucciones para el manejo y manipulación de la misma.