3. ARREGLOS Y ESTRUCTURAS

3.1. ARREGLOS UNIDIMENSIONALES

En la práctica de la programación suele ser necesario procesar un conjunto de datos, como podrían ser listas de clientes, cuentas bancarias, expedientes de alumnos, etc., y no sólo datos simples como números enteros o caracteres. Por ello es necesario que el lenguaje de programación, en este caso, el lenguaje C proporcione soporte para mantener y procesar conjuntos de datos que representan un todo y no tan solo variables simples. Éste soporte está dado a través de los llamados arreglos los cuales se dividen en arreglos lineales, unidimensionales o vectores y arreglos multidimensionales o de dimensión-n con n mayor o igual a 2.

Un arreglo lineal o vector es una estructura de datos en donde podemos almacenar un conjunto de valores o datos y no necesariamente uno sólo, todos del mismo tipo, identificados mediante un índice representado con un valor entero. Todo arreglo en principio es finito y tiene un tamaño, es decir, tiene un límite de almacenamiento. Si un arreglo almacenará datos de tipo entero entonces tendrá que ser declarado y definido como un arreglo de enteros (es decir, una estructura que almacenará elementos que serán números enteros). Si por el contrario el arreglo almacenará caracteres, entonces tendrá que ser definido como un arreglo de caracteres, etc.
No podemos tener, por tanto, un arreglo que almacene en algunas posiciones de él números enteros y en otros caracteres porque tendremos un problema de incompatibilidad de tipos pues el arreglo es declarado como de tipo entero o de tipo carácter pero no de ambos.

Para efectos de entendimiento, los arreglos se representan gráficamente como una serie de casillas contigüas numeradas con un índice que inicia desde el número cero y termina en el tamaño del arreglo menos uno, donde dentro de ellas se encuentran los datos a almacenados (todos del mismo tipo). Por ejemplo:

letras	‘B’	‘V’	‘H’	‘I’	‘4’	‘2’	‘t’
	0	1	2	3	4	5	6

En la figura se muestra un arreglo de caracteres llamado “letras” de tamaño 7 (casillas) en donde la primera casilla tiene almacenado el carácter ‘B’ con índice 0 y la última casilla tiene almacenado el carácter ‘t’ con índice 6.

Un arreglo posee entonces las características siguientes:

- **Es una estructura homogénea:** Es decir, todos los elementos almacenados en el arreglo son del mismo tipo de datos.
- **Es una estructura lineal de acceso directo:** Significa que podemos acceder a los datos almacenados en el arreglo de manera directa a través de su posición mediante el índice.

- letras[3] significa que estamos accesando a la casilla referenciada por el índice 3 del arreglo llamado "letras". En otras palabras, letras[3] significa en el ejemplo que estamos accesando de manera directa al carácter 'l'.
- **Es una estructura estática:** El arreglo es finito, es decir, tiene un tamaño y es de un determinado tipo y se define en tiempo de compilación. Una vez creado el arreglo éste no cambia de tamaño durante la ejecución del programa debido a que en el momento de compilar el programa, el compilador reserva antes de la ejecución del mismo las casillas necesarias de memoria que el arreglo requiere para almacenar los datos.

3.1.1. Declaración y uso de arreglos lineales

En C, un arreglo unidimensional se declara indicando el tipo de dato del mismo, es decir, que tipo de elementos almacenará y el tamaño de éste bajo la siguiente sintaxis:

```
<tipo_de_dato_base> nombreArreglo [<tamaño>];
```

Por ejemplo: declaración de un arreglo de valores de tipo doublé de 10 casillas:

```
double sueldos_maestros[10];
```

Una vez declarado el arreglo podemos acceder a sus casillas o elementos tanto para escritura a él como para lectura de él a través de un índice:

```
double sueldos_maestros[10];
double salario= 1500.00;

/* Escritura al arreglo de la variable salario
   y del valor 126.34
*/
sueldos_maestros[4]=salario;
sueldos_maestros[0]=126.34;

/* lectura del elemento alamacenado en la cuarta posición del
   arreglo
*/
double valor= sueldos_maestros[4];
```

Cuando se trabaja con arreglos y en general para cualquier declaración de tipos, es recomendable declarar lo que se conoce como **"un tipo de dato de usuario"** mediante el uso de la palabra reservada *typedef*:

```
typedef double t_miArreglo[10];
t_miArreglo sueldos_maestros;
```

En la práctica, los arreglos se trabajan mediante ciclos, lo cual facilita el procesamiento de los datos almacenados en él, especialmente cuando se aplica una misma operación a todos los elementos del arreglo. El siguiente programa inicializa un arreglo con valores enteros generados de manera aleatoria y a continuación se buscan los valores máximo y mínimo en él.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define TAM 10

int main()
   {
    int enteros[TAM];
    int max, min, i;

    srand(time(NULL));
    for(i=0;i<TAM;i++)
      {
        enteros[i]=rand();
        printf("enteros[%d]= %d\n",i,enteros[i]);
      }

    min=enteros[0];
    max=enteros[0];

    for(i=1;i<TAM;i++)
      {
       if(enteros[i]>max)
           max= enteros[i];

       if(enteros[i]<min)
           min=enteros[i];
      }

    printf("Valor maximo = %d\n",max);
    printf("Valor minimo = %d\n",min);

    return 0;
   }
```

En este ejemplo distinguimos la parte de inicialización del arreglo y la parte del procesamiento de él. La inicialización se lleva a cabo con números generados de manera

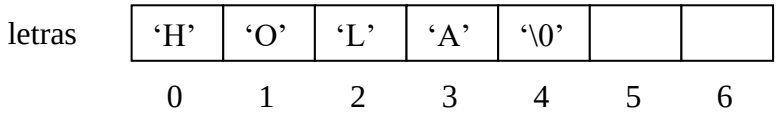
aleatoria, mediante el uso de las funciones *srand() y rand()*. La primera permite inicializar la serie de números pseudo-aleatorios a partir de una "semilla" recibida como parámetro (en este caso, se utiliza la hora del sistema como tal). Luego, cada posición del vector se inicializa con un número entero pseudo aleatorio generado con llamadas sucesivas a *rand()*.

3.2. MANEJO DE CADENAS

Para trabajar con cadenas desde un programa en C, éste proporciona como recurso los arreglos de caracteres. Cuando hacemos la siguiente declaración:

```
char cadena[30];
```

entendemos que estamos declarando un arreglo unidimensional de caracteres de 30 bytes (30 casillas X 1 byte), siendo 1 byte el tamaño de cada caracter. El arreglo alojará entonces 29 caracteres, más el carácter especial **'\0'** que representa el **"fin de la cadena"**. Por ejemplo: El siguiente es un arreglo de caracteres de 7 casillas que tiene almacenada la cadena "hola" constituida de 4 caracteres más el carácter de fin de cadena.



En C entonces, una cadena es un arreglo de caracteres sin particularidad alguna salvo la existencia del carácter '\0', que permite aplicar sobre estos arreglos un conjunto de funciones para manipulación de cadenas, que se encuentran en el archivo de encabezado: *string.h*. Algunas funciones se listan a continuación:

FUNCIÓN	DESCRIPCIÓN
`char* strcpy(char *s1, char *s2)`	Copia s2 en s1 y retorna s1
`char* strncpy(char *s1, char *s2, int n)`	Copia hasta n caracteres de s2 en s1 y retorna s1
`char* strcat(char *s1, char *s2)`	Concatena s2 a s1 y regresa s1
`char* strncat(char *s1, char *s2, int n)`	Concatena hasta n caracteres de s2 a s1 y regresa s1
`int strcmp(char *s1, char *s2)`	Compara s1 y s2. Regresa 0 si son iguales, un entero negative si s1 es menor o un entero positive si s1 es mayor
`char* strncmp(char *s1, char *s2, int n)`	Lo mismo que la anterior pero sólo se consideran los primeros n caracteres de ambas cadenas
`char* strchr(char *s,`	Busca el character c en s y regresa un apuntador a él si

int c)	se encuentra, o NULL en caso contrario
Char* strstr(char *s1, char *s2)	Busca la cadena s2 en s1 y regresa un apuntador a ella si se encuentra, o NULL en caso contrario
Int strlen(char *s)	Regresa la longitud (cantidad de caracteres) del arreglo s sin contar '\0'

Para que estas funciones e apliquen correctamente, es importante delimitar el fin de una cadena. De otra forma, los resultados serían inesperados.

```
char palabra[]={'H','o','l','a','\0'};
printf("%d\n",strlen(palabra)); /*devuelve 4*/
```

Como una cadena es un arreglo de caracteres, nada impide que la trabajemos como tal:

```
palabra[0]='H';
palabra[1]='o'
```

El siguiente ejemplo muestra el uso de cadenas:

```
#include <stdio.h>
#include <string.h>

int main()
   {
    char cadena1[50]={'H','o','l','a','\0'};
    char cadena2[50]={'M','U','N','D','O','\0'};
    int valor;

    printf("La cadena %s tiene %d caracteres...\n",
            cadena1,strlen(cadena1));
    printf("La cadena %s tiene %d caracteres...\n",
            cadena2,strlen(cadena2));
    strcat(cadena1,cadena2);
    printf("La cadena %s tiene %d caracteres...\n",
            cadena1,strlen(cadena1));

    valor=strcmp(cadena1,cadena2);

    if (valor==0)
       printf("%d: Las cadenas %s y %s son iguales...\n",
               valor,cadena1,cadena2);
    else  if (valor>0)
                  printf("%d: Las cadena %s es mayor que la cadena
                          %s...\n",valor,cadena1,cadena2);
          else printf("%d Las cadena %s es menor que la cadena
```

```
                        %s...\n",valor,cadena1,cadena2);

    system("pause");
    return 1;
   }
```

3.2.1. Funciones estándares para el manejo de caracteres y cadenas de caracteres

Existen en C varias bibliotecas estándares que proponen funciones para el trabajo con caracteres y cadenas.

Biblioteca **<ctype.h>:** En ella existen funciones que prueban la naturaleza de los caracteres ASCII, ya sea que se trate de ua cifra, letra, signo de puntuación, carácter de control, etc. La siguiente tabla muestra las principales funciones de ésta biblioteca:

`int isupper(char c)`	c es una letra mayúscula
`int islower(char c)`	c es una letra minúscula
`int isalpha(char c)`	Isupper(c) o islower(c)
`int isdigit(char c)`	c es un dígito
`int isalnum(char c)`	c es alfanumérico
`int isprint(char c)`	c es un carácter imprimible
`int iscontrol(char c)`	c es carácter de control
`int ispunct(char c)`	c es carácter de puntuación

Éstas funciones regresan el valor 0 si la condición es falsa, o un valor distinto de 0 si la condición es cierta.

Biblioteca **<stdlib.h>**: En esta biblioteca existen tres funciones que nos ayudan a convertir una cadena de caracteres en un valor numérico. Éstas son:

`double atof(char *s)`	Convierte la cadena s en un flotante
`int atoi(char *s)`	Convierte la cadena s en un entero
`long atol(char *s)`	Convierte la cadena s en un entero long

Si la conversión es posible en ellas, éstas regresan el valor del tipo indicado; sino las funciones regresan el valor de 0.

3.2.2. Lecturas y escrituras de y a cadenas de caracteres

Las lecturas/escrituras pueden hacerse con *printf y scanf* (de la biblioteca <stdio.h>) respetando el siguiente formato:

- %c para los caracteres
- %s para las cadenas de caracteres

La primera lectura de un carácter toma en cuenta el primer carácter proorcionado y la segunda lectura puede iniciar con el próximo carácter del flujo. Por eso la lectura de una cadena de caracteres con *scanf* siempre toma en cuenta la sucesión de caracteres hasta el primer separador (que puede ser un espacio, una línea nueva o una tabulación). La función *scanf* también coloca el carácter '\0' o NULL al final de la cadena. El código siguiente muestra un ejemplo de esto que se esta explicando:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
  {
   char car;
   char cadena[10];

   printf("Lectura de un caracter y de una cadena\n");
   scanf("%c%s",&car,cadena);

   printf("caracter= %c\n",car);
   printf("cadena= %s\n",cadena);
   system("pause");
   exit(0);
  }
```

Las salidas de este programa dependen de las entradas del mismo. Entonces:

```
Lectura de un caracter y de una cadena
3XYZ
caracter= 3
cadena= XYZ


Lectura de un caracter y de una cadena
4     RTRTR
caracter= 4
cadena= RTRTR
```

Lectura de un caracter y de una cadena
345 67 8989898
caracter= 3
cadena= 45

Las funciones *printf y scanf* trabajan con flujos estándares de salida y de entrada respectivamente. En **<stdio.h>** también existen dos funciones parecidas: **sprintf y sscanf**, en donde la escritura y lectura no se hacen en flujo sino en una cadena de caracteres. La sintaxis de dichas instrucciones es la siguiente:

```
int sscanf(char *s, char *formato, elementos…);
int sprintf(char *s, char *formato, elementos…);
```

por su parte, en **<stdio.h>** hay otras funciones de lectura/escritura de caracteres y cadenas:

`char getchar()`	Lee y regresa un carácter del flujo estándar de entrada
`char putchar(char c)`	Que escribe el carácter c en el flujo estándar de salida
`char *gets(char *s, int n)`	Que lee una cadena de caracteres s, desde el flujo de entrada hasta que se encuentra una línea nueva o que se hayan leido hasta n caracteres. Al final de la cadena se coloca el carácter '\0'
`char puts(char *s)`	Que escribe la cadena s y el carácter de fin de línea '\n'

El siguiente ejemplo funciona igual que el anterior, sólo que las entradas se hacen con otras funciones:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
  {
   char car;
   char cadena[10];

   printf("Lectura de un caracter y de una cadena\n");
   car=getchar();
   gets(cadena);

   printf("\n");
   putchar(car);
   printf("\n");
```

```
   puts(cadena);

   printf("\ncaracter= %c\n",car);
   printf("cadena= %s\n",cadena);
   system("pause");
   exit(0);
  }
```

Una salida del programa es:

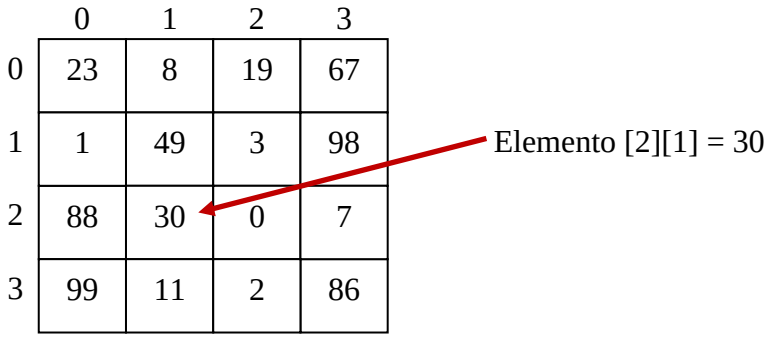
Lectura de un caracter y de una cadena
345 67 8989898

3
45 67 8989898

caracter= 3
cadena= 45 67 8989898

3.3. ARREGLOS BIDIMENSIONALES O MATRICES

El concepto de arreglo unidimensional se puede extender agregando más dimensiones para dar lugar a los arreglos multidimensionales. La implementación de éstos y su operación no cambia, salvo porque para acceder a una posición específica, se deberán utilizar tantos índices como dimensiones se tengan en el arreglo. Los más útiles son los arreglos bidimensionales o matrices, que son arreglos de dos dimensiones:



El siguiente fragmento de código inicializa con ceros un arreglo bidimensional de enteros de tamaño MxN

```
int enteros[M][N];
int i,j;
for(i=0;i<M;i++)
   {
     for(j=0;j<N;j++)
        {
          Enteros[i][j]=0;
         }
   }
```

Es interesante comentar que los arreglos multidimensionales se almacenan en memoria principal por líneas o por columnas, dependiendo del lenguaje de programación que se utilice. En el caso de C, éste almacena en memoria los arreglos multidimensionales por filas. El siguiente código muestra un programa que pide valores enteros a un usuario y los va almacenando en una matriz:

```
#include <stdio.h>
#define FILAS 3
#define COLUMNAS 3

int main()
   {
    int i,j;
    int matriz[FILAS][COLUMNAS];

    printf("Llenado de una matriz de %dx%d con valores
            enteros\n",FILAS,COLUMNAS);
    printf("_____________________________________________________\n\n");

    for(i=0;i<FILAS;i++)
       for(j=0;j<COLUMNAS;j++)
          {
           printf("Dame un numero entero para almacenarlo en la posicion
                   [%i,%i]: ",i,j);
           scanf("%i",&matriz[i][j]);
          }
    printf("\n");
    for(i=0;i<FILAS;i++)
       {
        for(j=0;j<COLUMNAS;j++)
           printf("%i ",matriz[i][j]);
        printf("\n");
       }
    system("pause");
    return 0;
   }
```

3.4. ESTRUCTURAS

Una estructura o **struct** en C es un registro o estructura heterogenea que es capaz de alojar datos de diferentes tipos, por ejemplo nombre y apellido de una persona, su edad, número de cuenta bancaria y domicilio. Cada porción de información en un struct se denomina campo y se puede acceder a el por su nombre.

3.4.1. Definición y uso de estructuras (structs)

La implementación en C de una estructura o registro se lleva a cabo utilizando el tipo de dato ***struct*** el cual agrupa un conjunto de campos los cuales pueden ser variables de tipos básicos o definidos por el usuario.

El siguiente ejemplo define un tipo de dato estructurado llamado ***t_libro*** que consta de cuatro campos, cada uno del tipo más adecuado para representar la definición del concepto "libro" y que podría servirnos para mantener y procesar la información de los libros que hay en una librería:

```
typedef struct
   {
     char titulo[20];
     char autor[30];
     float precio;
     int edición;
   } t_libro;

. . .
t_libro libro1, libro2;
```

O bien de esta otra forma:

```
Struct t_libro
   {
    char titulo[20];
     char autor[30];
     float precio;
     int edición;
   };
. . .
Struct t_libro libro1, libro2;
```

Se puede entonces acceder a cada uno de los campos que conforman una estructura, tanto para lectura como para escritura especificando el nombre de la variable registro, seguido del nombre del campo en cuestión, separados con un punto:

```
libro1.titulo=”Rayuela”;
strcpy(libro1.autor,”Julio Cortazar”);
libro1.precio=45.00;
libro1.edicion=1;
```

Se pueden copiar todos los campos de una variable de tipo registro a otra del mismo tipo, utilizando una sola operación de asignación:

```
libro2=libro1;
```

3.4.2. Estructuras (structs) jerárquicas

Los campos de los registros pueden ser de cualquier tipo definido por el usuario, incluso también registros. Un registro con uno o más campos de tipo registro se denomina registro jerárquico o estructura jerárquica. Por ejemplo un registro que contenga datos de una asignatura en donde algunos de sus campos sean a su vez registros:

```
typedef struct
   {
    int dia;
    int mes;
    int anio;
   } t_fecha;

struct persona
   {
    char nombre[50];
    char apellido[50];
   } ;

struct asignatura
   {
    char nombre[50];
    int num_hrs;
    struct persona profesor;
    struct persona ayudante;
    char salón[50];
    t_fecha fecha_inicio;
    t_fecha fecha_examen_ord
    t_fecha fecha_asignatura[40];
   };
```

El acceso a los campos de un registro anidado en otro puede hacerse como se muestra a continuación:

. . .

```
struct asignatura curso;

curso.nombre=”Programacion I”;
curso.profesor.nombre=”Juan Morales”;
curso.fecha_examen_ord.dia=10;
curso.fecha_examen_ord.mes=5;
curso.fecha_examen_ord.anio=2013;
```

3.4.3. Tablas

El uso combinado de arreglos lineales y registros (structs) da lugar a la creación de tablas. En otras palabras se pueden utilizar **arreglos de registros** los cuales permiten manipular colecciones de estructuras, por ejemplo un directorio telefónico. LA definición de tabla es muy simple y consiste en implementar un arreglo lineal donde cada casilla del mismo almacena un registro o struct. Gráficamente tenemos:

	CAMPOS				
	Nombre	**Apellido**	**Direccion**	**Telefono**	**Edad**
Registro 0	Juan	Pérez	Avda. San Manuel	222-2-11-11-11	38
Registro 1					
Registro 2					

A continuación un ejemplo que ilustra el uso de tablas:

```
#include <stdio.h>
#define MAX_ALUMNOS 100

typedef struct
   {
    char nombre[30];
    int edad;
    long matricula;
    int nrc_pe;
    char nombre_pe[20];
   } t_alumno;

typedef t_alumno t_tabla_alumnos[MAX_ALUMNOS];
```

```
int main()
  {
   t_tabla_alumnos escuela;
   int i,numAlumnos;

   printf("\n Cuantos alumnos vas a capturar: " );
   scanf("%d",&numAlumnos);
   while (getchar()!='\n');

   for(i=0;i<numAlumnos;i++)
      {
       system("cls");
       printf("\nALUMNO %d",i+1);
       printf("\nNombre? ");
       gets(escuela[i].nombre);
       printf("\nedad? ");
       scanf("%d",&escuela[i].edad);
       while (getchar()!='\n');
       printf("\nMatricula? ");
       scanf("%ld",&escuela[i].matricula);
       while (getchar()!='\n');
       printf("\nNRC del Programa Educativo? ");
       scanf("%d",&escuela[i].nrc_pe);
       while (getchar()!='\n');
       printf("\nNombre del Programa Educativo? ");
       gets(escuela[i].nombre_pe);
       printf("\n");
       system("pause");
      }

    system("cls");
    printf("\nTABLA DE REGISTROS CAPTURADOS");
    printf("\nNOMBRE\t\tEDAD\t\tMATRICULA\t\tNRC\t\tPE");
    printf("\n_____________________________________________");

    for(i=0;i<numAlumnos;i++)
    printf("\n%s\t\t%d\t%ld\t\t%d\t%s",escuela[i].nombre,
            escuela[i].edad,escuela[i].matricula,
            escuela[i].nrc_pe,escuela[i].nombre_pe);
    printf("\n");
    system("pause");
    return 1;
  }
```