1. Punteros

|  |
| --- |
| Los punteros son una de las poderosas herramientas que ofrece el lenguaje **C** a los programadores, sin embargo, son también una de las más peligrosas, el uso de punteros sin inicializar, etc., es una fuente frecuente de errores en los programas de **C**, y además, suele producir fallas muy difíciles de localizar y depurar. |

* 1. Definición

|  |
| --- |
| Un puntero es una variable que contiene una dirección de memoria, normalmente esa dirección es una posición de memoria de otra variable, por lo cual se suele decir que el puntero “apunta” a la otra variable. |

* 1. Declaración de un puntero

|  |
| --- |
| La sintaxis de la declaración de una variable puntero es: |

|  |
| --- |
| **tipo*\**** nombre***;*** |

|  |
| --- |
| El “**tipo**” base de la declaración sirve para conocer el tipo de datos de la variable apuntada por el puntero.  Esto es fundamental para poder leer el valor que almacena la zona de memoria apuntada por el puntero y para poder realizar ciertas operaciones aritméticas sobre los mismos. |

|  |
| --- |
| Algunos ejemplos de declaración de variables puntero son: |

|  |
| --- |
| **int*\**** a***;* char*\**** p***;* float*\**** f***;*** |

* 1. Operadores

|  |
| --- |
| Existen dos operadores especiales de los punteros:   * El operador ' & ' encargado de retornar la dirección de memoria de una variable. * El operador ' \* ' encargado de retornar el valor de la variable apuntada por un puntero.   Si declaramos:  **int*\**** a***;***  **int** b***;***  Y hacemos:    a ***= &***b***;*** |
| La variable puntero **'a'** contendrá la dirección de memoria de la variable **'b'**.  Veámoslo en otro ejemplo:  **int*\**** a***;***  **int** b***;***  **int** c***;***  Y hacemos***:***    b ***=*** 15***;*** // Asignamos el valor 15 a la variable 'b'    a ***= &***b***;*** // Obtenemos la posicion de memoria de 'b' con el operador '&'    c ***= \****a***;*** // Copiamos el contenido apuntado por el puntero 'a'  Entonces la variable **'c'** contendrá el valor 15.  **'*\**a'** devuelve el contenido de la dirección a la que “apunta” la variable puntero **'a'** que con anterioridad hemos hecho que contenga la “dirección de memoria” de la variable **'b'** usando para ello el operador **'&'** |

* 1. Asignación de punteros

|  |
| --- |
| Es posible asignar el valor de una variable de tipo puntero a otra variable de tipo puntero.  Por ejemplo: |

|  |
| --- |
| **int*\**** a***;***  **int*\**** b***;***  **int** c***;***  a ***= &***c***;*** // Obtenemos la posición de memoria de 'c' con el operador '&'  b ***=*** a***;*** // Asignamos el valor del puntero 'a' al puntero 'b' |

|  |
| --- |
| Entonces **'b'** contiene el valor de **'a'**, por lo tanto, **'b'** también “apunta” a la variable **'c'**. |

* 1. Comparación de punteros

|  |
| --- |
| Sobre las variables de tipo puntero es posible realizar operaciones de comparación.  veamos un ejemplo: |
| **int*\**** punteroA***;***  **int*\**** punteroB***;***  **int** auxiliarC***;***  **int** auxiliarD***;***  punteroA ***= &***auxiliarC***;*** // Obtenemos la posicion de memoria de 'auxiliarC'  punteroB ***= &***auxiliarD***;*** // Obtenemos la posicion de memoria de 'auxiliarD'  **if *(***punteroA ***<*** punteroB***)***  ***{***  printf***(***"El punteroA apunta a una direccion mas baja que punteroB"***);***  ***}***  **else if *(***punteroA ***>*** punteroB***)***  ***{***  printf***(***"El punteroB apunta a una direccion mas baja que punteroA"***);***  ***}*** |

* 1. Aritmética de punteros

|  |
| --- |
| Sobre las variables de tipo puntero es posible utilizar los operadores ***+***, ***-***, ***++*** y ***--*** .  Estos operadores incrementan o decrementan la posición de memoria a la que “apunta” la variable puntero.  El incremento o decremento se realiza de acuerdo al **“tipo base”** de la variable de tipo puntero, de ahí la importancia del tipo del que se declara la variable puntero.  Veamos esto con la siguiente tabla: |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **char\*** a | **int\*** a | **float\*** a |
| a ***=*** a ***-***3 | 0xA07D | 0xB06E | 0xC074 |
| a ***=*** a ***-***2 | 0xA07E | 0xB078 | 0xC078 |
| a***--*** | 0xA07F | 0xB07C | 0xC07C |
| Posición inicial | 0xA080 | 0xB080 | 0xC080 |
| a***++*** | 0xA081 | 0xB084 | 0xC084 |
| a ***=*** a ***+***2 | 0xA082 | 0xB088 | 0xC088 |

|  |
| --- |
| Como podemos observar en la tabla cada operación sobre el puntero se rige por su tipo, por lo tanto, si tenemos: |

|  |
| --- |
| **tipo*\**** a***;***  Y hacemos:  a ***=*** a ***+*** num***;***  La posicion a la que apunta a se incrementa en:    contenido de “a” ***=*** direccion que contiene “a” ***+ (***num ***\** sizeof*(*tipo*))***    Para la resta se decrementa de igual forma en:    contenido de “a” ***=*** direccion que contiene “a” ***- (***num ***\** sizeof*(*tipo*))***  Los operadores ***++*** y ***--*** son equivalentes a realizar num ***=*** 1 , y con ello quedan obviamente explicados. |

* 1. Vectores y punteros

|  |
| --- |
| Existe una estrecha relación entre los punteros y los vectores.  Consideremos el siguiente fragmento de código:  **char** cadena[80];  **char*\**** p***;***  p ***= &***cadena[0];// equivalente a: p = cadena  Este fragmento de código pone en la variable puntero **'p'** la dirección del primer elemento del array **'cadena '**.  Entonces, es posible acceder al valor de la quinta posición del array mediante:  cadena***[***4***]*** y ***\*(***p ***+*** 4***)***  *(recuérdese que los índices de los arrays empiezan en 0).*  Esta estrecha relación entre los arrays y los punteros queda más evidente si se tiene en cuenta que el nombre del array sin el índice es la dirección de comienzo del array, y si además, se tiene en cuenta que un puntero puede indexarse como un array unidimensional, por lo cual, en el ejemplo anterior, podríamos referenciar ese elemento como:  p[4]  Es posible obtener la dirección de un elemento cualquiera del array de la siguiente forma:  **int** arrayInt[80];    **int*\**** p1***;***  **int*\**** p2***;***  p1 ***=*** &arrayInt[4];  p2 ***=*** &arrayInt;  Entonces, el puntero **'p1'** contiene la dirección del quinto elemento del **'arrayInt'** y el puntero **'p2'** contiene la dirección del primer elemento del arrayInt.  Hasta ahora hemos declarado variables puntero aisladas. Es posible, como con cualquier otro tipo de datos, definir un array de variables puntero.  La declaración para un array de punteros **int** de tamaño **10** es:  **int*\**** a[10];  Para asignar una dirección de una variable entera, llamada **'var'**, al tercer elemento del array de punteros, se escribe:  a[2] ***=*** &var;  Y para obtener el valor de **'var'**:  ***\****a[2];  Dado además, que un puntero es también una variable, es posible definir un puntero a un puntero.  Supongamos que tenemos lo siguiente:  **int** a***;***  **int*\**** punteroInt***;***  **int*\*\**** punteroPuntero***;***  punteroInt ***= &***a***;*** // Obtenemos la posicion de memoria  punteroPuntero ***= &***puntero***;*** // Obtenemos la posicion de memoria |

|  |  |  |
| --- | --- | --- |
| POSICION | VARIABLE | CONTENIDO |
| 0xB06E | **int** a***;*** | 2 |
| 0xB078 |  |  |
| 0xB07C | **int*\**** punteroInt***;*** | 0xB06E |
| 0xB080 |  |  |
| 0xB084 | **int*\*\**** punteroPuntero***;*** | 0xB07C |
| 0xB088 |  |  |

|  |
| --- |
| Y entonces,  ¿ De qué formas podemos ahora acceder al valor de la variable **'a'** ?  a *(forma habitual)*  ***\****punteroInt *(a través del puntero)*  ***\*\****punteroPuntero *(a través del puntero a puntero)*  Esto es debido a que **'punteroPuntero'** contiene la dirección de **'punteroInt'**, que contiene la dirección de **'a'**.  Este concepto de puntero a puntero podría extenderse a puntero a puntero a puntero, etc.,  además, existe el concepto de puntero a una función, al cual nos referiremos más adelante. |

1. Punteros II
   1. Tipos de Punteros

|  |
| --- |
| Un puntero es una variable como cualquier otra destinada a guardar una posición de memoria, esta posición de memoria generalmente corresponde a otra variable.  Lo que se pretende decir es que si a un puntero se le carga cualquier valor, éste será interpretado como si fuera una dirección de memoria aunque el valor cargado no corresponda a una dirección de memoria. |

* + 1. Punteros a cadenas de caracteres

|  |
| --- |
| Un puntero a “string” no es más que un puntero a **char**. Por lo tanto la forma de definir un puntero a una cadena de caracteres es:  **char*\**** p***;***  Por ejemplo la siguiente línea declara un puntero a **char** y un vector para guardar la cadena de caracteres.  **char** cadena***[***N\_LONGITUD***];***  **char*\**** puntero***;***  Así se reserva espacio para una cadena de N\_LONGITUD caracteres.  En C, el nombre de las listas monodimensionales equivale a la dirección del primer elemento de las mismas.  Una vez que se dispone de espacio reservado para la cadena, se puede construir un puntero de la cadena empleando las siguientes sentencias:  **char** cadena***[***LONGITUD***];***  **char*\**** puntero***;***  puntero ***=*** cadena*;*// es equivalente a escribir puntero = &cadena[0]  Esto dará lugar a que en **'puntero'** se almacene la dirección de **'cadena'**, es decir, se le asigna a **'puntero'** el valor de la dirección de un carácter, que va a ser el primero de la cadena. |

* + - 1. Recorrido de una cadena mediante aritmética de punteros e índices.

|  |
| --- |
| El mecanismo de la aritmética de punteros en C tiene su expresión más sencilla en el caso de utilizar, precisamente, punteros de caracteres.  Consideremos las siguientes declaraciones:  **tipo** lista***[***NUMERO\_DE\_ELEMENTOS***];***  **tipo*\**** puntero***;***  puntero ***=*** lista***;***  Se debe recordar que en general, cuando se suma 1 a un puntero, el valor numérico de la dirección no se incrementa en 1, sino en:  **sizeof*(***tipo***)***  Entonces, al sumar **'i'** a un puntero se tiene que la dirección obtenida al calcular **'puntero +i'** tiene el valor:  direccion\_actual\_del\_puntero ***+ (***i ***\** sizeof*(***tipo***))***  Dando como resultado un nuevo puntero, que señala una posición de memoria que correspondería a un elemento de una lista situado en **'i'** posiciones más allá del elemento señalado por **'puntero'**.  Se detalla a continuación un ejemplo en el cual aplicando aritmética de punteros recorreremos una cadena de caracteres :  **void** mostrarCadena***(* char*\**** punteroCadena ***)***  ***{***  **while*( \****punteroCadena ***!=*** '\0' ***)***  ***{***  printf***(*** "%c"***, \****punteroCadena ***);***  punteroCadena***++;***  ***}***  ***}***  Como se puede observar, la condicion del bucle indica que mientras que sea el contenido de la posicion apuntada por **'punteroCadena'** distinto a **'\0'** se continuará iterando, en cada iteración se imprime el contenido de la posición apuntada por **'punteroCadena'** y se incrementa el valor del puntero mediante **'punteroCadena++'**. |

* + 1. Punteros a estructura

|  |
| --- |
| Como una estructura es una variable, se la puede apuntar con un puntero. De acuerdo a la definición de puntero, deberá guardar la dirección de memoria de la estructura , si pensamos que una estructura puede estar formada por un grupo de variables distintas, el puntero tendrá la dirección de comienzo del grupo de variables.  Veamos un ejemplo: |

|  |
| --- |
| **typedef sruct**  ***{***  **char** nombre***[***20***];***  **int** nota***;***  ***}***sAlumno***;***  sAlumno auxiliarAlumno***;***  sAlumno***\**** punteroAlumno***;***  punteroAlumno ***= &***auxiliarAlumno***;*** |

|  |
| --- |
| Si se piensa en un vector de estructuras y se almacena en un puntero, del tipo de esa estructura, la posición de memoria que corresponde al comienzo de dicho vector, al momento de recorrerlo, utilizando el puntero, en cada incremento pasará a apuntar a la próxima estructura almacenada en el array. Lo recién enunciado cumple con la aritmética de punteros explicada anteriormente.  Veamos un ejemplo: |

|  |
| --- |
| #define CANTIDAD 5  **typedef struct**  ***{***  **char** nombre***[***20***];***  **int** nota***;***  ***}***sAlumno***;***  **void** main***(* void *)***  ***{***  **int** i***;***    sAlumno arrayAlumnos***[***CANTIDAD***];***  sAlumno***\**** punteroArrayAlumnos***;***    punteroArrayAlumnos ***=*** arrayAlumnos***;***    **for*(***i ***=*** 0***;*** i ***<*** CANTIDAD***;*** i***++)***  ***{***  punteroArrayAlumnos***++;*** //...  ***}***  ***}*** |

* + - 1. Acceso a los campos: Operador flecha (->)

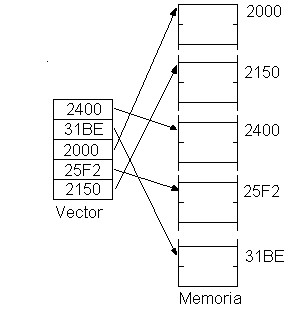
|  |
| --- |
| Cuando se usan los punteros a estructura la forma de acceder a un campo es usando el operador FLECHA ( ***->*** )el cual se forma escribiendo un guion y a continuacion un mayor.  Veamos un ejemplo: |

|  |
| --- |
| **typedef struct**  ***{***  **char** nombre***[***20***];***  **int** nota***;***  ***}***sAlumno***;***  **void** main***(*void*)***  ***{***  sAlumno auxiliarAlumno***;***  sAlumno***\**** punteroAlumno***;***    punteroAlumno ***= &***auxiliarAlumno***;***    strcpy***(*** punteroAlumno***->***nombre***,*** "Ernesto" ***);***  punteroAlumno***->***nota ***=*** 10***;***  ***}*** |

|  |
| --- |
| Como se puede ver al momento de querer acceder al campo “nota” de la estructura “auxiliarAlumno” mediante el puntero “punteroAlumno” , se remplaza el operador punto (auxiliarAlumno.nota) por el operador flecha (punteroAlumno->nota). |

* + - 1. Vector de punteros

|  |
| --- |
| Un vector de punteros es simplemente un vector que dentro de cada uno de sus elementos existe un puntero.  La forma de definir un vector de punteros es:  **tipo*\**** nombre***[***MAX***];***  En el caso que se requiera definir un vector de punteros **int**.  **int*\**** vec ***[***MAX***];***  El vector tiene posiciones de memoria (punteros) de algunas variables, por ejemplo, el elemento vec***[***i***]*** contiene la dirección de memoria de una variable y por lo tanto el contenido de ***\****vec***[***i***]*** será un entero.  Al igual que se inicializaba un puntero, se debe inicializar el vector de punteros para que cada posición del vector apunte a algún lugar determinado.  Veamos el siguiente dibujo que puede ilustrar mejor la situación. |



|  |
| --- |
| A la izquierda del dibujo tenemos el vector “vec” que es un vector de punteros , por lo tanto cada elemento del vector es una dirección de memoria (puntero). Inicialmente como pasa en cualquier variable tiene un valor cualquiera al momento de arrancar el programa, por lo cuál los punteros están apuntando a cualquier posición.  Normalmente se asocia un vector de punteros a otro de estructuras, es decir que el vector de punteros tiene las direcciones de cada uno de los elementos del vector de estructuras. Veamos una utilidad del vector de punteros.  Supongamos que se debe realizar un programa de agenda que cargue los datos de 100 alumnos y luego tenga que ordenar los datos según su nota. En el proceso de ordenamiento lo que se hace es copiar el elemento **'i'** a un **'auxiliar'** , luego copiar el elemento **'j'** al elemento **'i'** y finalmente el **'auxiliar'** al elemento **'j'** , o sea está realizando 3 copias de estructuras lo cual implica un movimiento importante de datos. |

|  |
| --- |
| **for*(***i ***=*** 0***;*** i ***<*** CANTIDAD ***-***1***;*** i***++)***  ***{***  **for*(***j ***=*** i ***+*** 1***;*** i ***<*** CANTIDAD***;*** j***++)***  ***{***  **if*(***arrayAlumnos***[***i***].***nota ***>*** arrayAlumnos***[***j***].***nota***)***  ***{***  auxiliarAlumno ***=*** arrayAlumnos***[***i***];***  arrayAlumnos***[***i***] =*** arrayAlumnos***[***j***];***  arrayAlumnos***[***j***] =*** auxiliarAlumno***;***  ***}***  ***}***  ***}*** |

|  |
| --- |
| Ahora si usted tiene un vector de punteros, el movimiento de datos se realiza en el vector de punteros, con lo cuál la copia se reduce notablemente (la cantidad de bytes a copiar se reduce a 2 bytes que es el tamaño de un puntero) y por otra parte no se modifica el vector de estructuras. Cuando se quiere mostrar los datos ordenados simplemente se recorre el vector de estructuras.  Veamos un ejemplo: |

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #define CANTIDAD 3  **typedef struct**  {  **char** nombre[50];  **int** nota;  }sAlumno;  **void** ordenar(sAlumno\* arrayPunterosAlumnos[3], **int** longitudArray);  **int** main()  {  **int** i;  **int** auxiliarNota;  **char** auxiliarNombre[50];  sAlumno arrayAlumnos[CANTIDAD];  sAlumno\* arrayPunterosAlumnos[CANTIDAD];  **for**(i = 0; i < CANTIDAD; i++)  {  // COPIAMOS LA DIRECCION DE CADA ELEMENTO[i]  EN EL ARRAY DE PUNTEROS  arrayPunterosAlumnos[i] = &arrayAlumnos[i];  // CARGAMOS LOS DATOS EN CADA ELEMENTO DEL  ARRAY ESTRUCTURA  printf("Ingrese el nombre: ");  scanf("%s", auxiliarNombre);  strcpy(arrayAlumnos[i].nombre, auxiliarNombre);  //FALTARIA VALIDAR  printf("Ingrese la Nota: ");  scanf("%i", &auxiliarNota);  arrayAlumnos[i].nota = auxiliarNota;  //FALTA VALIDAR  }  printf("-MOSTRAMOS EL ARRAY SIN ORDENAR-\n");  **for**(i=0; i<CANTIDAD; i++)  {  printf("%s", arrayPunterosAlumnos[i]->nombre);  printf("-%i\n", arrayPunterosAlumnos[i]->nota);  }  ordenar(arrayPunterosAlumnos, CANTIDAD);  printf("-MOSTRAMOS EL ARRAY ORDENADO-\n");  **for**(i=0; i<CANTIDAD; i++)  {  printf("%s", arrayPunterosAlumnos[i]->nombre);  printf("-%i\n", arrayPunterosAlumnos[i]->nota);  }  **return** 0;  }  **void** ordenar(sAlumno\* arrayPunterosAlumnos[], **int** longitudArray)  {  sAlumno\* punteroAuxiliarAlumno;  **int** i;  **int** j;  **for**(i = 0; i < longitudArray -1 ; i++)  {  **for**(j = i+1; j < longitudArray ; j++)  {  **if**(arrayPunterosAlumnos[i]->nota > arrayPunterosAlumnos[j]->nota)  {  punteroAuxiliarAlumno = arrayPunterosAlumnos[i];  arrayPunterosAlumnos[i] = arrayPunterosAlumnos[j];  arrayPunterosAlumnos[j] = punteroAuxiliarAlumno;  }  }  }  } |

|  |
| --- |
| Observe que cuando se realiza el ordenamiento se ordena el vector de punteros, con lo cuál el vector de estructuras queda intacto.  Además de no modificarse el vector de estructuras, en el proceso de ordenar al modificar el vector de punteros siempre se mueve la misma cantidad de datos, es decir que no interesa la cantidad de campos que posea la estructura.  Entonces si una estructura contiene 20 campos y otra solo 2, en las asignaciones dentro del algoritmo que ordena solo se copiarán datos del tamaño del puntero. |

1. Ejercicios y videos

[VIDEO PARTE 1](https://youtu.be/sdo7miC8jSQ)

[VIDEO PARTE 2](https://youtu.be/pbjB6l4xMBQ)

[VIDEO PARTE 3](https://youtu.be/xujd4olvCK0)

[RESOLUCION EJERCICIO 1](https://youtu.be/qVW7UfAB7Dk?list=PLZU3OfSJlsrfKiKElmKxjCsyQQF_1ApuV)

[RESOLUCION EJERCICIO 2](https://youtu.be/pQbN5631cqg?list=PLZU3OfSJlsrfKiKElmKxjCsyQQF_1ApuV)

**Ejercicio 1:**

1) Escribir 4 funciones que reciban un array de números int y su tamaño, e impriman los números por pantalla.

* 1era funcion: Recibe el array como vector y accede a los datos utilizando notación vectorial ([]).
* 2da funcion: Recibe el array como vector y accede a los datos utilizando aritmetica de punteros.
* 3era funcion: Recibe el array como puntero y accede a los datos utilizando notación vectorial ([]).
* 4ta funcion: Recibe el array como puntero y accede a los datos utilizando aritmetica de punteros.

**Ejercicio 2:**

1. Realizar una función para cargar los datos de una persona en una struct, para ello se debe definir una estructura con los campos nombre, edad, dni.
2. La función tendrá el siguiente prototipo:

int loadPerson(struct S\_Person\* pPerson);

1. La función deberá devolver un 1 si los datos ingresados son correctos, y un 0 si no lo son.
2. Hacer otra función que reciba un puntero a la estructura e imprima por pantalla los datos, el prototipo es:

void printPerson(struct S\_Person\* pPerson);

1. Escribir un programa para probar las funciones.