Vectores y Punteros

Principal material bibliográfico utilizado

- www.jorgesanchez.net
- Fundamentos de Programación C/C++ - Ernesto Peñaloza Romero.
- Lenguaje C – Adolfo Beltramo, Nélida Matas.

Vectores

- También llamados arreglos (arrays).
- Es una colección de variables del mismo tipo, con el mismo nombre.
- Los elementos se diferencian mediante un subíndice.
- La forma general para declarar un arreglo es
  - Tipo nombre_de_variable[numero de elementos];
- Cuando el compilador se encuentra con dicha declaración reserva la cantidad de espacio de memoria para poder contenerlo de acuerdo a las posiciones que le indicamos. Los espacios de memoria para c/u de los elementos del vector, en C son reservados de forma contigua.
- En C el primer subíndice es el 0, así por ejemplo para un vector de diez posiciones el primer subíndice será 0 y el último el 9.

Vectores

- Los vectores contienen una serie finita de elementos, es decir que de antemano es necesario conocer la cantidad de elementos que poseen. El valor inicial de cada uno de los elementos del vector es indeterminado, poseerá lo que haya en la memoria, por lo tanto, es muy IMPORTANTE saber que hay que inicializarlos.

- IMPORTANTE: C no valida los limites de los arreglos, esta responsabilidad es del programador, por lo que el C nos permitirá acceder a cualquier elemento fuera de los limites pero seguramente generará en algún momento errores. Por esto es muy importante que el programador se encargue de efectuar todas las validaciones necesarias al momento de

- DEFINE: una buena practica es fijar los valores de los arrays con la clausula DEFINE.

- Para calcular el tamaño de un vector: sizeof (int) * tamaño del vector.

Vectores

```c
#define TAMANIO 20
int nota[TAMANIO];
/* Contador para recorrer el array */
int i;

for(i=0;i<TAMANIO;i++) {
  printf("Escriba la nota %d:",i);
  scanf("%d",&nota[i]);
}
```

```c
#include <stdio.h>
void main()
{
   int enteros [10]  ;
   int i;
   for (i=0; i < 10; i++)
     enteros [i] = i ;
}
```

Programa que guarda los números del 1 al 9 en un vector.

Vectores

Programa que guarda 20 números e informa su promedio.

```
#include <stdio.h>
void main()
{
 int valor [20] , i;
 float promedio;
 for (i=0; i < 20; i++)
  {
    printf ("Introducir un numero");
    scanf ("%d", &valor [i]  );
  }
  promedio = 0;
  for ( i=0; i < 20; i++ )
    promedio = promedio + valor [i];
  printf (" El promedio es %f\n ", (float) promedio/20);
}
```

Vectores

```
  Una tabla de acumulados */
#include <stdio.h>
void main(void)
{
   int iCantidad[12],  iContador;
   long int lSuma;
   printf("\n\t\t Acumulados mensuales");

   for (iContador=0; iContador <= 11 ; iContador++)
   {
     printf("\n Digite las ventas del mes %2d :",iContador+1);
     scanf("%d",&iCantidad[iContador]);
   }
   lSuma=0;
   printf("\n\n\t\t TABLA DE ACUMULADOS");
   printf("  \n\t\t ====================");
   printf("\n\n \t Mes          Ventas    Acumulados");
   for (iContador=0; iContador <= 11 ; iContador++)
   {
     lSuma=lSuma+iCantidad[iContador];
     printf("\n\t [%2d]  %10d %10d" ,iContador+1,
iCantidad[iContador], lSuma);

   }
   printf("\n");
}
```

Vectores

```
double factorial[21];
int i;

factorial[0]=1;
for(i=1;i<21;i++){
  factorial[i]=i*factorial[i-1];
}
/*Mostrar el resultado*/
for(i=0;i<21;i++){
  printf("El factorial de %d es .0f\n",i,factorial[i]);
}
```

Vectores y Cadenas

- Las cadenas habían sido manejadas hasta el momento como si fuesen variables nativas del lenguaje. Desde un punto de vista más correcto es que en realidad son vectores de caracteres.

- Es un array de caracteres que siempre terminará con el valor nulo (‘\0’) y cuya dimensión debe contemplarse al dimensionar el array.

- La diferencia principal es que los arreglos de caracteres son gestionados por funciones que tratan al arreglo como una sola entidad.

- Por ejemplo, una excepción a las reglas de vectores en C, es que las cadenas se pueden leer en una sola función scanf(). Esto solo es válido para las cadenas, ya que ningún otro tipo de arreglo es posible leerlo todo completo, sino que se tiene que leer elemento por elemento.

Vectores y Cadenas

- Algunas de las funciones principales que ya vimos para manejo de vectores de caracteres (cadenas) eran:
  - strcpy
  - strlen
  - strcat
  - strcmp

  En la bibliografía se mencionan muchas otras.
- Uso del terminador nulo: como todas las cadenas terminan en nulo, se pueden simplificar las operaciones.

Vectores y Cadenas

Ejemplo: Programa que pasa a mayúsculas una cadena.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main()
 {
     char cadena [80]  ;
     int x;
     strcpy (cadena, "pase a mayusculas" );
     for (x=0; cadena [x] ; x++ )
         cadena [x]  =  toupper ( cadena [x]  );
      printf ("%s\n", cadena );

 }
```

Se puede omitir el especificador de formato, utilizando: printf (cadena). Es lo mismo, pero sólo para cadenas.

Vectores y Cadenas

Programa para comprobar una password
(agregar el #include<stdio.h>)

```
#include <string,h>
#include <conio.h>
void main ()
{
  char clave [80] ;
  int x = 0:
   do
  {
        clrscr();
        if ( x != 0)
      {
          printf ( " Intente nuevamente\n");
          printf ("%d", x );
      }
        printf ("Ingrese palabra de acceso\n");
        gets (clave);
   }       while ( x==strcmp ("entrar", clave ));
```

Arreglos bidimensionales

- Un arreglo puede tener mas de una dimensión de forma tal que forme matrices de dos, tres o mas dimensiones.

- La forma de declarar un array de dos dimensiones es
  - Tipo nombre_del_arreglo[d1] [d2];
  - d1 es el subíndice para las filas.
  - d2 es el subíndice para las columnas.

	columna 0	columna 1	columna 2	columna 3	columna 4	columna 5
fila 0	4	5	6	8	9	3
fila 1	5	4	2	8	5	8
fila 2	6	3	5	7	8	9

- En vectores multidimensionales, es necesario manipular cada uno de los subíndices en forma separada, lo que hace necesario que se aniden estructuras de repetición para poder manejar cada subíndice

Arreglos bidimensionales

```
#include <stdio.h>
void main(void)
{
  int iMatriz1[5][5], iMatriz2[5][5],iSuma[5][5];
  int iRenglon, iColumna,iOrden1,iOrden2;

  printf("\n\t\t Suma de dos matrices");

  /* Las dos matrices deben ser conformables */
  printf("\nDigite el orden de la matriz [2-5] x [2-5] ");
  scanf("%d",&iOrden1);
  printf("%d x ",iOrden1);
  scanf("%d",&iOrden2);

  /* En C los indice comienzan con 0 por lo tanto se deben
  decrementar el orden */
  iOrden1--;
  iOrden2--;

  /* Captura de cada elemento de la matriz A */
  printf("\n\n Captura de la matriz A");

  for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
    for (iColumna=0; iColumna <=iOrden2; iColumna++)
    {
      printf("\n Digite el elemento (%2d,%2d) :",iRenglon+1,
          iColumna+1);
```

Arreglos bidimensionales

```
    scanf("%d",&iMatriz1[iRenglon][iColumna]);
  }

/* Captura de los elementos de la matriz B */
printf("\n\n Captura de la matriz B");

for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
  for (iColumna=0; iColumna <=iOrden2; iColumna++)
  {
    printf("\n Digite el elemento (%2d,%2d) :",iRenglon+1,
        iColumna+1);
    scanf("%d",&iMatriz2[iRenglon][iColumna]);
  }

printf(" \n RESULTADO:\n\n\n");

/* Suma de los elementos */
for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
  for (iColumna=0; iColumna <=iOrden2; iColumna++)
  {
    iSuma[iRenglon][iColumna]=iMatriz1[iRenglon][iColumna]+
       iMatriz2[iRenglon][iColumna];
  }
```

Arreglos bidimensionales

```
  /* Despliege de la matriz A */
  printf("\nMatriz A:\n");

  for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
  {
     for (iColumna=0; iColumna <= iOrden2; iColumna++)
     printf(" %10d ",iMatriz1[iRenglon][iColumna]);
     printf("\n");
  }

  /* Despliege de la matriz B */
  printf("\nMatriz B:\n");
  for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
  {
     for (iColumna=0; iColumna <= iOrden2; iColumna++)
     printf(" %10d ",iMatriz2[iRenglon][iColumna]);
     printf("\n");
  }

  /* Despliege de la matriz resultado */
  printf("\nMatriz Suma:\n");
  for (iRenglon=0; iRenglon <= iOrden1 ; iRenglon++)
  {
     for (iColumna=0; iColumna <= iOrden2; iColumna++)
     printf(" %10d ",iSuma[iRenglon][iColumna]);
     printf("\n");

  }
}
```

Arreglos bidimensionales

```
C:\> sum_mat
            Suma de dos matrices
Digite el orden de la matriz [2-5] x [2-5]

2
2 x 2

Captura de la matriz A
Digite el elemento (1, 1) : 10
Digite el elemento (1, 2) : 12
Digite el elemento (2, 1) : 13
Digite el elemento (2, 2) : 15

Captura de la matriz B
Digite el elemento (1, 1) : 20
Digite el elemento (1, 2) : 23
Digite el elemento (2, 1) : 22
Digite el elemento (2, 2) : 21

RESULTADO:

Matriz A:

        13          15

Matriz B:
        20          23
        22          21

Matriz Suma:
        30          35
        35          36
```

Arreglos de cadenas

- Se gestionan de forma diferente a como se gestionan los arreglos de números

- Si se quiere tratar un array de cadenas, es necesario recurrir a un array de 2 dimensiones.

  Por ejemplo:

  char cadenas[5][30]

  Esto representa una matriz de 5 cadenas, cada fila es una cadena, donde cada una puede contener hasta 29 caracteres. El acceso a la cadena se hace indicando el nombre del array y se indica el primer índice solamente.

Arreglos de cadenas

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
    char cad[5][11];
    int x,y;
    system("cls");
    for (x=0;x<5;x++)
    {
        printf("Ingrese cadena numero: %d\n",x+1);
        gets(cad[x]);
        }
    for (x=0;x<5;x++)
    printf(cad[x]);
    system("pause");
    }
```

Arreglos de cadenas

```
/* Archivo:Grupo.h
   Almacena el nombre de hasta 20 personas
 */
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main(void)
{
   char szNombre[20][50];
   char szCorreoE[20][25];
   int    iEdad[20];
   int    iNumElementos;
   int    iContador;

   printf("Agenda electronica no persistente\n Salir con <@>");

   /* Inicia las variables*/
   iNumElementos=0;
   szNombre[iNumElementos][0]='\0';
   while(szNombre[iNumElementos][0] != '@' && iNumElementos <20)
   {

      /*Asignacion de los datos de la persona*/
```

Arreglos de cadenas

```
    printf("\nNombre de la persona: ");
    flushall();
    gets (szNombre[iNumElementos]);
    if (strlen(szNombre[iNumElementos]) >0
      && szNombre[iNumElementos][0] != '@')
    {
      printf("\nDireccion de correo electronico: ");
      scanf("%s",szCorreoE[iNumElementos]);
      printf("\nEdad: ");
      scanf("%i",&iEdad[iNumElementos]);
      iNumElementos++;
    }

  }
  /* Listado de datos */
  printf("\n%-50s%-25s%-5s\n","Nombre","Direccion","Edad");
  for (iContador=0;iContador < iNumElementos;iContador++)
  {
    printf("%-50s%-25s%5i\n",szNombre[iContador],szCorreoE
        [iContador],iEdad[iContador]);
  }

}
```

Arreglos de cadenas

```
Agenda electronica no persistente
Salida con <@>
Nombre de la persona: Julio Verne

Dirección de correo electronico: jVerne@ficcion.fr

Edad: 30

Nombre de la persona : Isaac Asimov

Direccion de correo electronico: iAsimov@ficcion.com

Edad: 25

Nombre de la persona: @

Nombre                  Dirección               Edad

Julio Verne             jVerne@ficcion.fr         30

Isaac Asimov            iAsomov@ficcion.com       25
```

Inicialización de arrays

Forma general:

```
tipo nombre de variable [tamaño fila] [tamaño columnas]={  lista
                                                    de valores}
```

Ejemplos:

```
int enteros[10]  = { 1,2,3,4,5,6,7,8,9,10}  ;
char cadena[6]   = "hola";
char cadena [5]  = { 'h', 'o', 'l', 'a','\0'}  ;
int matriz [4] [2]   = { 1,1,2,4,3,9,4,16}  ;
```

Sin tamaño:

Por ejemplo, para inicializar una tabla de mensajes, que puede recibir cadenas de longitud variable:

```
char mensaje [3] [20]  = { "fuera de rango\n",
                           "entrada inválida\n",
                           "salida inválida\n" } ;
```

Asi aceptará 3 cadenas; si necesitamos una tabla de mensajes más amplia, será mucho más cómodo escribir:

```
char mensajes [][20] = idem .
```

Esto me permitirá ampliar la tabla sin ningún problema.
Sucede lo mismo con arrays de enteros:

```
int matriz [] [2]  = { 1,1,2,4,3,9,4,16 } ;
```

Esto permitirá alargar ó acortar la tabla, sin cambiar la dimensión del array.

Algoritmos de búsqueda y ordenación en vectores

- **ORDENAMIENTO**
  - SELECCIÓN
  - INSERCION
  - BURBUJEO
  - QUIKSORT
- **BUSQUEDA**
  - LINEAL
  - BINARIA

PUNTEROS

- Es una variable que contiene una dirección de memoria, apunta a otra variable
  - tipo *nombre de variable;
  - Por ejemplo:

```
char *p;
int *uno, *dos;
```

- Los operadores para manejo de punteros son el * y &.

- En cuanto a precedencia, los operadores * y & están por encima del resto de los operadores aritméticos

PUNTEROS

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int *puntero, valor, *direccion, valor1, valor2;
    valor=100;
    puntero = &valor;
    direccion = puntero;
    valor1=*puntero;
    printf("El valor ingresado es %d y su direccion %d\n",valor,puntero);
    printf("El valor ingresado es %d y su direccion %d\n",valor1,direccion);
    printf("El valor de la direccion en hexadecimal es %x %x\n",puntero,direccion);
    system("pause");
    }
```

PUNTEROS - Asignación

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int x, *punt1, *punt2;
    x=100;
    punt1 = &x;
    punt2 = punt1;
    printf("La direccion de x es %p\n",punt2);
    printf("El valor de x es %d\n",*punt2);
    system("pause");
    }
```

PUNTEROS - Aritmética

- Solo se pueden utilizar dos operadores aritméticos sobre punteros y estos son el + y el -. También es posible comparar dos punteros.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int x, *punt1;
    char *punt2, letra;
    letra='a';
    x=100;
    punt2 = &letra;
    punt1=&x;
    printf("Puntero a x es %p\n",punt1);
    printf("tamaño de entero %d bytes \n",sizeof(int));
    printf("Puntero a siguiente es %p\n",++punt1);
    printf("Puntero a letra es %p\n",punt2);
   printf("tamaño de char %d bytes \n",sizeof(char));
    printf("Puntero a siguiente es %p\n",++punt2);
    system("pause");
    }
```

PUNTEROS y ARRAYS

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
    char *cad, cadena[5]="hola"; int x;
    system("cls");
    printf(cadena);
    printf("\n");
    cad=cadena; /* equivale a:cad=&cadena[0]*/
    printf(cad);
    printf("\n");
    /* Para imprimir de a una letra por renglon*/
    for(x=0;cadena[x];x++) printf("%c\n",cadena[x]);
    /* ó*/
    for(;*cad;cad++) printf("%c\n",*cad);
    /* ó*/
    cad=cadena;
    for(x=0;cad[x];x++) printf("%c\n",cad[x]);
    system("pause");
    }
```

PUNTEROS y ARRAYS

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main()
{
    int *ent, enteros[5]={1,2,3,4,5}, x;
    ent=enteros; /*lo mismo es ent=&enteros[0]
    3 formas de imprimir lo mismo*/

    for(x=0;x<5;x++) printf("%d\n",enteros[x]);
    /* ó*/
    for(x=0;x<5;x++) printf("%d\n",ent[x]);
    /* ó*/
    for(x=0;x<5;x++)
    {
    printf("%p\n",ent+x);
    printf("%d\n",*(ent+x));
                              }

    system("pause");
    }
```

PUNTEROS y Cadenas

- Como ya se menciono, una cadena es un array de caracteres, por lo tanto, su nombre es un puntero al primer elemento del vector.

Ejemplo de cómo funciona strlen().

```
Int strlen(char *cad)
{
Int x=0;
while (*cad) /*recordar que no cuenta el nulo*/
{
        x++;
        cad++;
                }

return x;
}
```

EJERCICIO: Ingresar una cadena con un espacio en blanco, y luego imprimir la misma desde el espacio en blanco.

Pasar vectores como parámetros de funciones

- Cuando vimos funciones, vimos que significaba pasar por valor o por referencia.
- Cuando se pasa por valor, se toma en la función una copia del dato, cuando se pasa por referencia se toma la dirección de memoria donde esta almacenado el dato.
- En variables simples, la dirección de memoria del dato se indica mediante &.
- En el caso de los arrays, hay que considerar que la dirección donde comienza el array es la dirección de su primer elemento. Por ejemplo para el array nota, el vector comienza en la direccion &nota[0], y en realidad la variable que hace referencia a todo el array (nota en este caso), apunta a la dirección de comienzo del array. Por lo tanto nota y &nota[0], se refieren a la dirección de memoria en la que se almacena el primer elemento del array..

Pasar vectores como parámetros de funciones

```
int nota[8]={4,6,7,6,8,9,8,4};
printf("%d\n",&nota[0]); /* Escribe una dirección de memoria */
printf("%d\n",nota); /* Escribe la misma dirección */
printf("%d\n",&nota); /* De nuevo escribe la misma dirección */
```

- Cuando se recibe un array como parámetro de una función, lo que esta recibe es una referencia al primer elemento.

```
double media(int numero[], int tamanio){
  double res;
  int i;
  for(i=0;i<tamanio;i++){
        res+=numero[i]; /* Se acumula en res la suma
                           total del array */
  }
  return res/tamanio; /* res/tamanio es la media */
}
```

Pasar vectores como parámetros de funciones

- Hay que tener en cuenta, que al pasarse por referencia el vector, cuando se efectúan cambios en la función, cambia el valor original del vector pasado, por que al ser una dirección de memoria, pasó por referencia.

```
#include <stdio.h>
void prueba(int[] x);
int main(){
   int a[4];
   a[0]=12;
   prueba(a); /*Se llama a la función prueba con el valor
a */
   printf("%d",a[0]); /* escribirá el valor 3 */
}

void prueba(int[] x){
   x[0]=3;
}
```

Pasar vectores como parámetros de funciones

- En los ejemplos vistos, pasamos los vectores sin dimensión pero podrían haber sido pasados con dimensión.
- Otra forma de pasar vectores a las funciones es a través de punteros (la mas usada).

```
#include <stdio.h>
#include <stdlib.h>
void mostrar (int * ent);/**/
void main()
{
    int enteros[5],x;
    for(x=0;x<5;x++)
                enteros[x]=x;
                mostrar (enteros);

    system("pause");
    }

void mostrar (int * ent)
{
    int y;
    for (y=0;y<5;y++)
        printf("%d ",ent[y]);

        printf("\n\n");
        /*oooooooo*/

    for (y=0;y<5;y++)
        printf("%d ",*ent++);
}
```

EJERCICIO: Ingresar una cadena en minúscula, pasarla como puntero a una función, dentro de la función cambiarla a mayúsculas y mostrarla (dentro de la función y fuera de la función)