|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Nombre de la práctica** | **Apuntadores** | | | | **No.** | | **1** |
| **Asignatura:** | **Métodos Numéricos** | **Carrera:** | **INGENIERÍA EN SISTEMAS COMPUTACIONALES** | **Duración de la práctica (Hrs)** | | **2** | |

Nombre del alumno:García López María Guadalupe

Grupo:3042

**I. Competencia(s) específica(s):**

**II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):**

* otro

**III. Material empleado:**

* Computadora
* Visual estudio code

**IV. Desarrollo de la práctica: antes de empezar a dar a conocer el tema de apuntadores es importante mencionar que es un apuntador en cual es un puntero es un objeto que apunta a otro objeto. Es decir, una variable cuyo valor es la dirección de memoria de otra variable.**



En C nose debe indicar numéricamente la dirección de la memoria, sino que se usa una etiqueta que conocemos como variable.

Las direcciones de memoria dependen de la arquitectura del ordenador y de la gestión que el sistema operativo haga de ella.

¿Cómo se declaran los apuntadores?

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador‘\*’, y el nombre del apuntador.

Un puntero tiene su propia dirección de memoria.

La sintaxis es la siguiente:

<tipo de dato apuntador> \*<identificador del apuntador> int\* punt; char\* car; float\* num;

5

¿Cómo se declaran los apuntadores?

Al igual que el resto de las variables, los apuntadores se enlazan a tipos de datos específicos, de manera que a un apuntador sólo se le puede asignar direcciones de variables del tipo especificado en la declaración

Int\* punt; char\* car; float\* num;

**Tipos de apuntadores**

**Hay tantos tipos de apuntadores como tipos de datos.**

**Se puede también declarar apuntadores a estructuras más complejas. Funciones Struct Ficheros**

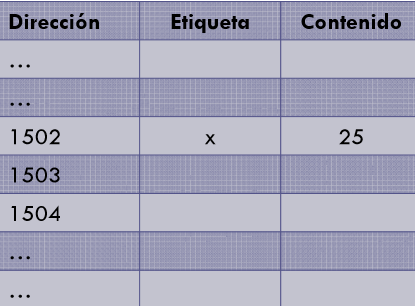
**Se pueden declarar punteros vacíos o nulos.**

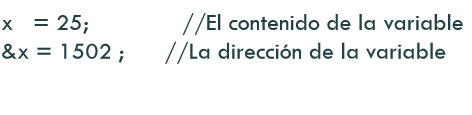
**¿Qué es la referenciación?**

**La referenciación es obtener la dirección de una variable.**

**Se hace a través del operador ‘&’, aplicado a la variable a la cual se desea saber su dirección &x ;//La dirección de la variable**

**¿Qué es la referenciación?**

**No hay que confundir una dirección de memoria con el contenido de esa dirección de memoria.**

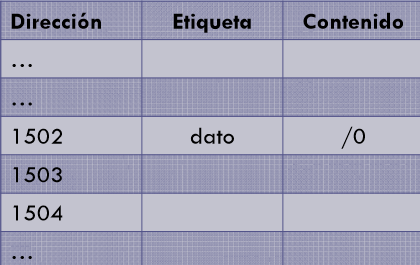
****

**Fragmento de código -referenciación**

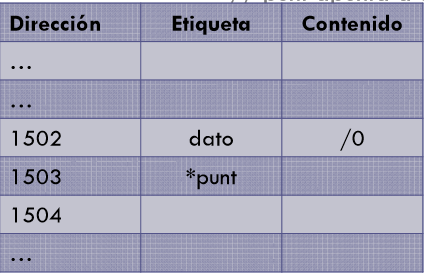
**int dato; //variable que almacenará un carácter.**

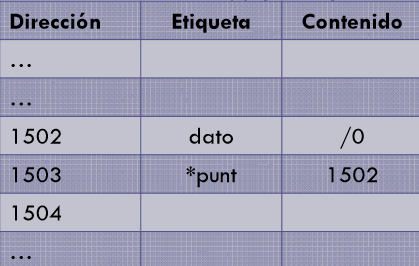
**Int \* punt; //declaración de puntero a carácter.**

**punt= &dato; //en la variable punt guardamos la dirección**

**//de memoria de la variable dato;**

**// punt apunta a dato.**

****

****

**punt= &dato; //en la variable punt guardamos la dirección**

**//de memoria de la variable dato;**

**// punt apunta a dato.**

**¿Qué es la desreferenciación?**

**Es la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador.**

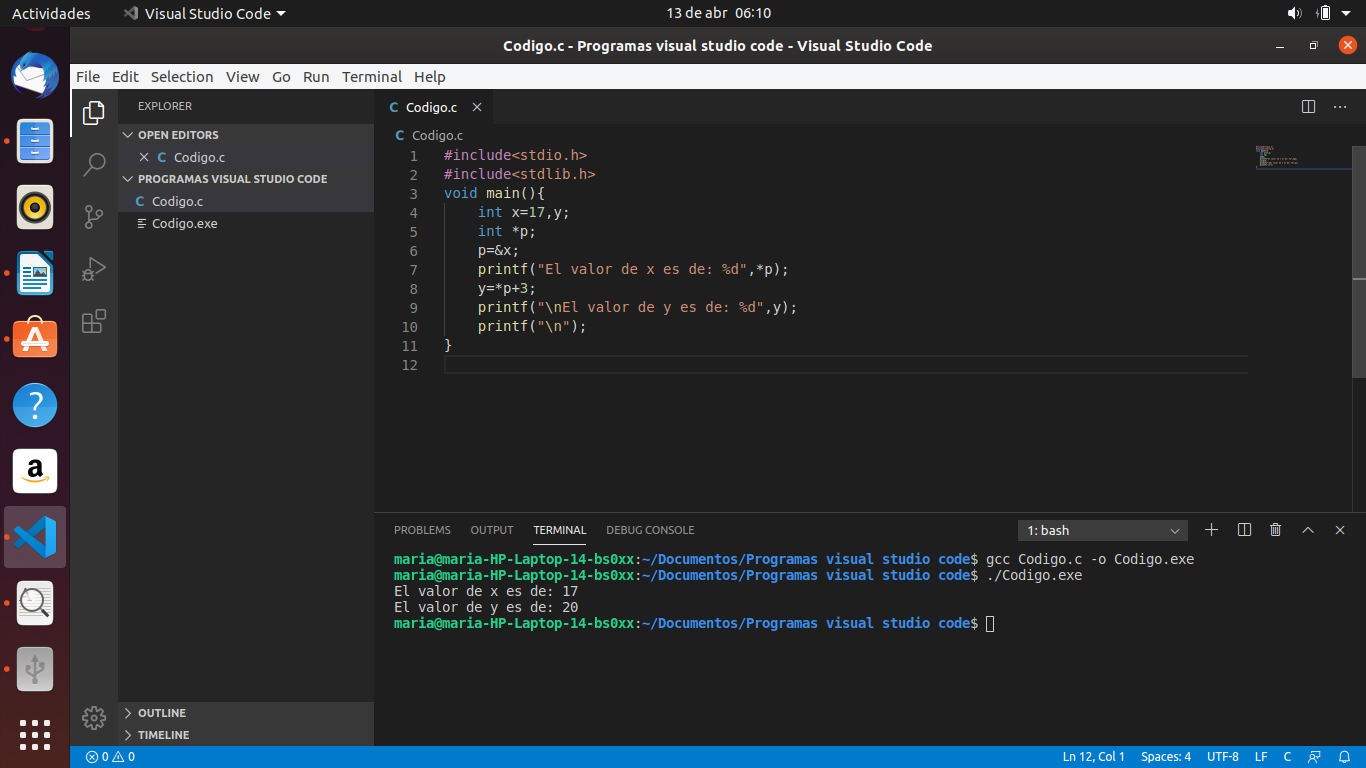
**Se hace a través del operador‘\*’, aplicado al apuntador que contiene la dirección del valor.**

**\*p ;//El contenido de p.**

**Ejemplo**

Con ayuda del Printf se imprimirá los valores de x e y

imprimir un salto de linea para que el código no se vea amontonado.

****

Bibliotecas

Se crean la variables de tipo entero una de ellas son inicializadas en este caso la x

Con la ayuda de printf imprimirá los valores de x e y

Creamos un salto de linea para que no se vea amontonado el código.

**Asignación de apuntadores**

**A un apuntador se pueden asignar direcciones de Variables a través del operador de referenciación**

**(‘&’) o direcciones almacenadas en otros apuntadores.**

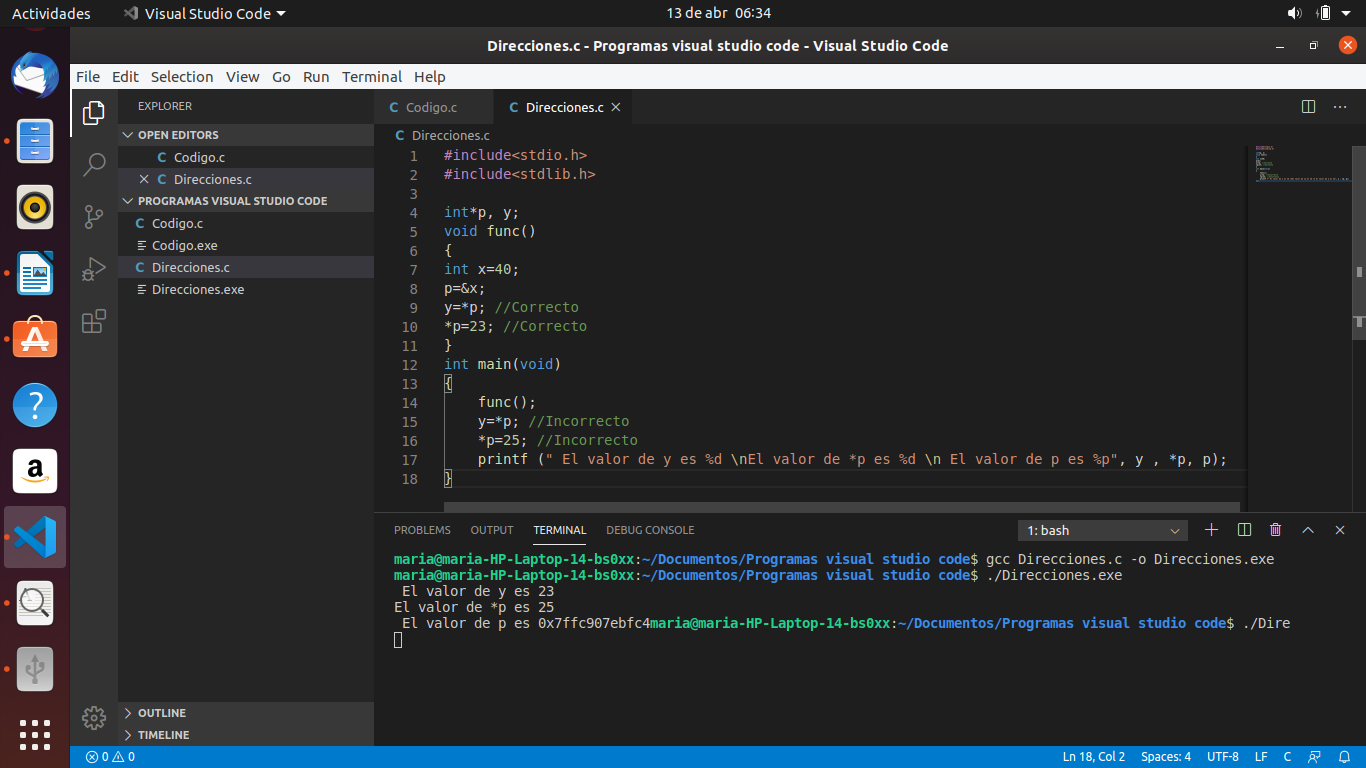
**Direcciones inválidas**

**Un apuntador puede contener una dirección inválida por:**

**Cuando se declara un apuntador, posee un valor cualquiera que no se puede conocer con antelación.**

**Después de que ha sido inicializado, la dirección que posee puede dejar de ser válida por que la variable asociada termina su ámbito o porque ese espacio de memoria fue reservado dinámicamente.**

**Ejemplo**

****

Se crean las bibliotecas

Fuera del metodo se crea la variable de tipo entera

Dentro del metodo se crean las variables

Printf ayuda a imprimir cada uno de los valores de x y de y.

**La dirección NULL**

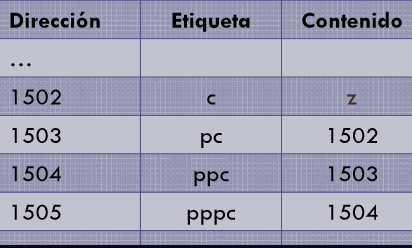
**Cuando no se desea que el apuntador apunte a algo, se le suele asignar el valor de NULL, en cuyo caso se dice que el apuntador es nulo (no apunta a nada).**

**NULL es una macro típicamente definida en archivos de cabecera como stdef.h y stdlib.h.**

**Se utiliza para proporcionar a un programa un medio de conocer cuándo un apuntador contiene una dirección inválida.**

**Apuntadores a apuntadores**

**Dado que un apuntador es una variable que apunta a otra, fácilmente se puede deducir que pueden existir apuntadores a apuntadores, y a su vez los segundos pueden apuntar a apuntadores.**

****

**charc = ‘z’;**

**charc\*pc= &c;**

**char\*\*ppc= &pc;**

**char\*\*\*pppc= &ppc;**

**\*\*\*pppc= ‘m’**

**Es posible declarar apuntadores a constantes. De esta manera, no se permite la modificación de la dirección almacenada en el apuntador, pero si se permite la modificación del valor al que apunta.**

****

**Int x = 5, y = 7;**

**int \* constp = &x; \*p=3;**

**p=&y;**

**Paso de parámetros por referencia**

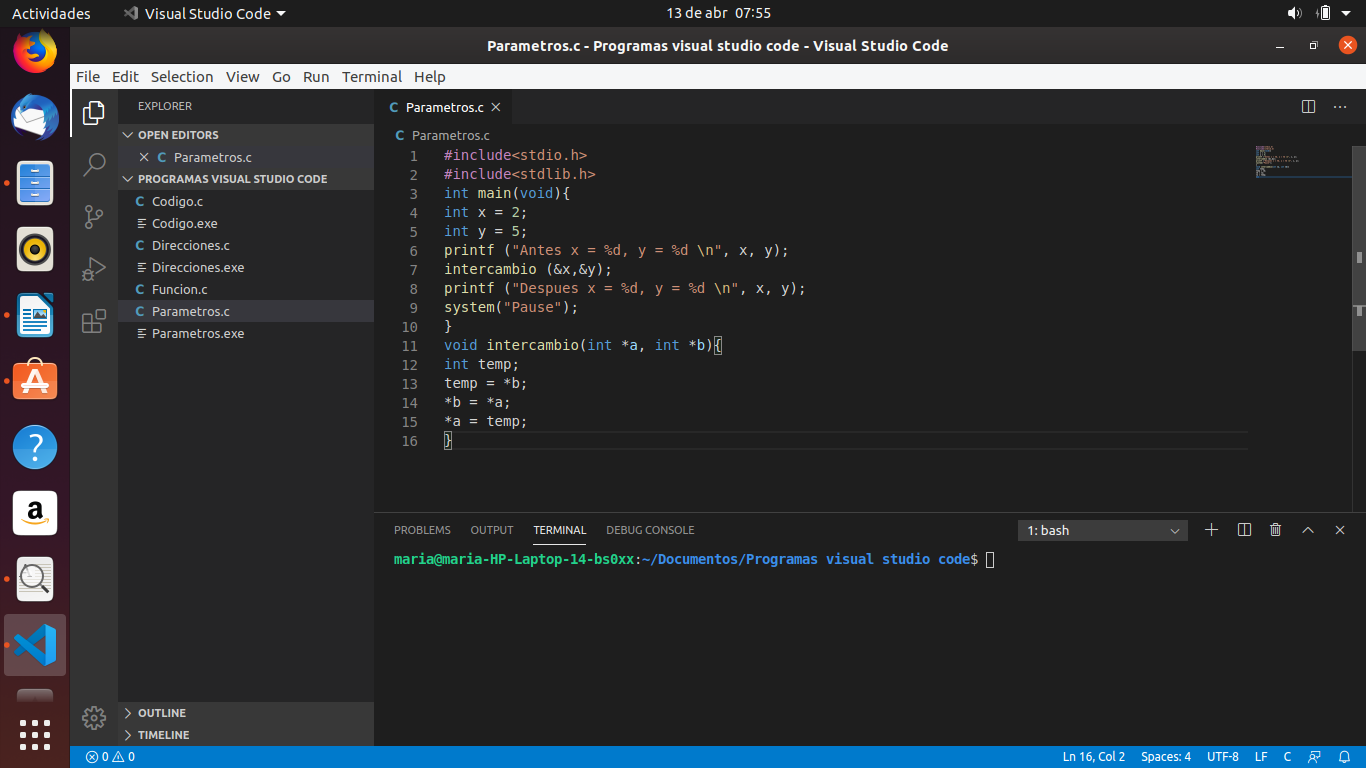
**En este tipo de llamadas los argumentos contienen direcciones de variables.**

**Dentro de la función la dirección se utiliza para acceder al argumento real.**

**En las llamadas por referencia cualquier cambio en la función tiene efecto sobre la variable cuya dirección se pasó como argumento.**

**Ejemplo**

Bibliotecas

****

Variables de tipo enteras

Dentro de otro método y dentro de este se almacenan las variables a utilizar.

**La función sizeof ()**

**Devuelve el tamaño en bytes que ocupa un tipo o variable en memoria.**

**Char cadena [10];**

**Printf (“un int ocupa %d bytes”, sizeof (int));**

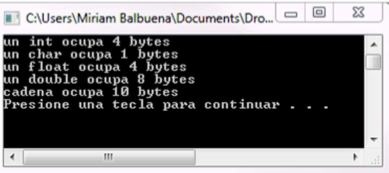
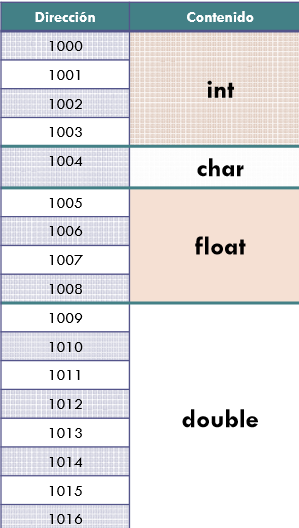
**Printf (“un char ocupa %d bytes”, sizeof (char));**

**Printf (“un float ocupa %d bytes”, sizeof (float));**

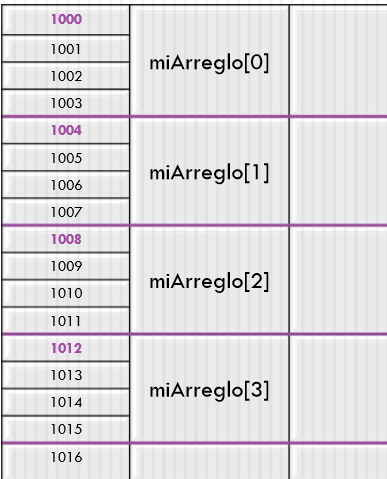
**Printf (“un double ocupa %d bytes”, sizeof (double));**

**Printf (“cadena ocupa %d bytes”, sizeof (cadena));**

**Función**

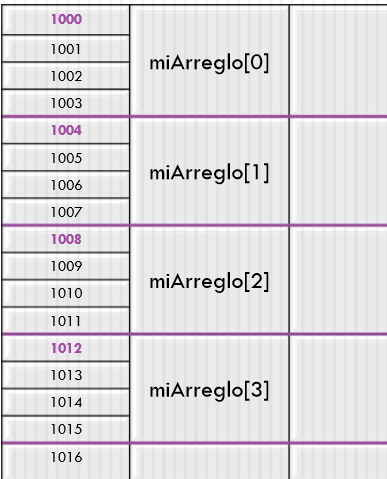


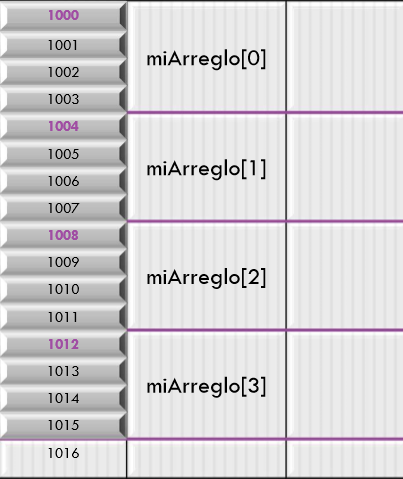
**Int mi Arreglo [4] = {1, 2, 3, 4}**

****

****

**int mi Arreglo [4] = {1, 2, 3, 4}**

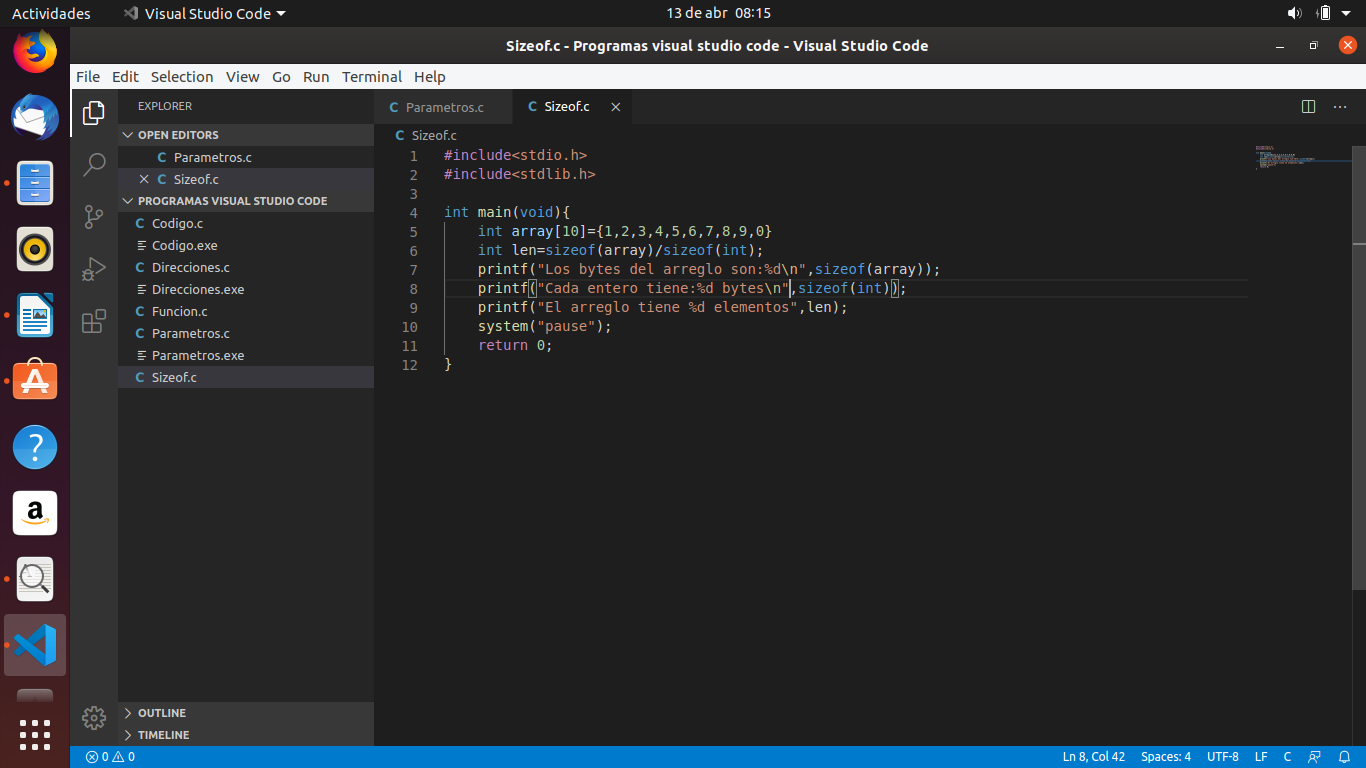
****

****

****

**Ejemplo**

bibliotecas

****

Se crea un arreglo donde después se mostraran las posiciones.

Se imprime cada uno de los valores.

**Asignación dinámica de memoria**

**Los programas pueden crear variables globales o locales.**

**Las variables declaradas globales en sus programas se almacenan en posiciones fijas de memoria (segmento de datos) y todas las funciones pueden utilizar estas variables.**

**Las variables locales se almacenan en la pila (stack) y existen solo mientras están activas las funciones donde están declaradas.**

**En ambos casos el espacio de almacenamiento se reserva en el momento de compilación del programa.**

**Para asignar memoria dinámicamente se utilizan las funciones malloc () y free (), definidas típicamente en el archivo stdlib.h.**



**free ()**

**La función free () permite liberar la memoria reservada a través de un apuntador.**

**Void free (void\* ptr);**

**ptres un puntero de cualquier tipo que apunta a un área de memoria reservada previamente con malloc.**

**malloc ()**

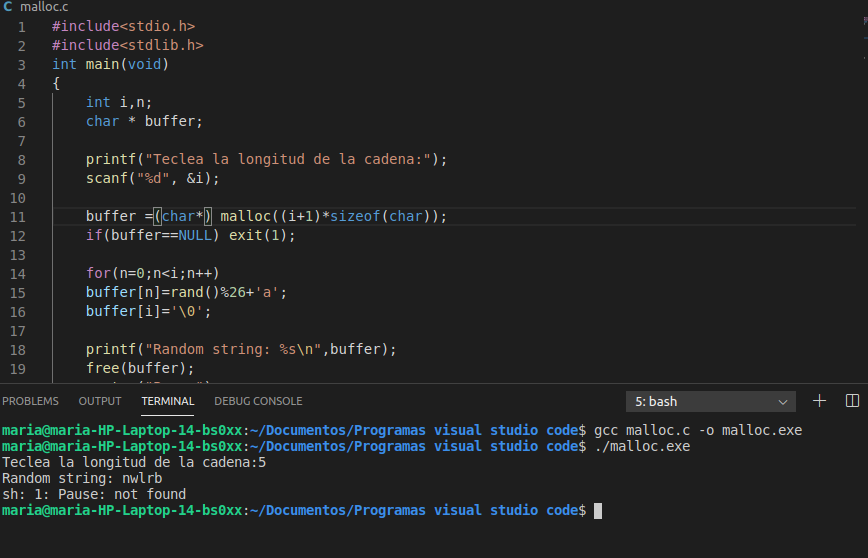
**La función malloc () reserva memoria y retorna su dirección, o retorna NULL en caso de no haber conseguido suficiente memoria.**

**Void\*malloc (size\_ttam\_bloque)**

**malloc () reserva memoria sin importar el tipo de datos que almacenará en ella.**

**Ejemplo**

Bliotecas

****

Apuntador llamado buffer

Ingresamos un mensaje que debe de teclear la longitud del arreglo

El buffer es igual a char lo reserva lógicamente donde i mas 1.

Es para evitar que haya desbordamiento de memoria.

Se hace un ciclo for para poder recorrer el arreglo.

**Ejercicio**

**Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.**

**Llena todos los elementos del arreglo con datos ingresados por el usuario.**

**Muestra los valores**

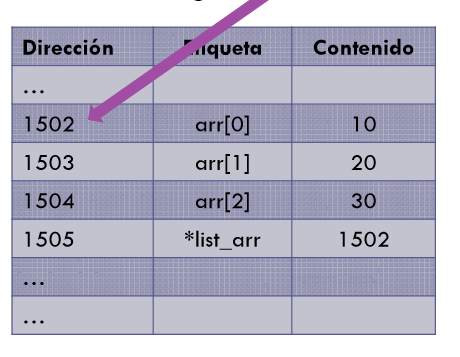
**Apuntadores a arreglos**

**El nombre de un arreglo es simplemente un apuntador constante al inicio del arreglo**

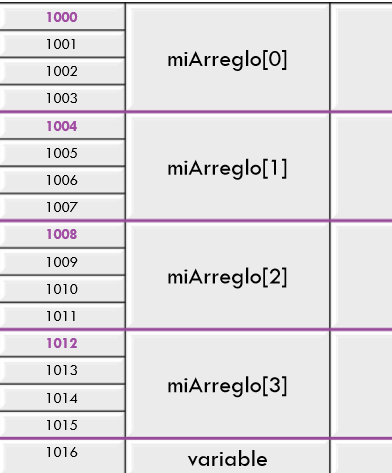
**Int lista\_arr [3]** **= {10, 20,30};**

**int \* lista\_ptr;**

**lista\_ptr= lista\_arr;**

****

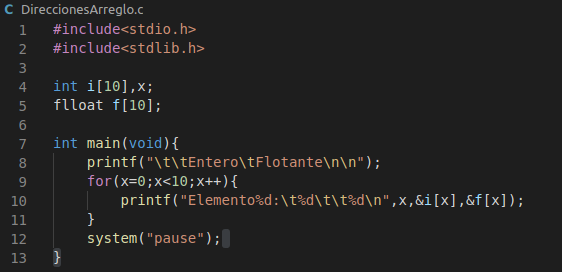
**Se crea el apuntador lista\_ptrpara poder ir modificando la dirección a donde apunta**

**miArreglo= 1000**

**&miArreglo[0] = 1000**

**&miArreglo [1] = 1004**

**Direcciones del arreglo**

****

bibliotecas

Se crea una variable de tipo entero y una variable de tipo float

Se imprime el valor entero y flotante dando saltos de linea

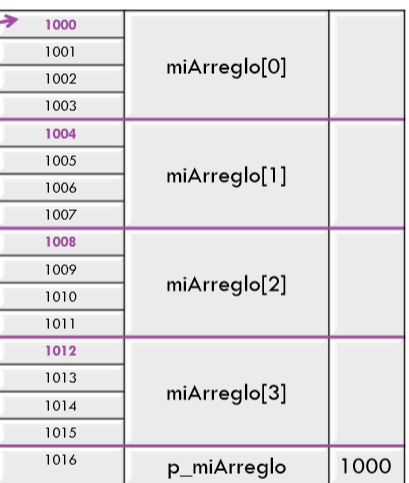
Con ayuda del for este inicia desde 0 y a su vez x es menor a 10 e incrementa.

**Aritmética de operadores**

**Incremento de operadores**

**Cuando se incrementa un apuntador se está incrementando su valor.**

**Si incrementamos en 1 el valor del apuntador, C sabe el tipo de dato al que apunta e incrementa la dirección guardada en el apuntador en el tamaño del tipo de dato.**

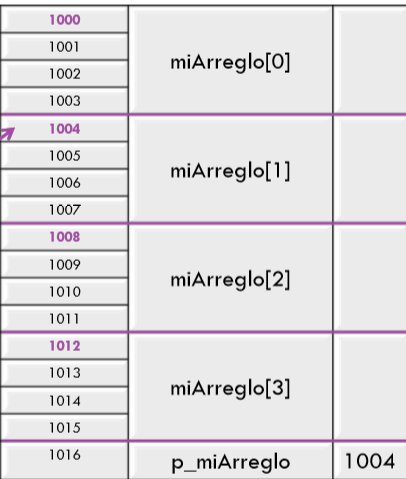
****

**miArreglo= 1000**

**p\_miArreglo= miArreglo**

**&miArreglo [0] = 1000**

**&miArreglo [1] = 1004**

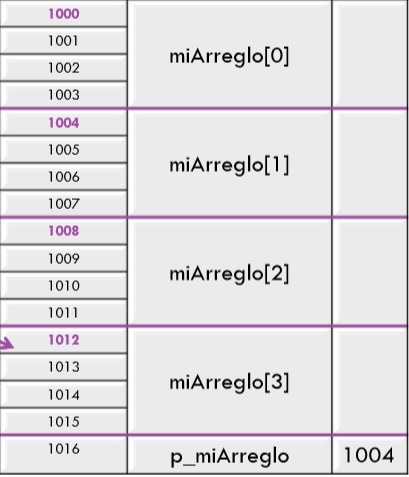
**miArreglo= 1000**

**p\_miArreglo= miArreglo**

**&miArreglo [0] = 1000**

**&miArreglo [1] = 1004**

**P\_miArreglo++;**

****

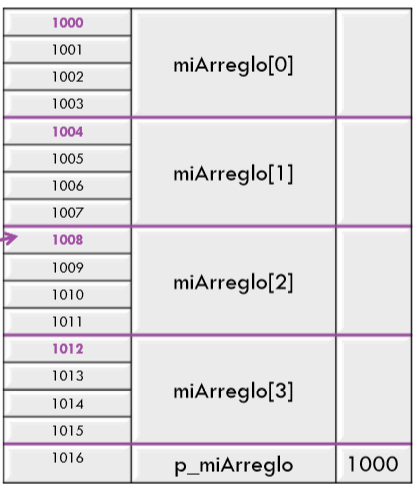
**miArreglo= 1000**

**p\_miArreglo= miArreglo**

**&miArreglo [0] = 1000**

**&miArreglo [1] = 1004**

**P\_miArreglo+= 2;**

****

**miArreglo= 1000**

**p\_miArreglo= miArreglo**

**&miArreglo [0] = 1000**

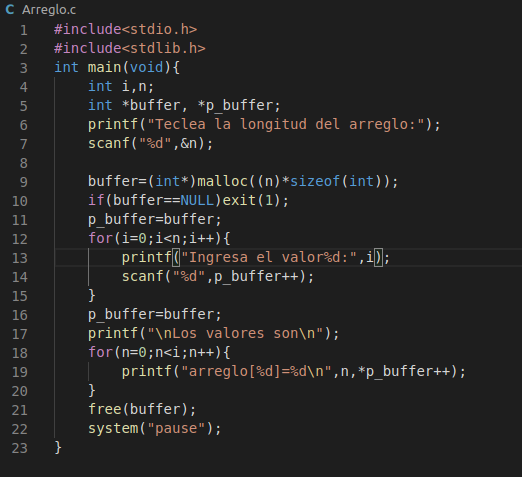
**&miArreglo [1] = 1004**

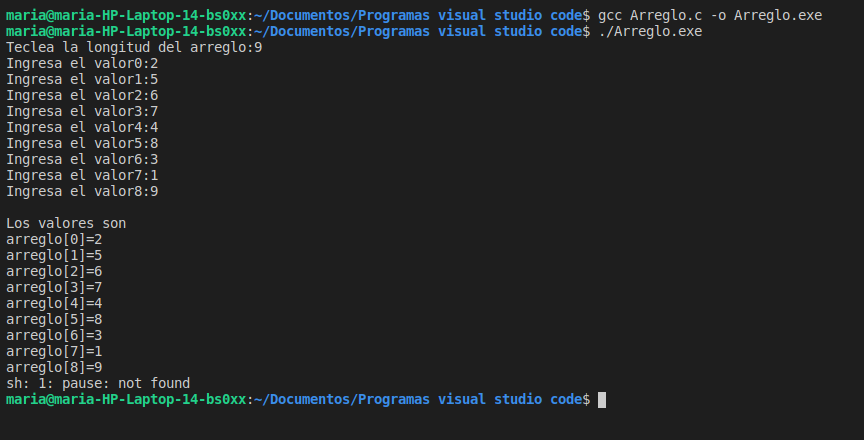
**P\_miArreglo--;**

**Ejercicio**

**Crea un arreglo entero de tamaño x, en donde x es ingresado por teclado.**

**Llena todos los elementos del arreglo con datos ingresados por el usuario usando apuntadores.**

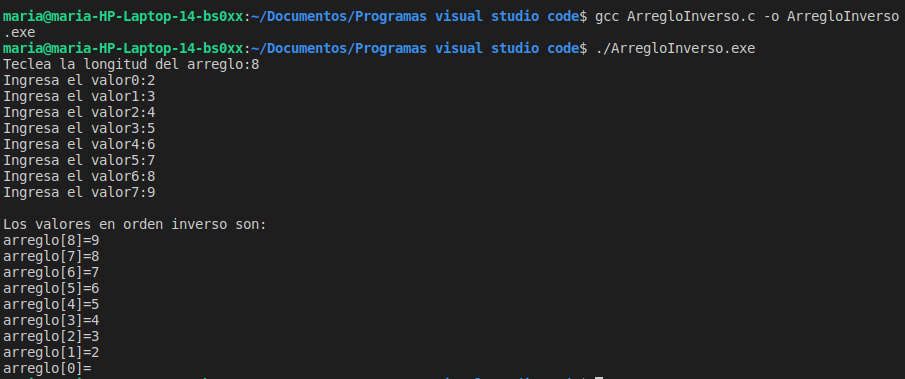
****

****

**getchar();**

**Ejercicio**

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

****

**V. Conclusiones:**

Los apuntadores son importantes ya que sirven para reducir los ciclos de reloj un apuntador es un tipo de dato, es decir que guarda un conjunto de números, normalmente en la practica este tipo de dato se usa para representar la dirección de memoria de variables que se puedan usar durante el código.

Sin embargo este tipo de datos son uno de los mas temidos se puede decir así , pero es uno de los mas usados y versátiles y de los que mayor potencia proporciona.

Un puntero es una variable

se me hizo muy complicado ya que no entiendo algunos términos pero de tal manera estoy aprendiendo.