Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Ramón Mejía Martínez

Matrícula: 372099

Maestro: Pedro Núñez Yépiz

Actividad No. : 8

Tema - Unidad : Arreglos y Funciones

Ensenada Baja California a 2 de octubre del 2023

1. Introducción

En esta actividad se abordarán los arreglos unidimensionales y bidimensionales.

2.Competencia

Manejo de arreglos, lectura, asignación, concatenación de arreglos.

3. Fundamentos

Los Arreglos son un conjunto de espacios de memoria consecutivos que guardan el mismo tipo de dato y son accesibles por el identificador y un índice que indica la posición en el arreglo.

La declaración en C de un arreglo es:

```
tipo nombre_arreglo[dimensión]
```

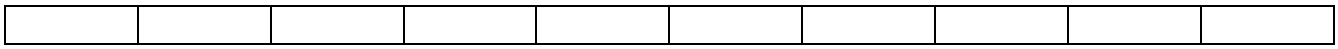
La siguiente declaración declara un array de 10 enteros

```
int a [10];
```

Los elementos se acceden como a[0], a[1] . . . a[9]

Los elementos de un array se almacenan consecutivamente en memoria

a: a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]



Para inicializar el arreglo en la declaración se hace de la siguiente mantera:

```
Int a[10]={1,2,3,4,5,6,7,8,9,10}
```

Y se guardan en orden desde la posición 0 a 9.

Otra forma de asignar es asignarlas uno por uno.

```
a[0]=1;
a[1]=2; . . .
```

Para manejarlos se pueden usar estructuras de control repetitivos, por ejemplo, para mostrar el contenido del vector:

```
for(i=0;i<10;i++)
{
        printf(" %d",a[i]);
}
```

Para hacer operaciones se pueden usar como cualquier variable, indicando el índice.

```
Int x;
x=a[0]*a[1];
```

x contendría 2 ya que estamos multiplicando 1*2.

4. Procedimiento

INSTRUCCIONES

1.- Realiza un programa en C
(Funciones e Introduccion a Arreglos en C)
2.- Realiza reporte de práctica y anexos (Teoria Arreglos y Funciones)
3.- Sube a Blackboard: Programa , Reporte de practica y anexo con capturas y código (3 Archivos 1 cpp, 2 PDF)
4.- Sube a GitHub en tu repositorio los 3 documentos y poner enlace en BlackBoard

NOTA: No se te olvide poner portada en los documentos e informacion en el programa, recuerda que tus conclusion es muy importante y sobre todo saber si se cumple el objetivo del tema

ACTIVIDAD 8

Realiza programa en C el programa deberá tener el siguiente menú.

MENÚ
1.- LLENAR VECTOR 1 (MANUALMENTE)
2.- LLENAR VECTOR 2 ALEATORIAMENTE
3.- LLENAR VECTOR 3 (CON VECTOR1 Y VECTOR2)
4.- IMPRIMIR VECTORES
5.- LLENA MATRIZ 4 X 4
6.- IMPRIMIR MATRIZ
0.- SALIR

NOTA: EL PROGRAMA DEBERÁ REPETIRSE CUANTAS VECES LO DESEE EL USUARIO
NOTA 2: EL **VECTOR 1** DE **10** POSICIONES, NÚMEROS DEL **30 AL 70**
NOTA 3: EL **VECTOR 2** DE **10** POSICIONES CON NÚMEROS GENERADOS ALEATORIAMENTE DEL **1 AL 20 (SIN REPETIR)**
NOTA 4: EL **VECTOR 3** DE **20** POSICIONES, CON LOS DATOS DEL ARREGLO1 Y ARREGLO2
NOTA 5: MATRIZ 4 X 4 LLENARLA CON LOS DATOS DEL VECTOR1 Y VECTOR2,

5. Resultados y conclusiones.

Constantes y prototipos.

```
#define STR 30      //Longitud de cadena pra validar
#define M 10        //Dimension de vectores 1,2
#define MM 20       //Dimension de vector 3
#define NM 4        //Dimension de matriz cuadrada

int validarlim(char txt[],int limi,int lims);
int msgs(void);
void menu(void);
int llenaVector(int vect1[]);
int randVector(int vect2[]);
void printVector(int vect[],int m);
int unirVector1_2(int vect1[],int vect2[],int vect3[],int v1,int v2);
int llenaMatriz(int vect1[],int vect2[],int matriz[][NM],int v1,int v2);
void printMatriz(int matriz[][NM],int m1);
```

El main solo contiene la llamada a la función menú.

```
int main()
{
    menu();
    return 0;
}
```

Función validarlim:

- Recibe el texto a mostrar cuando se leerá el valor.
- Recibe el linte inferior y superior para validar.
- Muestra mensaje, lee el valor como cadena.
- Convierte la cadena a entero.
- Si el valor no es valido muestra mensaje de error y vuelve a pedirlo.
- Regresa el valor valido.

```
int validarlim(char txt[],int limi,int lims)
{

    char r[STR];
    int rint;

    do
    {
        printf("%s",txt);
        fflush(stdin);
        gets(r);
        rint=atoi(r);
        if(rint<limi || rint>lims)
        {
            printf("Valor invalido, ingresalo de nuevo\n");
        }
    }while(rint<limi || rint>lims);

    return rint;
}
```

Función msgs:

- Muestra el menú.
- Lee y regresa la opción del usuario.

```
int msgs(void)
{
    printf(" Menu\n");
    printf("1) Llenar vector 1 manual\n");
    printf("2) Lenar vector 2 aleatorio\n");
    printf("3) Unir vector 1 y 2\n");
    printf("4) Imprimir vectores \n");
    printf("5) Llenar matriz 4x4\n");
    printf("6) Imprimir matriz\n");
    printf("7) Salir\n");
    return (validarlim("Escoge una opcion: ",1,7));
}
```

Función menu:

- Llama a la función msgs y guarda la opción, con un switch hace lo correspondiente a la opción.
- Caso 1:
  - Manda a llamar la función llenaVector.
  - Guarda en la bandera v1 que el vector 1 está lleno.

```
void menu(void)
{
    int op,vect1[M],vect2[M],vect2_2[M],vect3[M],matriz[NM][NM],v1=0,v2=0,v3=0,m1=0;
    do
    {
        system("cls");
        op=msgs();
        switch(op)
        {
            case 1:
                system("cls");
                printf("\nLlenar el vector 1\n");
                printf("\nValores del 30 al 70\n");
                v1=llenaVector(vect1);
                //printVector(vect1,M);
                break;
```

- Caso 2:
  - Manda a llamar la función randVector.
  - Guarda en la bandera v2 que el vector 2 está lleno.
- Caso 3:
  - Manda a llamar la función unirVector1_2Vector.
  - Guarda en la bandera v3 que el vector 3 está lleno.

```
case 2:
    v2=randVector(vect2);
    //printVector(vect2,M);
    break;
case 3:
    system("cls");
    printf("\nUnir vector 1 y 2\n");
    v3=unirVector1_2(vect1,vect2,vect3,v1,v2);
    break;
```

- Caso 4:
  - Comprueba la bandera de los vectores.
  - Si la bandera es 1 imprime el vector con la función printVector.
  - Repite para los 3 vectores.

```
case 4:
    system("cls");
    printf("\nMostrar vectores\n");

    if(v1==1)
    {
        printf("\nVector 1\n");
        printVector(vect1,M);
    }
    else
    {
        printf("\nEl vector 1 esta vacio\n");
    }
    if(v2==1)
    {
        printf("\nVector 2\n");
        printVector(vect2,M);
```

...

- Caso 5:
  - Llena la matriz con el vector 1 y 2 llamando a la función llenaMatriz.
  - Guarda en la bandera m1 que la matriz esta llena.
- Caso 6:
  - Imprime la matriz llamando a la función printMatriz.
- Caso 7:
  - Muestra mensaje de que saldrá saliendo del programa.

```
        case 5:
            m1=llenaMatriz(vect1,vect2,matriz,v1,v2);
            break;
        case 6:
            system("cls");
            printf("Mostrar matriz\n\n");
            printMatriz(matriz,m1);
            break;
        case 7:
            printf("\nSaliendo del programa\n");
            break;
    }
    printf("\n");
    system("pause");
}while(op!=7);
```

Función llenaVector:

- Con un ciclo for que va de 0 a 9 lee y valida lo que el usuario ingrese al vector 1.
- No manda ningún mensaje porque la función validarlim no puede recibir la i;
- Al leer el vector manda los limites 30 y 70 ya que el vector se debe llenar con valores dentro de ese rango.
- Regresa 1, indica que el vector esta lleno.

```
int llenaVector(int vect1[])
{
    int i;
    for(i=0;i<M;i++)
    {
        printf("Ingresa el valor de la posicion: %d\n",i+1);
        vect1[i]=validarlim("",30,70);
    }
    return 1;
}
```

Función randVector:

- Usa la función srand para poder generar números aleatorios.
- Con un ciclo for que va de 0 a 9 se le asigna un numero aleatorio a todos los espacios del vector 2.
- Regresa 1, indica que el vector 2 esta lleno.

```
int randVector(int vect2[])
{
    int i;
    srand(time(NULL));
    for(i=0;i<M;i++)
    {
        vect2[i]=(rand()%20)+1;
    }
    printf("\nSe lleno el vector 2\n");
    return 1;
}
```

Función printVector:

- Recibe el vector a imprimir y la longitud.
- Con un ciclo for que va desde 0 hasta longitud – 1, imprime todas las posiciones del vector.

```
void printVector(int vect[],int m)
{
    int i;
    for(i=0;i<m;i++)
    {
        printf(" %d",vect[i]);
    }
}
```

Función unirVector1_2:

- Verifica que el vector 1 y 2 estén llenos, si no están llenos no muestra un mensaje de que están vacíos y regresa 0 indicando que no se llenó el vector.
- Si el vector 1 y 2 están llenos asigna los valores del vector 1 a las primeras 10 posiciones del vector 3, y los valores del vector 2 a las ultimas 10 posiciones del vector 3.

```
int unirVector1_2(int vect1[],int vect2[],int vect3[],int v1,int v2)
{
    int i;
    if(v1==1)
    {
        if(v2==1)
        {
            for(i=0;i<M;i++)
            {
                vect3[i]=vect1[i];
                vect3[i+M]=vect2[i];
            }
            printf("\nSe unieron los vectores\n");
            return 1;
        }
        else
        {
            printf("\nEl vector 2 esta vacio\n");
            return 0;
        }
    }
    else
    {
        printf("\nEl vector 1 esta vacio\n");
        return 0;
    }
}
```

Función llenaMatriz:

- Verifica si el vector 1 y 2 están llenos, si no lo están muestra un mensaje de eso y regresa 0, lo que indica que no se llenó la matriz.
- Si el vector 1 y 2 están llenos, llena la matriz con todos los valores del vector 1, ya que la matriz es de 16 espacios, llena los últimos 6 espacios con los primeros valores del vector 2.

```
int llenaMatriz(int vect1[],int vect2[],int matriz[][NM],int v1,int v2)
{
    int j;
    if(v1==1)
    {
        if(v2==1)
        {
            for(j=0;j<NM;j++)
            {
                matriz[0][j]=vect1[j];
                matriz[1][j]=vect1[j+NM];
                if(j<2)
                {
                    matriz[2][j]=vect1[j+NM*2];
                }
                else
                {
                    matriz[2][j]=vect2[j-2];
                }
                matriz[3][j]=vect2[j+2];
            }
            printf("\nLa matriz 4x4 se lleno\n");
            return 1;
        }
        else
        {
            printf("\nEl vector 2 esta vacio\n");
            return 0;
```

Función printMatriz:

- Verifica si la matriz esta llena, si no lo esta muestra un mensaje de eso.
- Si esta llena la imprime usando dos ciclos for, el primero usa i que controla los renglones, el segundo usa j que contarla las columnas.

```
void printMatriz(int matriz[][NM],int m1)
{
    int i,j;
    if(m1==1)
    {
        for(i=0;i<NM;i++)
        {
            for(j=0;j<NM;j++)
            {
                printf(" %4d",matriz[i][j]);
            }
            printf("\n");
        }
    }
    else
    {
        printf("\nLa matriz esta vacia\n");
    }
}
```

Conclusión:

Manejar los arreglos es muy similar a como trabajamos con las cadenas, las banderas fueron útiles para validar que los arreglos estén llenos o no.

Un problema que hubo fue que la memoria al trabajar con ellos puede fallar, declare la variable vect2_2 que no es usada para solucionar un problema.

```
void menu(void)
{
    int op,vect1[M],vect2[M],vect2_2[M],vect3[M],matriz[NM][NM],v1=0,v2=0,v3=0,m1=0;
    do
    {
```

Sin esta variable declarada al imprimir el vector 2 se imprime mal.

```
Mostrar vectores

Vector 1
 30 30 30 30 30 30 30 30 30 30
Vector 2
 1 9 11 9 3 13 12 10 12 10
Vector 3
 30 30 30 30 30 30 30 30 30 30 20 6 1 9 11 9 3 13 12 10
Presione una tecla para continuar . . .
```

El vector 3 contiene los valores correctos del vector 2 pero al imprimir el vector 2 se recorre dos posiciones y repite los ultimos 2 valores.

Esto solo ocurre al imprimirlo dentro del switch, fuera de él se imprime correctamente (igual a la segunda parte del vector 3), además que al actualizar el vector 2 sin haber vuelto a unir los vectores se actualizaba el vector 3.

Con la variable declarada se muestra correctamente y ya no se actualiza solo:

```
Mostrar vectores

Vector 1
 30 30 30 30 30 30 30 30 30 30
Vector 2
 3 20 14 17 9 12 15 3 5 13
Vector 3
 30 30 30 30 30 30 30 30 30 30 3 20 14 17 9 12 15 3 5 13
Presione una tecla para continuar . . .
```

6. Anexos

PDF:

https://drive.google.com/file/d/1eMBurEDQzflM7Njv_pfg4JNoQke3nMfL/view?usp=sharing

Repositorio:

https://github.com/Muners24/Programacion-Estructurada

7.REFERENCIAS

Diseño de algoritmos y su codificación en lenguaje C

Corona, M.A. y Ancona, M.A. (2011)..

España: McGraw-Hill.

ISBN: 9786071505712

Programación estructurada a fondo:implementación de algoritmos en C

:Pearson Educación.Sznajdleder, P. A. (2017)..

Buenos Aires,Argentina: Alfaomega

Como programar en C/C++

H.M. Deitel/ P.J. Deitel

Segunda edición

Editorial: Prentice Hall.

ISBN:9688804711

Programación en C.Metodología, estructura de datos y objetos

Joyanes, L. y Zahonero, I. (2001)..

España:McGraw-Hill.

ISBN: 8448130138

Operativos, S. (s/f). El Lenguaje de programación C. Udc.es. Recuperado el 2 de octubre de 2023, de https://www.dc.fi.udc.es/~so-grado/current/Varios/CursoC.pdf