Unidad académica: Ingenierías
Facultad: Facultad de Ingeniería Informática
Profesor: Carlos A. Rodriguez

LENGUAJE C

1 LENGUAJE C INTRODUCCIÓN 5

2 ELEMENTOS DE PROGRAMACIÓN EN C 6

2.1 Partes de un programa en C 7
2.1.1 Directiva include (Archivos con extensión h) 8
2.1.2 Directiva #define: (Definición de Macros) 9
2.1.3 Declaración de variables 10
2.1.4 Función scanf(): 11
2.1.5 Función printf( ): 14

3 OPERACIÓN DE ASIGNACIÓN: 21

3.1 Operadores aritméticos 23
3.1.1 Operador residuo 23
3.1.2 Operadores de incremento y decremento 24
3.1.3 Otros operadores de asignación 25
3.1.4 Jerarquía de los operadores aritméticos 26
3.1.5 Conversión de tipo de datos 29
3.2 Definición de funciones 31
3.2.1 double log(double x): 32
3.2.2 double exp(double x): 33
3.2.3 double sqrt(double x): 33
3.2.4 double pow(double base, double exp): 33
3.2.5 double sin(double x), double cos(double x) 34
3.2.6 double atan(double x) 34
3.2.7 double atan2(double y, double x) : 34
3.2.8 double abs(double x) : 34
3.2.9 double fabs(double x) : 34
3.2.10 double floor(double x): 35
3.2.11 double fmod(double x, double y) : 35

4 INSTRUCCIONES DE CONTROL y CICLOS REPETITIVOS 35

4.1 Flujo de control 35

4.2 Expresión lógica 36
4.2.1 Operadores de relación 36
4.2.2 Operadores lógicos 37
4.2.3 Prioridad de operadores: 38
4.3 Instrucción if (expresión lógica) else 39
4.4 Sentencia Compuesta 41
4.5 If Anidados 44
5 INSTRUCCIÓN WHILE 45
6 INSTRUCCIÓN DO / WHILE (EXPRESION LÓGICA) 49
7 LA INSTRUCCIÓN FOR 52
8 EL OPERADOR COMA 54
8.1 Variación del ciclo for 55
9 INSTRUCCIÓN BREAK 56
10 INSTRUCCIÓN CONTINUE 56
11 OBTENCIÓN DE NUMEROS ALEATORIOS 58
11.1 La función rand() 59
12 INSTRUCCIÓN SWITCH 59
13 VECTORES Y MATRICES EN C: 63
13.1 Declaración de un vector 64
13.2 Inicialización de arreglos 65
13.3 Inicialización en tiempo de ejecución 66
13.4 Arreglos bidimensionales 69
13.5 Inicialización de arreglos matricial 69
13.6 Inicialización en tiempo de ejecución 71
13.7 Escribir una matriz 72
14 CADENAS DE CARACTERES 84
14.1 En el momento de declararlos 85
14.1.1 Por medio de una asignación de cada uno de los caracteres 86
14.1.2 Por medio de una instrucción de Lectura 86

15 FUNCIONES PARA EL MANEJO DE CADENAS 87

15.1 Función strcpy() 87
15.2 Función strcat() : 88
15.3 Función strlen(): 88
15.4 Función strcmp (const char *s1, const char *s2) 89

16 VARIABLES DE TIPO PUNTERO 91

16.1 ¿Que es una variable de tipo puntero? 91
16.2 ¿Cómo se declara una variable de tipo puntero? 92
16.3 ¿Cómo se le da valores a una variable de tipo puntero? 93
16.3.1 Utilizando el operador & 93
16.3.2 El operador *, operador de indirecto o 94
16.3.3 Aritmética de punteros 97

17 INICIALIZACIÓN DE UN PUNTERO A CARACTERES 100

18 ASIGNACIÓN DINÁMICA DE MEMORIA 104

18.1 Función malloc 104
18.2 La Función free 105
18.3 La función sizeof 105

19 FUNCIONES 109

19.1 Salida de una función 112
19.2 Valor devuelto 112
19.3 El uso de los prototipos 113
19.4 Llamada de una función 113
19.5 Paso de parámetros por valor , llamada por referencia 114
19.6 El uso de los punteros en los parámetros 115
19.7 Paso de Arrays unidimensionales a funciones 120
19.8 Paso de Arrays bidimensionales a funciones 121
19.9 El uso de los punteros como parámetros para procesar vectores 124
19.10 El concepto de matriz como vector de puntero a punteros 127

20 QUÉ DEVUELVE MAIN ( ) 130

20.1 Argumentos de main() 131

21 ESTRUCTURAS 133

21.1 Declaración de estructuras y variables de tipo estructura.....133

22 DEFINICIÓN DE VARIABLES DE TIPO ESTRUCTURA..... 134

22.1 La palabra typedef.....136
22.2 Como se entran datos a un campo de una variable estructura.....138
22.2.1 Entrando un dato con la instrucción scanf..... 138
22.2.2 Entrando un dato con la instrucción de asignación..... 139

23 ARRAY DE ESTRUCTURAS..... 140

24 PUNTEROS A ESTRUCTURAS..... 142

24.1 El operador _ 143

25 ARCHIVOS EN C..... 144

25.1 ¿Qué es un archivo?.....144
25.2 Clasificación de los archivos.....144
25.3 Clasificación por la forma en que se almacena la información.....144
25.3.1 Archivos de Texto..... 144
25.3.2 Archivos Binarios..... 145
25.4 Clasificación por la forma en que se accede a la información.....145
25.4.1 Archivos Secuenciales..... 145
25.4.2 Archivos Aleatorios..... 145
25.5 Flujos.....146

26 ARCHIVOS ALEATORIOS EN C..... 146

26.1 El puntero File.....146
26.2 Apertura de un archivo aleatorio.....147
26.3 Escritura en un archivo aleatorio.....148
26.4 Leer los datos de un archivo aleatorio.....149
26.4.1 Leer de manera aleatoria un dato en un archivo binario151
26.5 Realizando cambios en un archivo Aleatorio.....152

27 BIBLIOGRAFIA..... 154

LENGUAJE C

E.I. Carlos A. Rodríguez C.

1 LENGUAJE C INTRODUCCIÓN[1]

C es un lenguaje de programación de propósito general que ha sido estrechamente asociado con el sistema UNIX en donde fue desarrollado. Las ideas importantes del lenguaje provienen del lenguaje BCPL, desarrollado por Martin Richards. Luego otro lenguaje que influyo indirectamente sobre el C fue el lenguaje B, el cual fue escrito por Ken Thompson en 1970 para el primer sistema UNIX de la máquina DEC PDP-7.

Algunas de las características de C son:

- **Proporciona una variedad de tipos de datos como: carácter, enteros y números de punto flotante de varios tamaños.**
- **Existe una jerarquía de tipos de datos derivados, creados con apuntadores, arreglos, estructuras y uniones.**
- **Las expresiones se forman a partir de operadores y operandos. Los apuntadores proporcionan una aritmética de direcciones independiente de la máquina.**
- **C es un lenguaje de relativo bajo nivel.**
- **C no proporciona operaciones para tratar directamente con objetos compuestos tales como cadenas de caracteres, conjuntos, listas o arreglos.**
- **Aunque algunas limitaciones como las anteriores y otras que se mencionan en el libro el Lenguaje de Programación C pueden parecer limitantes, hacen de C un lenguaje liviano que se puede**

[1] E.I. CARLOS A RODIGUEZ C

describir rápidamente, y por tanto aprender de la misma manera.

El lenguaje fue desarrollado por Brian W. Kernighan, Dennos M. Ritchie y su primera definición se hizo en la primera edición del libro "El lenguaje de programación C" de dichos autores.

2 ELEMENTOS DE PROGRAMACIÓN EN C

Como lo dicen los autores del lenguaje C, la mejor manera de aprender un lenguaje es mediante la escritura de programas, veamos como se puede usar este con un primer problema.

Ejemplo

Crear un programa que permita sumar dos números a y b. El programa debe escribir la suma y los dos números sumados.

Análisis:

Datos de entrada:

Dos números reales a y b

Datos de salida:

La suma de los dos números.

El siguiente es el algoritmo del problema:

Variables
Real a, b, s;
Escriba (“Ingrese los valores a y b a sumar”);
Leer(a,b)
s=a+b;
Escriba(a,”+”,b,”=”,s);

```
#include <stdio.h>
#include <conio.h>
#define MENSAJE “Termine”

int main(int argc, char *argv[])
{
    double a,b,s;

    printf("Ingresar los dos sumando");
    scanf("%lf %lf", &a,&b);
    s=a+b;
    printf("%lf + %lf = %lf \n",a,b,s);
    printf(MENSAJE);
    getchar();
    return 0;
}
```

2.1 Partes de un programa en C

Un programa en escrito en C está conformado por pequeños módulos de código llamados funciones.

El módulo principal o programa principal está conformado por la función que denominamos main( ) las instrucciones que conforman ésta se encierran entre { }.

Sobre la parte superior de la función main() y dentro de la función se pueden declarar los siguientes objetos que pueden manipular un programa.

1. Directivas para Incluir archivos, #include

2. Directivas para incluir macros y constantes #define

3. Declaración de Prototipos de funciones

4. Declaración de Variables globales

5. Declaración de tipos de datos.

6. Dentro del main() se encuentran las declaraciones de variables y las instrucciones que definen procesos que debe realizar el computador.

7. Por fuera del main() y después de la llave } que cierra el main aparecen las declaración de las funciones.

2.1.1 Directiva include (Archivos con extensión h)

Los archivos con extensión h (Ejemplo: stdio.h, conio.h etc) se llaman archivos de cabecera, contienen información técnica que requiere el compilador para la buena operación de las funciones que usan nuestros programas.

El programa está usando las siguientes directivas para incluir archivos son:

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

Los archivos con extensión h vienen con el compilador. Pero también podemos crear archivos de este tipo propios.

Cada función en el manual tiene la referencia al archivo h.

2.1.2 Directiva #define: (Definición de Macros)

La directiva #define se utiliza para definir un identificador y una cadena que será sustituida por cada vez el identificador se utilice en el programa. Al identificador se le denomina nombre de la macro y al proceso de reemplazo sustitución de macro.

Nuestro programa está usando directivas define para definir la macro MENSAJE

Ejemplos:

```
#define MENSAJE "Termine presione Tecla"
```

Otro tipo de macro podría ser:

```
#define CIERTO 1
#define FALSO 0
```

Una vez se define una macro se puede usar el nombre de ella como parte de la definición de otra, ejemplo:

```
#define UNO 1
#define DOS UNO + UNO
#define TRES UNO + DOS
```

Mas adelante veremos otros tipos de macros que se pueden definir en C y que pueden ser útiles.

2.1.3 Declaración de variables

Antes del main() o después de él se deben definir las variables, ejemplo:

```
float a, b, x;
float y, c;
float d, e, f;
```

En una declaración de variables le estamos diciendo al compilador que debe: reservar espacio en memoria, que a cada espacio en memoria le asigne un nombre y un número determinado de byte, también se le dice que tipos de datos puede almacenar, el compilador deduce esa información del tipo de dato que se escribe con cada declaración.

Las variables que se declaran dentro del main() se dice que son variables locales, las que se declaran fuera se dicen globales, cuando veamos el concepto de función ampliaremos la diferencia entre estos tipos de variables.

La siguiente tabla muestra los distintos tipos de dato que podemos usar en C para declarar variable en un programa en C.

TIPOS DE DATOS BÁSICOS RECONOCIDOS POR TURBO C			
TIPO	**TAMAÑO en Bytes**	**RANGO de Valores que Puede Almacenar**	**Cuando Usamos**
char	1	0..255	'Á'...'Z', $
Int	2	-32767..32768	Para almacenar números enteros
Double	8	1.7E-308..1.7E308	Para almacenar números Reales
Flota	4	3E-38..3E38	Variables que almacenan números reales.
Pointer	2 byte	Punteros near, __es, _es, ss	Para variables que guardan direcciones de memoria
Long double	10 bytes	3.4E-4932 a 1.1 E 4932	Variables Reales

2.1.4 Función scanf():

Permite entrar datos por el teclado y almacenarlos en las variables en memoria.

Prototipo de la función scanf():

int scanf(char *cadena_formato, lista de variables);

La función scanf lee los datos que son escritos desde el teclado, examina la cadena_formato, y convierte los caracteres escritos en el formato especificado, para luego colocarlos en memoria.

1. Donde la cadena_formato consta de tres caracteres

2. Especificador de formato

3. Caracteres de espacio en blanco (' '), el tabulador ('\t') o el salto de línea

('\n')

4. Caracteres Simples

La función scanf( ) devuelve el número de datos a los que se les asigno exitosamente un lugar en memoria en el momento de la lectura. Los especificadores de formato de entrada van precedidos por el signo % y le indican a la función scanf a que tipo de dato se va a convertir, el dato que se digite en esa entrada.

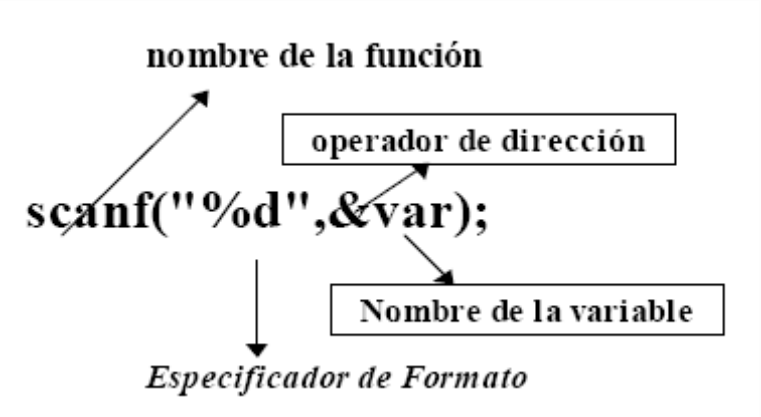
Si alguno de los datos entrados scanf no puede convertirlos en algún de los tipos de datos especificados, la ejecución de la función terminará; asignando los caracteres leídos hasta ese punto a su argumento, algunos de los especificadores, más comunes son:

ESPECIFICADOR DE FORMATO	TIPO DE DATO
%c	Leer un sólo carácter
%d	Leer un entero decimal
%i	Leer un entero decimal
%l	Leer un entero largo
%e	Leer un entero en coma flotante
%f	Leer un entero en coma flotante
%h	Leer un entero corto
%o	Leer un numero octal
%s	Leer una cadena de caracteres
%x	Leer un número hexadecimal

Los espacios en blanco en la cadena de control hacen que la función scanf salte un o más caracteres en la secuencia de entrada. Lo anterior significa que un espacio en blanco dentro de la cadena de formato hace que scanf lea pero no almacene los espacios en blanco hasta que encuentre el primer dato que no sea de ese tipo.

Ejemplo:

El siguiente cuadro muestra los elementos de la sintaxis de la función scanf:



El especificador de formato como *%d* le indica que los caracteres que se escriban en el teclado deben ser convertidos en este caso a un número entero.

La variable de nombre var debe ser declarada de tipo int.

La expresión *&var* es la dirección en memoria donde se encuentra la variable var en memoria y donde se quiere almacenar el dato que se va a ingresar.

2.1.5 Función printf( ):

La función printf nos permite en el lenguaje C imprimir en pantalla mensajes y/o valores en la pantalla del computador.

Prototipo de la función printf:

int printf(char * cadena_formato, lista de argumentos);

La cadena de formato consiste de una secuencia de caracteres simples, secuencias de escape y/o especificadores de formato, todos ellos delimitados por " (comillas dobles).

Secuencia de Escape	Carácter de Salida
\a	Sonar alerta
\b	Retroceso
\f	Salto de Página
\n	Salto de Línea
\t	Tabulador Horizontal
\v	Tabulador Vertical
\r	Retorno de Carro
\"	Doble Comillas
\'	Comilla Simple
\\	Barra Invertida
\?	Signo de Interrogación
\DDD	Código Octal
\xHHH	Código Hexadecimal de un carácter

Especificador de formato:

Las especificaciones de formato vienen dadas por el símbolo de % seguido de uno o más especificadores permitidos, por la sintaxis:

%[<banderas>][<ancho>][<.prec>] [{h|l|L}[2]]tipo

[2] Son los prefijos que se utilizan antes de los caracteres de tipo.

Carácter tipo	Tipo Argumento	Formato de Conversión
d	entero	Entero decimal
i	entero	entero decimal con signo

Carácter tipo	Tipo Argumento	Formato de Conversión
u	entero	entero decimal con signo
o	entero	entero octal sin signo
x	entero	entero hexadecimal sin signo
X	entero	entero hexadecimal sin signo
f	punto flotante	valor con signo [-]dddd.ddd
e	punto flotante	valor con signo [-]d.ddde[signo]ddd
E	punto flotante	valor con signo[-d]d.dddE[signo]ddd
g	punto flotante	salida como en f o en e dependiendo del valor de precisión
G	punto flotante	salida como en f o en e dependiendo del valor o precisión
c	carácter simple	carácter simple
s	Cadena	cadena de caracteres
P	puntero	dirección almacenda en el puntero

<bandera>:

Es opcional, con ella especificamos cómo va aparecer la conversión del número cuando se escriba. Puede ser cualquiera de las que se indican en la tabla siguiente:

BANDERA	EFECTO
-	Justifica el resultado por la izquierda Si se omite se ajusta a la derecha.
+	Escribe el signo (+ **o** -) del resultado
' '	Escribe un espacio blanco a la cabeza de lo que escribe, si el dato es positivo, o un signo - si el resultado es negativo

<ancho>:

Es un número entero positivo, opcional con el cual se indica el mínimo número de caracteres con los que se debe presentar un dato de algún tipo.

Si el número de caracteres a imprimir es menor que el ancho especificado, la salida se rellenará con espacios en blanco. En caso contrario el ancho será ignorado.

Si la longitud se encabeza con un cero, la salida se rellenará con ceros en lugar de blancos.

Si la longitud es un **, printf* utilizará el valor suministrado por la lista de argumentos, como la especificación de longitud.

<.prec>:

Es opcional y perteneciente a la especificación de formato, indica la precisión del valor de salida y siempre aparecerá precedida de un punto para separarla de otras especificaciones.

Con un número entero después del punto se especifica el número de posiciones decimales que se han de imprimir para un número en punto flotante.

Si lo anterior se aplica a enteros, el número que sigue al punto especifica la longitud máxima del campo. Si la cadena es más larga que la longitud máxima del campo, se truncarán los caracteres finales de la derecha.

```
#include "stdio.h"
#include "conio.h"

#define MENSAJE "Termine presione tecla"

int main( )
{ double a,b,s;

  printf("Ingresar los dos sumando");
  scanf("%lf %lf", &a,&b);
  s=a+b;
  printf("La suma:%7.3lf + %7.3lf = %7.3lf \n",a,b,s);
  printf("La suma:%-7.3lf %-+7.3lf = %7.3E \n",a,b,s);
  printf("La suma:% 7.3lf + %7.3lf = %10.2E \n",a,b,s);
  printf("La suma:% 7.3lf + %7.3lf = %10.5e \n",a,b,s);
  printf(MENSAJE);

  getche();

  return 0;
}
```

Observe como los prefijos h|l|L se anteponen al carácter de tipo así:

h: Se utiliza como prefijo de los tipos d, i, o, x y X, para especificar que el argumento es short int, o con u para especificar un short unsigned int.

l: Se utiliza como prefijo con los tipos d, i, o, x y X, para especificar que el argumento es long int, o con u para especificar

un long unsigned int. También se utiliza con los tipos e, E, f, g, y G para especificar un double antes de un float.

L: Se utiliza como prefijo con los tipos e, E, f, g y G, para especificar long double. Este prefijo no es compatible con el ANSI C.

3 OPERACIÓN DE ASIGNACIÓN:

Es la operación mediante la cual se le asigna un valor determinado a una variable en memoria.

SINTAXIS:

<Identificador>=<expresión>

Con toda expresión de la forma anterior le estaremos indicando al computador que: evalúe la expresión y lo almacene en la variable que se identifica por el identificador.

Donde una expresión puede ser un valor de constante, o fórmula matemática.

Ejemplo:

Los siguientes son algunas de las expresiones de asignación en C.

Almacena en la variable horas el valor 30, se escriben en C así:

```
Horas=30;
salario=5000;
```

Si desea almacenar un carácter en un sitio de memoria

caracter='C';

Observe que el carácter se encierra entre comillas simples.

Si desea agregar 1000 al valor que existe en salario y el resultado almacenarlo en

salario escriba así:

salario=salario + 1000;

Después de la operación anterior la variable salario tiene almacenado el 6000 y el valor de 5000 se perdió.

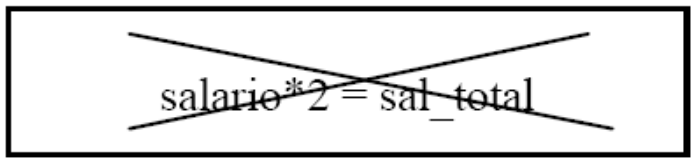
Si desea multiplicar por las horas el valor almacenado en la variable salario, haga lo siguiente:

salario=salario*horas;

Después de la asignación anterior en la variable salario se almacena el valor de salario es 180000.

Ejemplo:

Por ultimo tenga presente que NUNCA se escribe la expresión a la izquierda del operador de asignación:



3.1 Operadores aritméticos

Para indicar que vamos a realizar alguna operación aritmética en nuestros programas tenemos operadores.

OPERADOR	OPERACIÓN QUE REALIZA
+	Sumar
-	Restar
*	Multiplicar
/	Dividir
%	Calcular el Residuo de división entera
++	Incremento
--	Restar

Los operadores +,-,*,/, realizan las operaciones aritméticas tradicionales.

3.1.1 Operador residuo

El operador % calcula el residuo que queda al dividir dos números enteros, el siguiente programa muestra lo que realiza ese operador:

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

int main(int argc, char *argv[])
{
   int a=3,b=2 ;
   printf("%d",a%b);
   printf("\nPresione cualquier tecla");
   getche();
   return 0;
}
```

3.1.2 Operadores de incremento y decremento

Los operadores ++ o – son operadores que no existen en lenguajes como BASIC y PASCAL y que permiten abreviar muchas expresiones aritméticas.

Suponga que tenemos el siguiente programa en C:

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

void main( void)
{       int a=3 ;
        int e,c=0,d=23,b=5;

        printf("a=%d \t d=%d \tb=%d\t e=%d \t c=%d",a,d,b,e,c);
        a=a+1; c=b++,
        e=++d;
        printf("\na=%d \t c=%d b=%d e=%d",a,c,b,e);
        printf("\n Termine presione tecla");
        getche();
}
```

3.1.3 Otros operadores de asignación

En C existen los siguientes operadores de asignación: +=, *=, =, /= y %=, donde la sintaxis cuando se quiere utilizar uno de esos operadores es :

<identificador><operador de asignación><expresión>

Veamos algunos ejemplos del uso de esos operadores:

EXPRESIÓN	EXPRESIÓN EQUIVALENTE
a=a+2	a+=2 ;
b=b+(i+1)	b+=i+5 ;
z=z/10	z/=10 ;
j=i%(j-2)	j%=(j-2)

Es claro que el identificador es una variable.

3.1.4 Jerarquía de los operadores aritméticos

Cuando el computador evalúa una expresión aritmética sigue los siguientes criterios para obtener el resultado.

1. En una operación aritmética que incluya varios operadores aritméticos, los operadores ++,-- tienen la máxima prioridad, le siguen en prioridad *, /, %.

2. Lo anterior significa que C primero realizan las operaciones que están asociadas con los operadores aritméticos ++ o—que con los otros operadores.

3. En una expresión aritmética compuesta, las operaciones que están asociadas con los operadores +, - se ejecutan después de

haberse ejecutado todos los operadores aritméticos enunciados en la primera regla.

4. Si en una expresión existen varios operadores aritméticos que tengan la misma prioridad, estos se resuelven de izquierda a derecha.

EJEMPLOS:

Cuando vamos a escribir expresiones algebraicas con C debemos tener cuidado al escribirlas, teniendo en cuenta las prioridades de los operadores.

Discutamos esos problemas.

Suponga que tiene la expresión algebraica.

Z = Z + (B - C)/D

y la escribimos en un programa en C así:

Z =Z+B+C/D,

La expresión anterior el computador la evalúa así :

Primero calcula C/D luego Z + B y por último suma los resultados anteriores.

Por qué suma Z + D? porque después de hacer la división, las dos operaciones que quedan por hacer tienen la misma prioridad, entonces realiza primero la que está más a la izquierda.

Supongamos ahora que algebraicamente se tiene:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Si en un programa escribimos :

```
X = -b + sqrt(b * b - 4 * a * c)/ 2 * a;
```

No se extrañe si al evaluar esa expresión el computador le da unos valores incorrectos.

Por la prioridad de los operadores involucrados en la expresión, el computador haría lo siguiente:

Primero evalúa b*b luego evalúa 4*a*c , fuego hace la resta b*b - 4*a*c para calcular la raíz cuadrada al valor así obtenido y luego ese valor lo divide por 2 y ese resultado lo multiplica por a para finalmente sumarle -a, es claro que el resultado no debe ser correcto.

La expresión queda bien escrita si usa los paréntesis así:

```
X = (-b + sqrt(b * b - 4*a*c)) / (2 * a);
```

3.1.5 Conversión de tipo de datos

Se dice que una expresión de asignación involucra operaciones de modo mixto cuando involucra variables de distinto tipo. Cuando C y C++, evalúan una expresión que involucra variables de distinto tipo realiza conversión automática de un tipo de dato a otro, Que implicaciones tiene eso?

Es bueno tener presente que los datos de diferente tipo son almacenados de modo diferente en memoria.

Suponga que ha sido almacenado el número 123. Su representación depende de su tipo, esto es el patrón de ceros y unos en memoria será diferente dependiendo de si 123 es almacenado como un int o de si es almacenado como float.

EJEMPLO:

Si en un programa en C tenemos que a y b son variables de tipo float y c es de tipo int, las siguientes expresiones:

```
a=5.0 ;
b=2.0 ;
c=a/b ;
```

Almacenarían en c el valor 2 pues la división da 2.5 pero al convertirlo a entero lo

Trunca.

EJEMPLO:

Suponga que se ejecuta la operación siguiente:

resul_float=valor_float*vlalor_int,

y la variable valor_float es declarada de tipo float y la variable valor_int, es declarada entera.

La operación anterior es de tipo mixto, cuando se va a realizar la multiplicación el valor de la variable valor int se convierte a número float y luego se hace la multiplicación. El proceso se realiza así: el computador lee el valor almacenado en la variable y es convertido a float para la realización de la multiplicación pero sin cambiar el valor en memoria, cuando se realiza la multiplicación el resultado es float.

En conclusión tenga en cuenta que cuando C tiene que evaluar una expresión en la que intervienen operándos de diferentes tipos, *primero convierte, sólo para realizar las operaciones solicitadas, los valores de los operandos al tipo del operando cuya precisión sea más alta*. Cuando *se trata de una asignación, convierte el valor de la derecha al tipo de la variable en la cual se va asignar siempre que no haya pérdida de información*. En otro caso, C exige que la conversión se realice explícitamente.

La siguiente tabla muestra la jerarquía de conversiones desde la prioridad más alta hasta la más baja:

double
float
long
int
short

EJEMPLO:

Supongamos que tenemos el siguiente programa:

```
valorl_int=3 ;
valor2_float=4 ;
valor_float=7.0 ;
resul_float=valor_float + valorl_int/valor2_float;
```

El computador al dividir valorl_int/valor2_float dicha operación es de tipo mixta por tanto el 3 se convierte a número float y la división da 0. 75 que al sumarle 7 da 7.75.

3.2 Definición de funciones

Las computadoras no saben realizar por si mismos operaciones complejas como: calcular un logaritmo, evaluar la función seno, la tangente, por tal motivo es necesario que los lenguajes, den al programador, instrucciones que le permitan al computador saber como calcular dichos valores.

El C del estándar ANSI define 22 funciones matemáticas que entran en las siguientes categorías: Funciones trigonométricas, hiperbólicas, logarítmicas y exponenciales, otras.

Todas las funciones matemáticas necesitan que se incluya el archivo de cabecera math.h en cualquier programa que las utilice.

3.2.1 double log(double x):

Permite calcular el logaritmo neperiano (base e) del argumento x. Produce error si x es negativo o si x =0.

EJEMPLO:

En un programa en C se puede presentar la siguiente instrucción así:

Y= log (x) + 6;

3.2.2 double log10(double x)

Permite calcular el logaritmo en base 10 del argumento x. Produce error si x es negativo o si x =0.

EJEMPLO:

En un programa en C se puede presentar la siguiente instrucción así:

Y= log 10(x) + 6;

3.2.2 double exp(double x):

Permite calcular el exponencial del argumento x, es decir permite calcular ex

EJEMPLO:

En un programa en C se puede tener que:

Y= exp(x);

3.2.3 double sqrt(double x):

Permite calcular la raíz cuadrada del argumento. El argumento debe ser mayor o igual que cero y real, el resultado es real. Se usa de igual manera que las funciones anteriores.

3.2.4 double pow(double base, double exp):

Nos devuelve el argumento base elevado a exp(baseexp)

EJEMPLO:

El siguiente programa escribe dos elevado al cuadrado

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

void main(void)
{
        printf("%lf"',pow(2,2))
 }
```

3.2.5 double sin(double x), double cos(double x)

Estas dos funciones nos permiten calcular Seno y Coseno respectivamente de sus argumentos, dado en radianes. Se usa de igual manera que las funciones anteriores.

3.2.6 double atan(double x)

Esta función devuelve el arco tangente de x. El valor de x debe estar en el rango de -1 a l; en cualquier otro caso se produce un error de dominio. El valor se especifica en radianes.

3.2.7 double atan2(double y, double x) :

Esta función devuelve el arco tangente de y/x. Utiliza el signo de su argumento para obtener el cuadrante del valor devuelto. El valor de x se especifica en radianes.

3.2.8 double abs(double x) :

Calcula el valor absoluto de un número dado.

3.2.9 double fabs(double x) :

Devuelve el valor absoluto de x.

3.2.10 double floor(double x):

Toma el argumento y retorna el mayor entero que no es mayor que x. Por ejemplo floor de 1.02 devuelve 1.0, el floor de -1.02 devuelve -2.

3.2.11 double fmod(double x, double y) :

La función fmod calcula el residuo de la división entera de x/y.

4 INSTRUCCIONES DE CONTROL y CICLOS REPETITIVOS

4.1 Flujo de control

Se llama flujo de control de un programa al orden en que se ejecutan las instrucciones que lo conforman.

El flujo de control de los programas, es lineal, esto significa que el computador ejecuta una a una las instrucciones que se le indican, sin alterar el orden en que se escriben.

Pero en muchos procesos que es necesario programar, es importante indicarle al computador que ejecute un conjunto determinado de instrucciones, cuando se cumpla una determinada condición y que ejecute otras cuando la condición no se cumpla, para esos casos el lenguaje C nos da la instrucción if else.

4.2 Expresión lógica

Una expresión lógica en C es una sentencia que al ser evaluada, el computador da un valor 0 si es falsa y un valor distinto de cero si es verdadera.

4.2.1 Operadores de relación

Los siguientes operadores los utilizaremos para construir expresiones lógicas y establecen relaciones que pueden ser falsas o verdaderas.

OPERADOR	SIGNIFICADO
>=	Mayor o igual que
<=	Menor o igual que
= =	Igual que
!=	Diferente
<	Menor que
>	Mayor que

EJEMPLO:

Cualquiera de las siguientes expresiones, son validas en un programa escrito en C o C++.

A+B>=C*2

La expresión anterior compara el valor de la suma de A y B con el doble del valor de C, si la suma es mayor o igual que el doble de C entonces el resultado de la expresión es 1, en otro caso es 0.

La expresión: X= ='S', compara el valor almacenado en la variable X con la letra S, el resultado da diferente de cero si en la variable X hay una letra distinta a la S.

La expresión: X<>S compara los valores almacenados en las variables con nombre X y S si son iguales la expresión da 0, sin son diferentes el valor es distinto de cero.

La expresión: z= a+b>=c*2; compara el valor de la suma de a y b con el doble de c, si es menor almacena en z el valor 0 si es mayor o igual almacena un valor distinto de cero.

4.2.2 Operadores lógicos

C se tiene los siguientes operadores lógicos para formar expresiones lógicas más complejas.

OPERADORES	SIGNIFICADO
&&	y
||	o
!	no

EJEMPLOS:

Si tenemos la expresión: !(5>3), al evaluarla el resultado que da es 0.

La expresión: z=!(5>3); almacena 0 en la variable z

La expresión: z=!(a+b>=c*2) almacena 1 en la variable z si el resultado de comparar a+b con el doble de c es menor, puesto que la proposición es falsa, al negarla da verdadera.

4.2.3 Prioridad de operadores:

Teniendo los operadores ya definidos, debemos definir la prioridad nuevamente así:

CATEGORÍA DEL OPERADOR	OPERADORES	ASOCIATIVA
Operadores Monarios	-,++,--,!, sizeof(tipo)	Derecha a Izquierda
Multiplicación división y residuo	*, /, %	Izquierda a Derecha
Suma y sustracción aritmética	+, -	Izquierda a derecha
Operadores de relación	<, <=, >, >=	Izquierda a derecha
Operadores de igualdad	= = , !=	Izquierda a derecha
y	&&	Izquierda a derecha
o	||	Izquierda a derecha
Operadores de asignación	=, +=, - =, *=, /=, %=	Izquierda a derecha

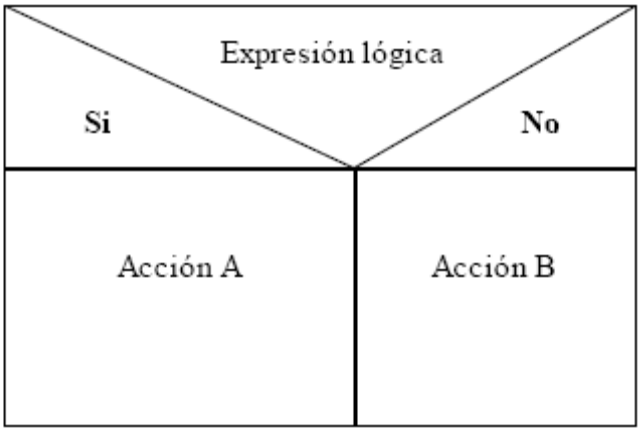
Las expresiones entre paréntesis se evalúan primero.

Los operadores de asignación tienen menor prioridad que todos los otros operadores. Por tanto las operaciones monarios, aritméticos, de relación, de igualdad y lógicos se realizan antes que las de asignación.

4.3 Instrucción if (expresión lógica) else

La instrucción si evalúa la expresión lógica, si ésta es verdadera, ejecuta la instrucción definida en <acción A>, si es falsa se ejecuta la instrucción inmediata al else definida en <acción B>, después de eso ejecuta la instrucción siguiente a la <acción B> y las que le siguen.

```
if (expresión lógica)
        <Acción A>;
else
        <Acción B>;
```



EJEMPLO:

Escriba un programa en lenguaje C que le permita a un estudiante calcular la nota definitiva de tres notas, que tienen el mismo valor porcentual. El programa debe imprimir la nota definitiva y un mensaje adecuado que diga 'Gano la materia', o 'Perdió la Materia', según si la nota sea mayor igual que cero o menor que cero.

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

void main()
{
     double nota1, nota2, nota3;
     double definitiva;

     printf("\nEscriba la primera nota");
     scanf("%lf",&nota1);
     printf("\nEscriba la segunda nota";
     scanf("%lf",&nota2);
     printf("\nEscriba la tercera nota");
     scanf("%lf",&nota3);
     definitiva=(nota1+nota2+nota3)/3
     if (definitiva>=3)
        printf("\nSu nota es %5.2lf  Gano la materia",definitiva);
     else
        printf("\n Su nota es  %5.2lf  Perdió la materia  ",definitiva);
}
```

Otra versión del programa usando la instrucción que hemos visto es:

```
#include <stdio.h>
#include <conio.h>
#define GANO "Gano la Materia"
#define PERDIO "Perdio la Materia"

void main()
{
  double nota1, nota2, nota3;
  double definitiva;

  printf("\nEscriba la primera nota");
  scanf("%lf",&nota1);
  printf("\nEscriba la segunda nota";
  scanf("%lf",&nota2);
  printf("\nEscriba la tercera nota");
  scanf("%lf",&nota3);
  definitiva=(nota1+nota2+nota3)/3
  if (definitiva>=3)
     printf("\nSu nota es %5.2lf  %s",definitiva, GANO);
  else
     printf("\n Su nota es  %5.2lf  Perdió la materia %s ",definitiva,
PERDIO);
}
```

4.4 Sentencia Compuesta

Cuando un programador, necesita que el computador ejecute más de una instrucción en el momento de que la expresión lógica sea verdadera o sea falsa, lo indica encerrando entre llaves { } las instrucciones que siguen al then o al else.

EJEMPLO:

Unidad académica: Ingenierías
Facultad: Facultad de Ingeniería Informática
Profesor: Carlos A. Rodriguez

Escriba un programa que lea tres números reales a, b, c. Una vez leídos, el programa debe calcular la solución a la ecuación:

$$A * X^2 + B * X + C = 0$$

y debe escribir las soluciones reales si existen y las soluciones complejas.

```
Variables
real a,b,c, x1, x2, discri;

Escriba("Ingrese coeficientes de ecua");
Leer(a,b,c);
discri=b^2 – 4*a*c;
Si (a!=0){
        Si (discri>=0){
                x1=(-b+sqrt(discri))/(2*a);
                x2=(-b-sqrt(discri))/(2*a);
                Escriba("x1=",x1);
                Escriba("x2=",x2);
        }sino{
                x1=-b/(2*a);
                x2=sqrt(abs(discri))/(2*a);
                Escriba("x1=", x1,"+i",x2);
                Escriba("x2=", x1,"-i",x2);
        }
}sino
        Si (b!=0){
                x1=(-c)/b;
                Escriba ("x1=",x1);
        }sino
                Escriba("No hay Ecuacion");
```

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

int main(int argc, char *argv[])
{
   double a,b,c, discri;
   double x1,x2;

     printf("\t\t\Entre los coeficientes de la
     ecuacion");
     scanf("%lf %lf %lf",&a,&b,&c);
     discri=pow(b,2.0)-4*a*c;
   if (a!=0)
     if (discri>=0){
         x1=(-b+sqrt(discri))/(2*a);
         x2=(-b-sqrt(discri))/(2*a);
         printf("\nx1=%-+10.3lf=",x1);
         printf("\nx2=%-+10.3lf=",x2);
     } else{
         x1=-b/(2*a);
         x2=sqrt(abs(discri))/(2*a);
         printf("\n  x1=%-+10.3lf   %-+6.3lf
         %-5c ",x1,x2,'i');
         printf("\n x2=%-+10.3lf  %-+10.3lf
         i\n",x1,-x2);
   } else
     if (b!=0){
        x1=-c/b;
        printf("\nx1=%10.3lf=",x1);
      }else
         printf("\n No puedo resolver ");

 }
```

4.5 If Anidados

Tendremos situaciones lógicas en las cuales después de hacernos una pregunta se nos hace necesario hacer una nueva pregunta, en esos caso tenemos entonces que después del then y/o else es necesario volver a tener una sentencia if then else decimos entonces que anidamos if.

EJEMPLO:

Se requiere un programa que lea A y permita evaluar la función

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void main()
{       double a,y;

        clrscr();
        printf(" Entre un numero real ");
        scanf("%lf",&a);
        if (a<3)
                y=5*a + sqrt(++a);
        else
                if ((a>=3)&&(a<=9))
                        y=a+5-(a+2)/3;
                else
                        y=exp(3*log(a))+7*a;
        printf("y=%7.2lf",y);
  getch();
}
```

$$y = \begin{cases} 5*x+\sqrt{x+1} & x > 3 \\ x+5-\dfrac{x+2}{3} & 3 <= x <= 9 \\ x^3+7*x & x >= 9 \end{cases}$$

Observe que el if después del else no se encierra entre llaves, pues el if else es una sola instrucción. Recuerde los if anidados, los usa el programador para crear tomas de decisión eficientes.

5 INSTRUCCIÓN WHILE

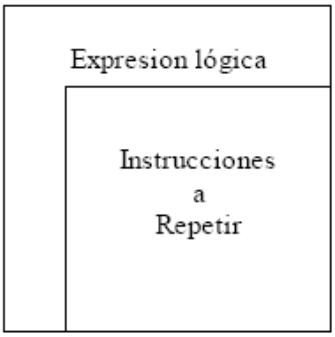
La instrucción while la utilizan los programadores para lograr que el computador ejecute de manera repetida durante un número finito de veces un conjunto de instrucciones.

La instrucción while hace que el computador repita la ejecución de las instrucciones mientras la condición es verdadera. En el lenguaje C esa instrucción tiene la siguiente sintaxis:

```
while (expresión lógica )
{

   Instrucciones a
   Repetir


}
```



Las llaves encerrando las instrucciones le indican al computador cuales son las instrucciones que se deben ejecutar mientras la condición se cumple.

Dentro de las instrucciones a repetir debe haber al menos una instrucción que haga que la condición sea falsa, de lo contrario no saldrá del ciclo.

EJEMPLO:

Escribir un programa que lea un número entero n, luego de lo cual el computador debe escribir todos y cada uno de los números entre 1 y n luego debe escribir cuanto vale la suma de esos números.

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

void main()
{       float    n, numero, suma       ;

        printf("\n Escriba numero hasta donde desea que cuente ?");
        scanf("%f",&n);
        numero=1; suma=0;
        while (numero<=n)
        {       suma += numero;
                printf("%5.2f",numero);
                numero += 1;
        }
        printf("La suma=%f",suma);
}
```

EJEMPLO:

Escriba un programa que lea un número entero n y luego calcule y escriba los n

primeros términos de la serie 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.....

Análisis:

Datos de entrada: n el número de terminus a generar

Salida: Secuencia de n números que se forman así: 0, 1, 1, 2, 3, 5, 8, 13, ...

¿Que debe hace la máquina?

Pues si analizamos con cuidado nos podemos dar cuenta de que se debe buscar la forma mediante operaciones aritméticas, de que el computador vaya generando cada un de los términos anteriores:

Si llamamos a_n el termino n-esimo a imprimir y le damos el valor inicial de 0, y

observamos que el siempre se obtiene de los dos términos: el anterior (n-1) y el

anterior al anterior (n-2) que llamaremos respectivamente ai, af y les damos

valores de 1 y 0 respectivamente, vemos que podemos generar la secuencia

fácilmente mediante la expresión:

$$a_n = a_i + a_f$$

La cual podemos leer como el término n-esimo es igual a la suma del anterior más el anterior al anterior.

```
#include <stdio.h>
#include<conio.h>
int main(int argc, char *argv[])
{   double n, t, af, ai, an, suma;

    printf("Escriba el número de terminos a crear");
    scanf("%lf",&n);
    af=0;
    ai=1;
    an=0;
    t=1;
    suma=0;
    while (t<=n)
    {       printf("%4.0lf",an);
            suma+=an;af = ai;ai= an;
            an =af + an;
            t+=1;
    }
    printf("\n\n\a La suma vale %5.0lf",suma);
    _getch();
}
```

6 INSTRUCCIÓN DO / WHILE (EXPRESION LÓGICA)

La instrucción do/while la utilizan los programadores para indicarle al computador que debe ejecutar una o varias instrucciones mientras que una condición definida en la expresión lógica sea verdadera, a diferencia del while, la condición se evalúa después de ejecutar las instrucciones a repetir al menos una vez.

```
do
{
        Instrucciones a
        Repetir

}while(expresión lógica);
```

EJEMPLO:

Escriba un programa que lea un número entero n y luego calcule y escriba los n primeros términos de la serie 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.....

Este programa es el mismo que vimos en las páginas del while, observe como se puede realizar lo anterior usando el do/while.

```
#include<stdio.h>
#include<conio.h>
void main()
{  double n, t, af, ai, an, suma;

   printf ("Escriba el numero de termino a crear");
   scanf("%lf",&n);
   af=0;
   ai=1;
   an=0;
   t=1;
   suma=0;
   do
   {       printf("\n\t %lf",an);
           suma=suma + an;
           af = ai;
           ai= an;
           an=af + an;
           t=t+1;
   }while (t<=n);
   printf("\n\n\a La suma vale %lf", suma);
   _getch();
}
```

EJEMPLO:

Escriba un programa en C que lea un número N, luego el computador debe dividir

N por todos los números que hay entre 2 y N/2, hasta cuando encuentre un primer número que divida a N, en ese momento el computador debe hacer que una variable entera tome el valor 1, y debe imprimir el valor que divida a N.

Si no encuentra divisores el computador debe escribir un mensaje que diga que no hay divisores.

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

int main()
{   int    estado, n,divisor;
     char p;


     printf("Escriba el numero a analizar");
     scanf("%d",&n);
     divisor=2;
     estado= 0;
     do {
          if (n % divisor ==0)
          {      estado= 1;
                 printf("%d",divisor);
          }
           else
                 divisor=divisor +1;
     }while ( (divisor<=n/2) && !(estado));
     if (estado==0)
          printf("\nNo hay Divisores");
      _getch();
}
```

La variable estado se vuelve 1 cuando el numero n es divisible por divisor.

7 LA INSTRUCCIÓN FOR

La utilizan los programadores para indicarle al computador que ejecute un conjunto finito de instrucciones mientras una condición dada se cumpla.

El formato se encuentra en casi todos los lenguajes, pero en "C" tiene una potencia y flexibilidad sorprendente.

La sintaxis de la forma general de la sentencia for es:

***for* (inicialización; condición; incremento)**

sentencia_a_ejecutar;

El bucle for permite muchas variaciones, pero existen tres partes principales:

1. La inicialización: Normalmente es una sentencia de asignación que se utiliza para la inicializar una variable de control del bucle.

2. La condición: Es una expresión lógica que determina cuando finaliza el bucle.

3. El incremento: Define como cambia la variable de control cada vez que se repite el bucle.

Las tres secciones se separan por ; (punto y coma) el bucle se repite hasta que la condición sea falsa.

EJEMPLO:

Escriba un programa que calcule el factorial de un número entero n.

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

void main ( )
{
        float factorial = 1, n, factor;

        printf( "Entre un numero al cual le quiere calcular el factorial");
        scanf("%f",&n);
        for (factor = n; factor; factor--)
                factorial*=factor;
   printf("%f!=%f",n, factorial);
   _getche();
}
```

8 EL OPERADOR COMA

La coma en "C" tiene como función encadenar varias expresiones. Esencialmente, la coma produce una secuencia de operaciones. Cuando se usa en la parte derecha de una sentencia de asignación, el valor asignado es el valor de la última expresión de la lista separada por coma.

EJEMPLO:

```
y =100;                    /* y toma el valor de 100*/
x =( y = y-50,100/y);      /* y vale 50 y 100/50 es 2*/
```

tras la ejecución x =2.

Se puede pensar en el operador coma, teniendo el mismo significado que la palabra en español normal "haz esto y esto y esto".

8.1 Variación del ciclo for

Utilizando el operador coma se puede hacer que el ciclo for tenga dos o más variables de control del bucle.

EJEMPLO:

Escribe un programa que lea un número entero n, el programa debe encontrar cuánto suman los n primeros números pares e impares.

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

void main ( )
{  int n, par, sum,imp, sump, sumi, numeros;

   sum=sump=sumi = 0;
   printf("Cual es el valor de n");
   scanf("%d",&n);
   for(numeros = 1, par=2, imp=1; numeros<= n; ++numeros, par+= 2, imp+= 2)
   {  printf("\n\t\t %d par  %d impar",par,imp);
      sump+=par;
      sumi+=imp;
   }
   printf("\n\t Suma impares %d Suma pares %d",sumi,sump);
   _getch();
}
```

9 INSTRUCCIÓN BREAK

Se puede usar para forzar la terminación inmediata de un bucle, saltando la evaluación de la condición normal del ciclo.

Cuando se encuentra la instrucción break dentro de un bucle finaliza inmediatamente, y la ejecución continua con la instrucciones que siguen al ciclo.

10 INSTRUCCIÓN CONTINUE

La sentencia continue, fuerza una nueva iteración del bucle y salta cualquier código que exista entre medios.

```
do          {
    cout<<"Entre un numero igual a 100";
     cin>>x;
     if ( x<0) continue;
              cout<< x;
} while (x!=100);
```

El ciclo anterior repite la lectura mientras x sea negativo.

Para los bucles while y do-while, una sentencia continue hace que: el control del programa no ejecute la prueba del condicional y continue el proceso de iteración.

En el bucle for se ejecuta se ejecuta la parte del incremento del bucle, seguida de la prueba condicional y finalmente hace que el bucle continue.

EJEMPLO:

Escriba un programa que lea un número entero n y determine si n es primo o no. Recuerde que un número es primo si no existen números en el intervalo entre 2 y n/2 que dividan a n.

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

int main()
{    int n, divisor,es;

     printf("Escriba el numero que va analizar?");
     scanf("%d",&n);
     divisor=2; es=1;
     while (divisor<=n/2)
     {  if(n % divisor==0){
           printf("\n %d no es primo",n);
           es=0;
           break;
        }else
           divisor+=1;
     }
        if (es)
           printf("\n %d es primo",n);
        printf("\n Termine presione cualquier tecla");
        _getche();
}
```

La variable es es una variable que funciona como una bandera.

Cambia su valor valor cuando ocurre que se encuentra que algun valor de divisor logro dividir a n

La instrucción break hace que el ciclo termine en el momento en que se encuentra algún divisor de n. Cuando se termina el ciclo el computador chequea si la variable es tiene almacenado un 1 si eso es verdad el número es primo.

11 OBTENCIÓN DE NUMEROS ALEATORIOS

A veces queremos que nuestro programa obtenga números de forma aleatoria, por ejemplo, para simular una tirada de dados o el reparto de cartas en un juego.

11.1 La función rand()

En C, para obtener números aleatorios, tenemos la función rand(). Esta función, cada vez que la llamamos, nos devuelve un número entero aleatorio entre 0 y el RAND_MAX (un número enorme, como de 2 mil millones).

El primer problema que se nos presenta es que no solemos querer un número aleatorio en ese rango, sería un dado muy curioso el que tenga tantas caras.

Podemos querer, por ejemplo, un número aleatorio entre 0 y 10. O de forma más general, entre 0 y N. El calculo que nos permite hacer esto es:

numero = rand() % (N+1);

Si queremos un rango de numeros aleatorios entre M y N con M<N se debe hacer el siguiente cálculo.

numero = ***rand*** () % (N-M+1) + M;

12 INSTRUCCIÓN SWITCH

Esta instrucción permite verificar si una variable de tipo char o int tiene un valor determinado.

```
switch(variable){
case  constante1:          instrucciones1;
                           break;
case  constante2    :      instrucciones2;
                           break;
case  constante3    :      instrucciones3;
                           break;
.....
case  constanten    :      instrucciones3;
                           break;
default             :      Instrucciones por defecto;
}
```

La instrucción switch compara el valor de la variable con la constante1, si son iguales ejecuta las instrucciones1 y llega al break, y ejecuta luego las instrucciones siguientes a la llave }.

Si no son iguales compara el valor de la variable con el valor de la constante2, sino son iguales compara con el de la constante3, y así sucesivamente. Si no existe ninguna constante igual al valor de la variable ejecuta el default, si este existe, sino continúa con la ejecución de las instrucciones después de la llave }. El break en cada caso es opcional, sino existe se ejecutan las instrucciones siguientes hasta encontrar un break.

Ejemplo: Escriba un programa que presente de manera aleatoria sumas o restas a un niño que esta aprendiendo las operaciones básicas de aritmética. El programa debe proponer operadores aleatorios entre 0 y 100, en el momento de ofrecer una resta no debe permitir que el minuendo sea mayor que el sustraendo.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main(int argc, char *argv[])
{  int operando1, operando2, operacion,comp;
   int resEstudiante;

do{
    srand (time(NULL));
    operacion= rand()% (2-1+1)+1;
    printf("\n operacion %d",operacion);
   /* rnd()%(N-M+1)+M con M<N */
   operando1=rand()%(100+1);
   operando2=rand()%(100+1);

   switch (operacion){
    case 1:
            /* Entrenado de Sumas */
            comp=operando1+operando2;
            printf("\n Cuanto da %d + %d =",operando1,operando2);
            scanf("%d",&resEstudiante);
            if(resEstudiante==comp)
               printf("\n Usted acerto");
            else
               printf("\n Usted no acerto");
            break;
    case 2:
            /*Entrenando Restas */
            if (operando1>operando2){
               comp=operando1-operando2;
               printf("\n Cuanto da %d - %d =",operando1,operando2);
            }else{
               comp=operando2-operando1;
               printf("\n Cuanto da %d - %d =",operando2,operando1);
            }
            scanf("%d",&resEstudiante);
            if(resEstudiante==comp)
               printf("\n Usted acerto");
            else
               printf("\n Usted no acerto");
```

```
    }/* Fin del switch */
    printf("\n Presione cualquier tecla diferente a s");
    fflush(stdin);
 }while( getchar()!='s');
  return 0;
}
```

13 VECTORES Y MATRICES EN C:

Los vectores y matrices también conocidos como Arreglos, son variables en memoria que pueden almacenar más de un dato del mismo tipo. Estas variables tienen una característica y es que nos referimos a los elementos, almacenados en los vectores o matrices, utilizando un nombre y números enteros que indican la posición de la celda en el vector o la matriz.

En C++ podemos tener vectores que almacenen datos de tipo: carácter, enteros, reales, punteros, estructuras.

Propiedades:

1. Los datos individuales se llaman elementos.

2. Todos los elementos tienen que pertenecer al mismo tipo de dato.

3. Todos los datos son almacenados en celdas contiguas en la memoria de la computadora, y el subíndice del primer dato es el cero.

4. El nombre del array es una cte. que representa la dirección en memoria que ocupa el primer elemento del vector.

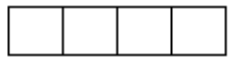
13.1 Declaración de un vector

Para declarar un vector siga la siguiente estructura:

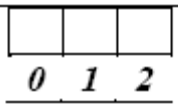
tipo_de_dato nombre[tamaño]

Donde el tamaño es un número entero que indica cuantas posiciones tiene el vector.

float nombre[4];



int datos[3];



Observe en la declaración anterior el número entero n entre los corchetes indica cuantas posiciones tiene el vector en memoria.

Las celdas en memoria se numeran desde 0 hasta n-1.

El vector nombre puede almacenar números de tipo float, el vector datos puede almacenar 3 números enteros.

13.2 Inicialización de arreglos

C permite darle valores a las celdas de un vector de la siguiente manera:

1. Por omisión, cuando son creados. El estándar ANSI de C especifica que siempre que declaremos un vector (arreglo), este se inicializa con ceros.

2. Explícitamente, al declarar el vector suministrando datos iniciales constantes.

3. Durante la ejecución del programa.

En los siguientes renglones les vamos a mostrar como realizar dichas acciones.

Sintaxis:

tipo nombre_del_arreglo[tamaño]={ *lista de valores* }

Lo anterior se llama inicialización Explícita: significa que en el momento de declarar el vector, se especifican los valores así:

```
int    numero[3]={1,2,3};
```

numero		
1	2	3

```
float        horas[ 4 ]={1.2,13.89,23.5,45.8};
```

horas			
1.2	13.89	23.5	45.8
0	1	2	3

13.3 Inicialización en tiempo de ejecución

Significa que el computador ejecuta un ciclo while o for y con la instrucción scanf o cin el programa almacena los valores en las variables.

Ejemplos:

El siguiente ciclo muestra como se pueden almacenar datos en un vector de n posiciones, usando la instrucción cin:

```
for (i=0;i<n;++i)
{
        printf("Ingrese elemento %d ",i);
        scanf("%formato",&vector[i]);
}
```

EJEMPLO:

Escribir un programa que lea un número entero n y el computador lea y almacene

n componentes reales en dos vectores a y b. Luego debe calcular el producto escalar entre a y b y escribir el resultado.

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

int main()
{       double  a[20],b[20];
        double  prodescalar;

        int     i,n;
        do{
                /*Este do while no permite que se usen mas de 20 celdas */
                printf("Cuantas celdas tienen los vectores");
                scanf("%d",&n);
                if (n>20)
                        printf("\nEl vector no puede declararse con mas de 20");

        }while (n>20);

        for (i=0;i<n;++i)
        {
                        printf("\nEntre a[ %d ]=",i);
                        scanf("%lf",&a[i]);
                        printf("\nEntre b[ %d]=",i);
                        scanf("%lf",&b[i]);
        }
        prodescalar=0;
        for (i=0;i<n;++i)
                prodescalar+=b[i]*a[i];
        printf("\n\tVector a");
        for (i=0;i<n;++i)
        {       printf("\n\t %5.2lf ",a[i]);
        }
        printf("\n\tVector b");
        for (i=0;i<n;++i)
          printf("\n\t %5.2lf ",b[i]);

        printf("\n\t Termine presione cualquier tecla, \n\t
        El producto escalar=%lf",prodescalar);
        fflush(stdin);
        getchar();
}
```

13.4 Arreglos bidimensionales

El lenguaje C y C++ permite declarar arreglos de forma matricial. Para declararlos siga la siguiente sintaxis:

```
tipo_de_dato      nombre_arreglo[numero_filas][numero_col];
```

EJEMPLO

```
float      a[3][3];
```

a

	0	1	2
0	34	45	3
1	35	67	45
2	12	34	56

Donde la matriz a tiene 3 filas y tres columnas en la memoria del computador.

13.5 Inicialización de arreglos matricial

C permite darle valores a las celdas de una matriz de la siguiente manera:

1. Por omisión, cuando son creados. El estándar ANSI de C especifica que siempre que declaremos una matriz, esta se inicializa con ceros.

2. Explícitamente, al declarar la matriz suministrando datos iniciales constantes.

3. Durante la ejecución del programa.

En los siguientes renglones les vamos a mostrar como realizar dichas acciones.

Sintaxis:

tipo nombre_del_arreglo[tamaño][tamaño2]={ *lista de valores* }

Lo anterior se conoce con el nombre de inicialización explícita:

Significa que en el momento de declarar la matriz, se especifican los valores así:

```
int   numero[3][3]={    {1,2,3},
                        {4,15,6}
                        {7,8,19} };
```

numero

1	2	3
4	15	6
7	8	19

13.6 Inicialización en tiempo de ejecución

Significa que el computador ejecuta un ciclo while o for y con la instrucción scanf o cin el programa almacena los valores en las variables.

Ejemplos:

El siguiente ciclo muestra como se pueden almacenar datos en una matriz de m filas y n columnas, usando la instrucción scanf:

```
for(i=0;i<n;++i)
        for(j=0;j<m;++j)
        {
                printf("Escriba nombreMatriz[ %d, %d]",i,j);
                scanf("%formato",&nombreMatriz[i][j]);
        }
```

El ciclo anidado anterior entra los datos a una matriz, por filas.

Ejemplo: El siguiente ciclo muestra como se pueden almacenar datos en una matriz de m filas y n columnas de nombre a, usando la instrucción scanf:

```
for(j=0; j<m ;++i)
        for(i=0;i<n;++j)
        {       printf("Escriba nombreMatriz [ %d, %d]",i,j);
                scanf("%formato ",& nombreMatriz [i][j]);

        }
```

El ciclo anidado anterior entra los datos a una matriz, por columnas.

13.7 Escribir una matriz

Para escribir en la pantalla del computador los datos de una matriz que se encuentra en la memoria, siga las siguientes instrucciones:

```
for(i=0;i<m;++i)
{  for(j=0;j<r;++j)
            printf("%formato",nombreMatriz[i][j]);
   printf("\n");
}
```

EJEMPLO:

Escriba un programa que lea dos números enteros m y n, el computador entonces permite entrar m*n elementos a una matriz A y a una matriz B, luego de ello obtiene una matriz C, que es la suma de A y B.

Unidad académica: Ingenierías

Facultad: Facultad de Ingeniería Informática

Profesor: Carlos A. Rodriguez

```
#include <stdio.h>
#define N 10
#define M 10
int main(){
int i,j;
int m,n;
double a[M][N],b[M][N],c[M][N];
printf("\n cuantas filas tienen las matrices a sumar");
scanf("%d",&m);
```

Unidad académica: Ingenierías

Facultad: Facultad de Ingeniería Informática

Profesor: Carlos A. Rodriguez

```
printf("\n cuantas columnas tienen las matrices a sumar");
scanf("%d",&n);

/* las siguientes instrucciones muestran como meter datos a una matriz*/
for(i=0;i<m;i++){
    for(j=0;j<n;j++){
            printf("ingrese el elemento a %d %d:",i,j);
            scanf("%lf",&a[i][j]);
            printf("ingrese el elemento b %d %d:",i,j);
            scanf("%lf",&b[i][j]);
            }
    }
/* la siguiente instruccion muestra como sumar las posiciones */
for(i=0;i<m;i++){
    for(j=0;j<n;j++)
            c[i][j]=a[i][j]+b[i][j];
    }
printf("========== Matriz A ============\n");

for(i=0;i<m;i++){
    for(j=0;j<n;j++){
            printf("\t %5.2lf",a[i][j]);
    }
    printf("\n");
}
printf("========== Matriz B ============\n");
for(i=0;i<m;i++){
    for(j=0;j<n;j++){
            printf("\t %5.2lf",b[i][j]);
    }
    printf("\n");
}
printf("========== Matriz C ============\n");
for(i=0;i<m;i++){
    for(j=0;j<n;j++){
            printf("\t %5.2lf",c[i][j]);
    }
    printf("\n");
}
 getche();
}
```

EJEMPLO:

Escriba un programa que lea tres números enteros m, r y n, el computador entonces permite entrar m*r elementos a una matriz A y r*n elementos a una matriz B, luego de ello obtiene una matriz C, que es la multiplicación entre A y B.

$$c_{ij} = \sum_{k=1}^{r} A_{ik} * B_{kj}$$

$$Con: \quad i = 1 \cdots m$$

$$j = 1 \ldots n$$

Para desarrollar dicho producto se realiza el siguiente ciclo anidado:

```
for(i=0;i<m;++i)
{       for(j=0;j<n;++j)
        {       c[ i ][ j ]=0;
                for(k=0;k<r;++k)
                        c[ i ][ j ]+=a[ i ][ k ]*b[ k ][ j ];
        }
}
```

El siguiente es el código del programa que encuentra el producto de las dos matrices.

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

void main(void)
{       int i,j,m,n,r,k;
        double a[10][10],b[10][10],c[10][10];

   printf("Cuantas filas tiene la matriz A?");
        scanf("%d",&m);
        printf("\nCuantas columnas tiene la matriz A?");
   scanf("%d",&r);
        printf("\nCuantas columnas tiene la matriz B?");
        scanf("%d",&n);
        for(i=0;i<m;++i)
        {       for(j=0;j<r;++j)
                {       printf("\nEscriba el A[ %d %d ]",i,j);
                         scanf("%lf",&a[i][j]);
                }
        }
        for(i=0;i<r;++i)
        {       for(j=0;j<n;++j)
                {
            printf("\nEscriba el B[ %d , %d ]",i,j);
                          scanf("%lf",&b[i][j]);
                }
        }
        for(i=0;i<m;++i)
        {       for(j=0;j<n;++j)
                {       c[i][j]=0;
                        for(k=0;k<r;++k)
                                c[i][j]+=a[i][k]*b[k][j];
                }
        }
        for(i=0;i<m;++i)
        {       for(j=0;j<r;++j)
                        printf("\t %lf",a[i][j]);
                printf("\n");
        }
   printf("\n");
        for(i=0;i<r;++i)
        {       for(j=0;j<n;++j)
                    printf("\t %lf",b[i][j]);
      printf("\n");
        }
```

```
        printf("\n");
        for(i=0;i<m;++i)
        {       for(j=0;j<n;++j)
         printf("\t %lf",c[i][j]);
                printf("\n");
        }
        printf("\n");
   fflush(stdin);
        getchar();
}
```

EJERCICIOS

1. **Sean A y B dos vectores que contienen los elementos numéricos de dos conjuntos A y B. Construya un algoritmo que permita encontrar R que es el vector que contiene los elementos de AÇB.**

Ejemplo:

A=(1,10,-5,-7,8,9,10) B=(7,10,11,3,4,5,8)
R=(10,8)

2. Escriba un programa en lenguaje C que lea los números enteros m y n, luego

permite entrar m*n elementos en una matriz A. Luego el computador encuentra un vector B que almacena el máximo elemento de cada columna de la matriz. Sea una matriz cuadrada NxN, construya un algoritmo que intercambie la fila I con una columna K dadas.

3. Sea una matriz cuadrada NxN. Construya un algoritmo que intercambie los contenidos de dos filas dadas.

4. Sea A una matriz cuadrada de NxN, construya un programa que analice si A es simétrica.

5. Escriba un programa para evaluar la norma infinito de la matriz NxN, que está definida por:

$$||A|| = \underset{i=1}{\overset{N}{\mathrm{MAX}}} [\sum_{j=1}^{N} |A(i,j)|] = Maxima\ suma\ absoluta\ de\ fila$$

6. Escriba un programa que calcule e imprima el triángulo de Pascal.

```
      1
    1   1
  1   2   1
1   3   3   1
```

7. Escribir un programa que permita resolver un sistema de ecuaciones de la forma siguiente:

$$a_{11} * x_1 + a_{12} * x_2 + a_{13} * x_3 + \ldots + a_{1n} * X_n = b_1$$
$$+ a_{22} * x_2 + a_{23} * x_3 + \ldots + a_{2n} * X_n = b_2$$
$$+ a_{33} * x_3 + \ldots + a_{3n} * X_n = b_3$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{nn} * X_n = b_n$$

8. Escribir un programa que calcule e imprima para una matriz de orden NxN los máximos elementos de cada fila y la columna en que están.

9. Escriba un programa que lea M datos para un vector de números reales, una vez hecho esto el programa debe usar procedimientos para calcular el numero de datos repetidos en el vector, el numero de valores impares y el numero de valores pares.

10. Escriba un programa que lea dos vectores, uno A de M elementos, que contiene los códigos de los estudiantes que perdieron matemática, el otro es un vector B que contiene N elementos con los códigos de los que perdieron estática. Se requiere un programa que obtenga un vector C con aquellos estudiantes que perdieron las dos materias.

11. Escriba un programa que lea un vector de M posiciones. El programa debe encontrar el máximo y el mínimo del vector y su posición en el vector.

12. Se tienen dos arreglos unidimensionales. Uno de ellos con N elementos y el otro con M elementos. Dichos elementos se encuentran ordenados de menor a mayor en ambos arreglos. Se pide formar otro arreglo de M+N elementos el cual contendrá los dos arreglos ordenados de menor a mayor.

13. Escriba un programa que lea una matriz cuadrada de números reales. El programa debe encontrar el valor máximo en la columna i, después de eso debe intercambiar la fila donde se encuentre el máximo con la fila que le sigue.

14. Escriba un programa que evalué.

$$\int_a^b f(x)dx = \frac{(b-a)}{3*n}[f(x_0) + 4f(x_1) + \ldots + 4 * f(x_{n-1}) + f(x_n)]$$

15. Los resultados de un examen verdadero o falso hecho a los estudiantes de ciencias de la computación, ha consistido de 10 preguntas falsa o verdaderas. Las respuestas correctas a las 10 preguntas han sido:

0,1,0,0,1,0,0,1,0,1

Escriba un programa que le permita al profesor entrar el código del estudiante y respuestas a cada pregunta. Una vez leídas las respuestas dadas a las preguntas por todos los estudiantes debe calcular para cada estudiante cuantas preguntas correctas e incorrectas tuvo.

El programa debe encontrar el estudiante con más preguntas buenas y asignar luego la calificación a cada estudiante así:

Si el número de buenas es igual al mejor, o el mejor -1 se le da una nota de A

Si el numero de buenas es igual al mejor-2 o al mejor-3 se le da la nota B

El programa debe imprimir el código y las respuestas dadas por cada estudiante, lo mismo que la nota.

16. Se tiene una matriz de 10 filas por 10 columnas. Cada elemento representa en dólares atribuibles a cada uno de los 10 vendedores de la compañía para cada uno de los 10 años de operación que ha tenido. Se requiere calcular e imprimir:

El total de ventas de cada vendedor en los 10 años.

El total de ventas en cada año.

El total de ventas de la compañía en los diez años.

17. Se tiene un vector de 100 elementos con la siguiente información: Cédula, teléfono, dirección y ciudad.

Se desea crear un programa en PASCAL que pida la cédula, la localice en el arreglo y muestre todos los campos para luego

modificar cualquiera de ellos. La operación se hace hasta que se desee terminar.

18. Dada una matriz de MxM elementos se pide:

Calcular la suma de la diagonal principal.

Calcular la suma de los elementos que se encuentren sobre la diagonal izquierda.

Imprimir los resultados de dicha suma.

19. Se tiene un vector de 100 elementos. Realizar un programa que invierta el contenido de este sobre si mismo. Imprimir el vector.

20. Se tiene un arreglo con los datos personales de un estudiante: Código, nombre, teléfono, dirección y dentro del arreglo un vector de 50 elementos con la historia académica de estudiante así: Asignatura, nivel, nota final, nota de habilitación, año y semestre que curso la materia.

Se pide realizar un programa que permita digitar el código del estudiante. Buscarlo dentro del arreglo e imprimir la historia académica en orden ascendente por año de las materias ya cursadas.

21. En un almacén se lleva un control semanal de los artículos que se venden. Al finalizar la semana se llena una tabla con las ventas así:

	Neveras	TV	Estufas	Licuadoras
Lunes	1	2	1	4
Martes	2	3	2	1
Miércoles	1	2	2	1
Jueves	2	2	1	7
Viernes	2	1	2	1
Sabado	1	0	0	0
Domingo	0	0	4	0

Se desea realizar un programa que permita calcular por días el total de artículos que se vendieron y por artículos el total de la semana.

22. Para la evaluación de un pruebas escrita de 100 estudiantes se procede de la siguiente forma:

Cada estudiante se identifica con un código y a cada estudiante se le da una hoja con 50 preguntas y sus respectivas opciones de respuesta. Además, se le entrega una hoja de respuestas, donde debe colocar al frente del número correspondiente la pregunta un número correspondiente a la respuesta según las posibles opciones en la hoja de respuestas.

Cuando el estudiante termina el examen, este es devuelto para ser analizado y dar una calificación final. El trabajo de revisado se hace manual, comparando las respuestas de los estudiantes con una hoja donde están las respuestas correctas a cada una de las preguntas.

El trabajo manual es tedioso, razón por la cual se desea realizar un programa que ayude a comparar los exámenes y al final produzca un resultado relacionando los ganadores de la prueba aparte de los perdedores. Recuerde que la prueba se gana con un 60% de las respuestas correctas. El informe debe salir en orden ascendente por código.

14 CADENAS DE CARACTERES

Para almacenar en la memoria de un computador, una cadena de caracteres, es necesario en el lenguaje C, declarar un arreglo (vector) de caracteres.

Cuando el computador almacena una cadena de caracteres, guarda después del último carácter, el carácter nulo; que se especifica '\0'. Por está razón para declarar arrays de caracteres es necesario que el número de posiciones que se declaren sea de uno más que la cadena más larga que pueda contener. Esto significa que, si se desea almacenar una frase como:

Nacional es campeón, necesitamos declarar un vector así: char frase [20];

En memoria la variable frase almacena el dato así:

N	a	c	i	o	n	a	l		e	S		c	a	m	p	e	ó	n	'\o'

Los vectores de tipo char se pueden inicializar de la siguientes maneras:

14.1 En el momento de declararlos

char nombre de array [tamaño] = "cadena";

char nombre del array[tamaño] = {lista de caracteres }

Ejemplo:

char agua_estado3[8]="Liquido";

de esta manera se añade automáticamente el caracter nulo al final de la cadena.

Ejemplo:

char frase [20] = "Nacional es campeón";

ó

char frase [20] = { 'N','a','c','i','o','n','a','l',' ' ,'e','s',' ','c','a','m','p','e','ó','n','\0' };

Después de la Declaración de la variable:

14.1.1 Por medio de una asignación de cada uno de los caracteres

Si declaramos los siguientes vectores de caracteres:

```
char        agua_estado1[4];
char        agua_estado2[7];
```

Podemos inicializar la variable agua_estado1[4] así:

```
agua_estado1[0] = 'g';
agua_estado1[1]= 'a';
agua_estado1[2]= 's';
agua_estado1[3]= '\0';
```

14.1.2 Por medio de una instrucción de Lectura

Se puede utilizar la función cin, pero no podrá digitar espacios en blanco pues la instrucción cin no los lee. Para poder entrar datos que contenga espacios en blanco utilice la función

cin.get(nombre_vector,longitud);

Donde nombre_vector es la variable donde se va almacenar lo que se digita, longitud es un número entero que indica cuantos elementos se van almacenar.

15 FUNCIONES PARA EL MANEJO DE CADENAS

15.1 Función strcpy()

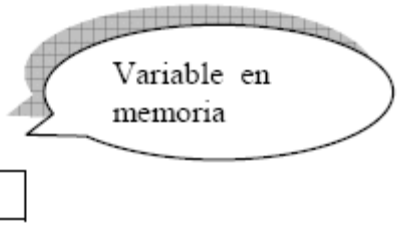
Pertenece a <string.h> copia la cadena apuntada por s2 en la apuntada por s1.

sintaxis:

```
char  *strcpy (char *s1, const char *s2);
```

Ejemplo:
Suponga que se declara una variable así:

```
char frase[ 5];
```



frase

Después de la ejecución de **strcpy**(frase,"Mio");
La variable en memoria es:

M	**i**	**o**	**'\0'**	

En el lenguaje C está prohibido la siguiente asignación:

```
frase="Mio";
```

15.2 Función strcat() :

Pertenece a <string.h> une la cadena apuntada por s2 en la apuntada por s1.

sintaxis:

```
char *strcat ( char *s1, const char *s2);
```

Ejemplo:

Suponga que declara las siguientes dos variables:

```
char frase[25]="La casa", frase2[10]=" es mía";
```

Después de ejecutar la instrucción strcat(frase,frase2); en la memoria del computador la variable frase se puede ver así:

frase

L	A		C	A	s	a		e	S		m	i	a	\0									

15.3 Función strlen():

Devuelve la longitud de una cadena.

Sintaxis:

int strlen (const char *s1);

15.4 Función strcmp (const char *s1, const char *s2)

Compara s1 y s2. Devuelve cero si las dos cadenas son iguales, mayor que cero si la cadena apuntada por s1 es mayor que s2 y menor que cero si la cadena apuntada por s1 es menor que la apuntada por s2.

Ejercicio:

Escriba un programa que lea una palabras. Luego el programa debe invertir la palabra así obtenida e imprimir la palabra que se obtiene al revés.

Ejemplo:

Si la palabra es logroñes debe imprimir: señorgol.

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

void main(void)
{   char frase[50],inversa[50];
```

```
    int longitud,i;

    printf("\t\nEscriba la palabra que desea invertir ");
    gets(frase);
    longitud=strlen(frase);
    for(i=0;i<strlen(frase);++i)
    {   longitud=longitud-1;
            inversa[i]=frase[longitud];
    }
    inversa[strlen(frase)]='\0';
    printf("\n\n\tLa inversa de %s es %s",frase,inversa);
    printf("\t\n\nTermine");
    getch();
}
```

Ejercicio:

Escriba un programa que lea dos cadenas de caracteres y determine cual cadena

es mayor que la otra, el programa debe imprimir la longitud de la cadena mayor.

23. En un vector de 80 elementos se tiene almacenado un párrafo. Cada elemento del

vector indica una línea del párrafo y cada línea tiene máximo 80 caracteres.

Se pide realizar el siguiente menú de opciones:

Mostrar la cantidad de palabras que tiene el párrafo.

Buscar y mostrar el numero de la línea y la palabra que termine en ar, er o ir.

Buscar todas las palabras que tengan una sola letra.

Buscar y mostrar todas las frases que estén entre comillas.

Mostrar la cantidad de caracteres y palabras que contiene el párrafo.

24. Realizar un programa Monto-Escrito, o sea, que permita digitar un valor numérico

máximo de 4 caracteres y al frente aparezca el numero convertido a su equivalente

en texto.

Ejemplo:

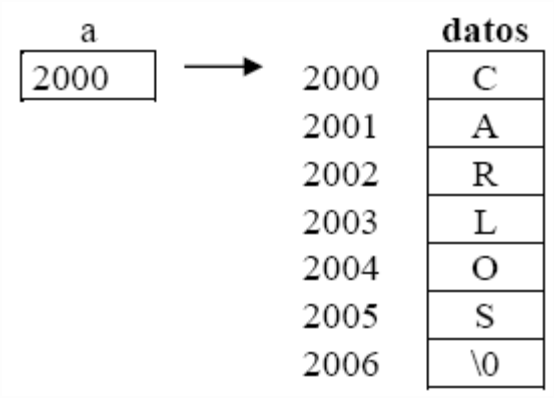
2400 : DOS MIL CUATROCIENTOS

1234 : MIL DOS CIENTOS TREINTA Y CUATRO

16 VARIABLES DE TIPO PUNTERO

16.1 ¿Que es una variable de tipo puntero?

Una variable de tipo puntero es una variable que almacena la dirección en memoria en que inicia una variable.



En el gráfico anterior la variable a es una variable de tipo puntero, pues almacena el número 2000 que es la dirección en memoria donde comienza la variable datos, que es un vector de caracteres con siete posiciones.

16.2 ¿Cómo se declara una variable de tipo puntero?

Para declarar una variable de tipo puntero proceda de la siguiente forma:

Tipo de dato de la variable a la que apunta ***nombre_variable;**

EJEMPLO:

```
int *p;
```

La variable p es una variable puntero a variable entera, en otras palabras, p puede guardar la dirección en memoria en que empiezan variables de tipo entero.

```
char *otra;
```

La variable otra es una variable puntero a variable char, en otras palabras, otra puede guardar la dirección en memoria en que empiezan variables de tipo char.

16.3 ¿Cómo se le da valores a una variable de tipo puntero?

Para almacenar en una variable de tipo puntero la dirección en que está una variable en memoria existen dos métodos:

1. **Utilizando el operador & (operador de dirección);**
2. **2. Utilizando la función malloc()**

16.3.1 Utilizando el operador &

El operador & lo utilizamos en los programas, cuando deseamos almacenar en una variable de tipo puntero la dirección en que se inicia una variable en memoria.

EJEMPLO:

Si en un programa en C se tiene la declaración siguiente:

```
int a,      *b;
b=&a;
```

Lo que ha hecho es asignar a la variable b la dirección de memoria donde se encuentra almacenada la variable a.

EJEMPLO:

El siguiente ejemplo muestra como podemos alcanzar a variar el valor de una variable entera, con el uso de un variable puntero a ella.

16.3.2 El operador *, operador de indirecto o

DESREFERENCIA:

El ejemplo anterior utilizó el operador * de una nueva manera en el lenguaje C, y es muy importante que la entendamos, dicho operador cuando se escribe a la izquierda de una variable de tipo puntero, permite acceder a la variable a la que apunta el puntero.

Analice y reflexione el siguiente ejemplo:

Suponga que se tiene un programa en C así:

```
int santos=1, y=2, z[10];
int          *puntero;
/* La variable puntero guarda la dirección donde está la variable santos en memoria, decimos que el puntero apunta a santos cuando hacemos*/ puntero=&santos;
/*La variable y guarda el valor de la variable Santos 1 */
y=*puntero;

*puntero=0; /* La variable santos ahora guarda 0 */
```

Ejemplo:

El siguiente ejemplo muestra los conceptos básicos de puntero. Se sugiere que el lector lo ejecute en su computador y observe los resultados.

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

int main(int argc, char *argv[])
{   int *p, a;

   printf("Hola dime un numero");
   scanf("%d",&a);
   printf("\nEl tamaño en byte de los enteros %d es",sizeof(int));
   printf("\nEl tamaño en byte de los double %d es",sizeof(double));
   p=&a;
   printf("La direccion donde esta a es %.4X y en esa direccion esta %d ",p,*p);
   getche();
}
```

Ejemplo:

El siguiente ejemplo muestra como se puede acceder a los elementos de un vector con el uso de un puntero.

```
#include <iostream.h>

void main()
{       int     a[5]={168,120,26,-6,7}

        int     i,*puntero;

        puntero=a;
        for (i=0;i<n;i++)
        {       printf("%d",*(puntero++));
        }
        printf("Termine, presione cualquier tecla");
        getchar();
}
```

Explicación, la instrucción puntero=a, asigna en la variable puntero la dirección donde comienza el vector a, es decir la dirección de la celda a[0]. Recuerde que cuando definimos un vector se dijo que el nombre de todo vector es una constante que tiene el valor de la dirección inicial del vector.

```
for (i=0;i<n;i++)
        {       printf("%d",*(puntero++));
}
```

En el ciclo anterior la variable puntero se incrementa en 1 unidad, lo que significa que el puntero va alcanzando las direcciones de las celdas a[0], a[1], a[2], a[3], ... a[n].

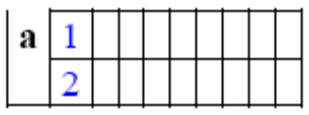
16.3.3 Aritmética de punteros

Recordemos que cuando se declara una variable en un programa en C, lo hacemos para que el computador reserve el espacio suficiente para poder representar los datos en memoria.

Por eso cuando declaramos una variable como:

int a, *puntero;

El computador crea en memoria la variable:



El gráfico muestra que la variable a ocupa en memoria 2 bytes, cada byte en memoria tiene una dirección, podemos decir que la variable a, ocupa los bytes 1 y 2 de la memoria.

Cuando escribimos a=34;

El computador almacena en la variable a, el número 34 en esos dos bytes así:

a	1	0	0	0	0	0	0	0	0
	2	0	0	1	0	0	0	1	0

Cuando en un programa en C, escribimos:

puntero=&a;

El computador almacena en la variable puntero, el número 1 que es la dirección en memoria donde se inicia la variable a.

Pero qué es lo que hace el computador cuando escribimos:

puntero= puntero + 1;

o

puntero+=1;

o

++puntero;

Lo que diríamos es que en la variable puntero ahora se almacena un 2, pues bien, esa es una suposición muy lógica por lo visto hasta aquí, pero es falsa.

Cuando el computador va incrementar en uno el valor de una variable que es un puntero, aumenta la dirección almacenada en el puntero en dos bytes, obteniendo de esta manera la siguiente dirección de una variable donde podría estar un dato de tipo entero.

La regla general es entonces: por cada unidad que se incremente el valor en una variable puntero, se incrementa la dirección en el número de byte que ocupa el tipo de variable en memoria a la cual apunta el puntero.

EJEMPLO:

El siguiente es un ejemplo básico del uso de una variable puntero para poder acceder a un vector.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{   int x[]={12, 30, 5, 6, 7},i;
    int *px;

    px=x;
    *(px+1)=20;
    for (i=0;i<5;i++)
        printf("\n%d",*(px+i));
    getche();
  return 0;
}
```

El código anterior muestra dos conceptos básicos importantes: el nombre de todo vector es una constante que almacena la dirección de existe el vector[3] que nos es más que la dirección de memoria donde esta almacenado la primera celda del vector.

17 INICIALIZACIÓN DE UN PUNTERO A CARACTERES

Una manera más versátil de manejar una cadena de caracteres en memoria es por medio de la declaración de la forma siguiente:

[3] El hecho anterior impide que el nombre del vector sea sometido a la aritmética de punteros.

char *variable="Cadena de caracteres";

La variable se dice que es un puntero a cadena de caracteres:

Ejemplo:

char *frase = "Arroz a la zorra";

Cuando el computador encuentra dicha declaración ocurre lo siguiente:

Frase	1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	r	R	o	z		a		l	a		z	o	r	r	a	\0

El computador crea un espacio en memoria de 17 bytes y guarda allí la frase Arroz a la zorra, determina en que dirección queda almacenado el primer carácter de la frase, luego crea la variable frase y guarda en esa variable la dirección donde quedo almacenado el primer carácter, para nuestro ejemplo la dirección 1.

Ejemplo:

El siguiente ejemplo muestra como escribir al revés una frase que esta referenciada por un puntero.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
int main()
{  char    *palabra="Arroz a la zorra";
   int     indice;

   for(indice=strlen(palabra)-1;indice>=0;indice--)
      putchar(*(palabra+indice));

   getch();
}
```

Ejemplo:

Este programa muestra como llevar los caracteres a un vector de caracteres:

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

#define MAXIMASLETRAS 19

int main()
{ char cadena[MAXIMASLETRAS], *puntero;
  int i,n;


  puntero=cadena;
  printf("\n Cuantas letras va ingresar");
  scanf("%d",&n);

 fflush(stdin);
 for (i=0;i<n;i++)
 {
   printf("\n Ingrese el caracter %d",i);
   *puntero=getchar();
   fflush(stdin);
         ++puntero;
 }
 *puntero='\0';
 printf("Los caracteres leídos son %s al reves son",cadena);
 puntero=cadena+(n-1);
 printf("\n");

 for(i=0;i<n;i++)
 {       putchar(*puntero);
         --puntero;
 }
  getchar();
}
```

18 ASIGNACIÓN DINÁMICA DE MEMORIA

Cuando se compila un programa de C, la memoria de la computadora se divide en 4 zonas que contienen el código del programa, toda la información global, la pila el montón. El montón es un área de memoria libre (a veces denominada almacenamiento libre) y que es utilizada por las funciones malloc y free.

18.1 Función malloc

Cuando utilizamos la función malloc en un programa es porque queremos hacer uso de una variable que será creada en el área de memoria conocido como el montón. La función malloc reserva el espacio en esa área y retorna la dirección donde inicia el espacio de memoria.

Sintaxis:

malloc(tamaño);

Pertenece a la librería stdlib.h

Donde tamaño es un número entero con el cual indicamos cuantos bytes de la memoria del montón deseamos reservar, la función retorna un número que es la dirección en memoria donde se inicia el espacio que reservo, si fracasa intentando reservar el espacio, retorna el valor NULL.

18.2 La Función free

Libera el espacio en memoria utilizado en la memoria del montón y creado por la función malloc. Se utiliza cuando no necesitamos tener en memoria ese espacio.

Sintaxis:

free(nombre de variable puntero);

Una vez se utiliza la función free ésta le asigna al puntero el valor NULL, y deja el espacio a donde apuntaba el puntero libre o disponible.

18.3 La función sizeof

Sirve para que el computador calcule el tamaño de un tipo de dato dado, de acuerdo con la máquina.

Sintaxis:

sizeof(tipo de dato);

Donde tipo de dato es la palabra int, float, long int, char etc.

EJEMPLO:

Universidad Pontificia Bolivariana

Unidad académica: Ingenierías

Facultad: Facultad de Ingeniería Informática

Profesor: Carlos A. Rodriguez

Suponga en un programa en C, quiere calcular el factorial de un número n usando variables creadas con la función malloc.

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

int main()
{   int *p,*i,*factorial;

    p=(int*)malloc(sizeof(int));
    if (p== NULL){
            printf("\n Memoria insuficiente");
            exit(0);
    }else{
    printf("Para que número quiero el factorial");
    scanf("%d",p);
}
factorial=(int *)malloc(sizeof(int));
if (factorial == NULL){
    printf("\nMemoria insuficiente");
    exit(0);
}
i=(int *)malloc(sizeof(int));
if (i== NULL){
    printf("\nMemoria insuficiente");
    exit(0);
}else{
       *factorial=1;
            for (*i=1;(*i)<=(*p);++(*i))
                    *factorial*= *i;
            fflush(stdin);
            printf("\n %d != %d",*p,*factorial);
```

Observe **p** contiene la dirección donde se va almacenar el número

```
}
free(i);
free(p);
free(factorial);
printf("\n Termine presione tecla ");
getch();
}
```

Ejemplo:

El siguiente ejemplo muestra como podemos crear un arreglo de dimensión n en memoria, y como podemos almacenar los n datos en el espacio reservado con la función malloc, para finalmente escribir y sumar las n componentes del vector.

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

```
int main()
{
   int *p,i,n,*suma;
   do{
            printf("\nCuantas posiciones quiere que tenga el vector")
            scanf("%d",&n);
            fflush(stdin);
            p=(int*)malloc(n*sizeof(int));
            if (p== NULL)
            {       printf("\n Memoria insuficiente");
                    exit(0);
            }
            else{
                    printf("\n Listo, pude reservar la memoria");
                    for (i=0;i<n;++i)
                    {       printf("\nEscriba el A[ %d ]",(i+1));
                            scanf("%d",(p+i));
                            fflush(stdin);
                    }
            }
            suma=(int *)malloc(sizeof(int));
            if (suma == NULL){
                    printf("\nMemoria insuficiente");
                    exit(0);
            }
            *suma=0;
            for (i=0;i<n;i++){
                    printf("\t %d",*(p+i));
                    *suma+=*(p+i);
            }
            printf("\nLa suma es= %d ",*suma);
            free(suma);  free(p);
            printf("\n Termine presione tecla n para finalizar ");
   }while (getch()!='n');
}
```

19 FUNCIONES

Las funciones son los bloques constructores de C y el lugar donde se da toda la actividad del programa.

La forma general de una función es:

```
especificador_de_tipo    nombre_función (parámetros formales )
{     declaracion de variables locales

      cuerpo de la función;

      return (valor devuelto);
}
```

especificador_de_tipo: especifica el tipo de valor que devuelve la función cuando ejecuta el proceso, para lo cual fue programada.

En otras palabras indica que tipo de datos sale de la función en el momento de ejecutar el return.

El especificador de tipo puede ser cualquier palabra como:

float, int, char, long int, double, int *, float *, char *; los últimos dos indican que la función retorna un puntero, en otras palabras la función devuelve la dirección de algún objeto en memoria.

Si no se especifica ningún tipo el compilador asume que la función retorna un entero. Si se escribe la palabra void indica que la función no retorna nada, y por tanto en la función no hay expresión return o esta puede ir sola en la función.

Si se escribe void * indica que la función retorna la dirección a un objeto de un tipo cualquiera.

La lista de parámetros formales: está constituida por una lista de nombres de variables con sus tipos asociados, separadas por comas.

Le sirven a la función para recibir información en el momento en que es llamada, los parámetros son variables que toman sus valores en el momento en que la función es llamada desde el main() o desde cualquier función, y en ese momento reciben los valores que la función no puede saber por si misma pero son necesarios para poder realizar la tarea para la cual fue programada.

EJEMPLO:

Piense siempre que vaya a construir una función como si fuese a construir una máquina.

Suponga que va a construir una función que calcula el factorial de un número.

Para que la máquina pueda realizar su tarea, necesita que le entremos el número al cual le vamos a calcular el factorial.

También necesita una salida por donde la función retorna el producto de su tarea, en este caso el factorial calculado. Esos son los parámetros formales y el valor retornado por una función.

Una función puede no tener parámetros en cuyo caso la lista de parámetros contiene sólo la palabra clave void.

Las variables locales: Son todas aquellas variables que son necesarias para que el cuerpo de la función pueda realizar sus tareas.

El cuerpo de la función: Son las instrucciones que le indican a la función como realiza el proceso que debe llevar a cabo.

La sentencia return: Tiene dos usos importantes: primero, fuerza una salida inmediata de la función en que se encuentra, o sea , hace que la ejecución del programa vuelva al lugar del código desde donde se llamó a la función. En segundo lugar, se puede utilizar para devolver un valor.

A continuación se examinan ambos usos.

19.1 Salida de una función

Una función puede terminar su ejecución y volver al sitió en que se le llamo de dos maneras: La primera ocurre cuando se ha ejecutado la última sentencia de la función lo que, conceptualmente, significa que se encuentra la llave } del final de la función.

La mayoría de las funciones emplean la sentencia return para terminar la ejecución, bien, porque se tiene que devolver un valor o bien para simplificar el código de la función y hacerlo más eficiente permitiendo múltiples puntos de salida.

Es bueno tener en cuenta que una función puede tener varios return.

19.2 Valor devuelto

Todas las funciones, excepto aquellas tipo void, devuelven un valor. Este valor se especifica explícitamente en la sentencia return. Si una función no es especificada como void y si no se especifica un valor de vuelta, entonces el valor devuelto por la función queda técnicamente indefinido.

Mientras que una función no se declare como void, puede ser usada como operando en cualquier expresión aritmética valida de C.

Por tanto el valor devuelto en el return es un valor que está almacenado en una variable, o es una constante.

19.3 El uso de los prototipos

Para que una función quede bien declarada en un programa en C se hace necesaria la declaración anticipada de la función. Esta declaración anticipada se denomina prototipo de función.

Los prototipos tienen dos cometidos especiales:

1. **Identificar el tipo de dato que devuelve la función.**
2. **Especificar el tipo y el número de argumentos que utiliza la función.**

19.4 Llamada de una función

Se puede llamar a una función especificando su nombre seguido de una lista de argumentos encerrados entre paréntesis y separados por comas. Si la llamada a la función no requiere ningún argumento, se debe escribir a continuación del nombre de la función un par de paréntesis vacíos. Los parámetros que se utilizan en la llamada normal a una función, se llaman parámetros actuales y debe haber uno por cada parámetro formal.

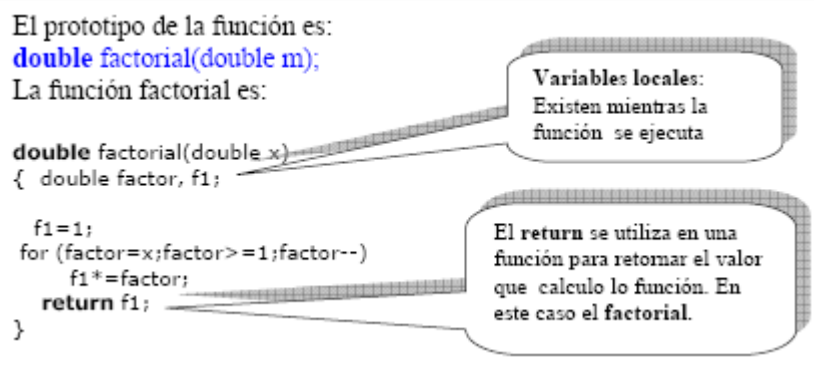
19.5 Paso de parámetros por valor , llamada por referencia

En general, se pueden pasar argumentos a las funciones de dos formas. El primer método se denomina llamada por valor:

En este método se copia el valor que tiene almacenado el parámetro actual (la variable utilizada en la llamada) en el parámetro formal, y los cambios que se hagan en el cuerpo de la función sobre las variables que son parámetros de la función no afectan a las variables que se usan en la llamada.

EJEMPLO:

Escriba una función que reciba por valor un número n y retorne el factorial del número.



El siguiente programa muestra como se utiliza la función factorial para calcular el valor del número combinatorio:

$$\binom{m}{n} = \frac{m!}{(m-n)!*n!}$$

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

double factorial(double m); /* Prototipo de la Funcion Factorial*/

void main()
{
   double combinatoria;
   int m, n;

   printf("Entre el numero de elementos ?");
   scanf("%d",&m);
   printf("Entre de a cuantos va tomar ?");
   scanf("%d",&n);
   fflush(stdin);
   combinatoria=factorial(m)/(factorial(n)*factorial(m-n));
   printf("Combinatoria= %lf",combinatoria);
   getchar();
}

double factorial(double x)
{  double factor, f1;

  f1=1;
   for (factor=x;factor>=1;factor--)
      f1*=factor;
   return f1;
}
```

19.6 El uso de los punteros en los parámetros

Cuando en una función se utiliza un puntero como parámetro formal, en el momento de llamar a la función se copia la dirección de la variable que se utiliza en la llamada a la función en el parámetro que es un puntero. Dentro de la función se usa la dirección que almacena el parámetro, para acceder al dato que se encuentra en dicha dirección. Esto significa que los cambios hechos a los parámetros afectan a las variables usadas en la llamada de la función.

EJEMPLO:

Escriba una función que reciba por valor un número n y retorne el factorial del número.

El prototipo de la función es:

```
void factorial(double m,*double f1 );
```

f1 es un **parámetro formal** que es un puntero, eso significa que al momento de llamar a la función pasamos la dirección de memoria de la variable a la cual se desea acceder desde la función.

La función factorial es:

```
void factorial(double m,*double f1)
{  double factor;

  *f1=1;
   for (factor=x;factor>=1;factor--)
      *f1*=factor;

}
```

*f1=1
Hace que la variable cuya dirección se paso en la llamada a la función setome el valor de 1.

El siguiente programa muestra como se utiliza la función factorial para calcular el valor del número combinatorio:

$$\binom{m}{n} = \frac{m!}{(m-n)!*n!}$$

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

void factorial(double m, double *f1);
/* Prototipo de la Funcion Factorial*/

void main()
{
   double combinatorio,z1,z2,z3;
   int m, n;

   printf("Entre el numero de elementos ?");
   scanf("%d",&m);
   printf("Entre de a cuantos va tomar ?");
   scanf("%d",&n);
   fflush(stdin);
   factorial(m,&z1);
   factorial(n,&z2);
   factorial(m-n,&z3);
   combinatorio=z1/(z2*z3);
   printf("Combinatoria= %lf",combinatorio);
   getchar();
}

void factorial(double x,double *f1)
{  double factor;

  *f1=1;
   for (factor=x;factor>=1;factor--)
      *f1*=factor;
}
```

EJEMPLO:

Escriba una función que reciba la dirección en donde se encuentran dos números enteros y retorna los dos números intercambiados.

```
void swap(int *a, int *b)
{    int temp;

     temp=*a;
     *a=*b;
     *b=temp;
}//Fin de la funcion swap
```

EJEMPLO:

Escriba una función que reciba dos números enteros y retorne el máximo común divisor de los dos números, usando el método de Euclides. La función que calcula el máximo común divisor utiliza la función swap del ejemplo anterior.

```
int mcd(int a, int b)
{   int c;

          if (a<b)
       swap(&a,&b);
          c=a % b;
          while (c!=0)
          {         a=b;
                    b=c;
                    c=a % b;
          }
          return b;
}// Fin de  la funcion mcd
```

El siguiente programa muestra como se pueden utilizar las funciones de los últimos ejemplos para crear un programa que lee cuatro números enteros y calcula el máximo común divisor:

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


void swap(int *a, int *b);
int mcd(int a, int b);

int main()
{
  int a, b, c, d;

  printf("\nEscriba cuatro numeros entero");
  scanf("%d %d %d %d",&a,&b,&c,&d);
  printf("\nEl maximo comun divisor de %d %d %d %d es=%d",
    a,b,c,d, mcd(mcd(a,b),mcd(c,d)));
  getche();
}

void swap(int *a, int *b)
{    int temp;

     temp=*a;
     *a=*b;     *b=temp;
}/*Fin de la funcion swap */

int mcd(int a, int b)
{   int c;
    if (a<b)
       swap(&a,&b);
    c=a % b;
    while (c!=0)
    {       a=b;

            b=c;
            c=a % b;
     }
    return b;
}/* Fin de  la funcion mcd*/
```

Unidad académica: Ingenierías
Facultad: Facultad de Ingeniería Informática
Profesor: Carlos A. Rodriguez

19.7 Paso de Arrays unidimensionales a funciones

Para declarar parámetros que son vectores, se escribe el nombre del vector seguido [ ], no hay necesidad de especificar el tamaño del vector.

Para pasar array unidimensionales a funciones, en la llamada a la función se pone el nombre del array sin índice. Esto pasa la dirección del primer elemento del array a la función. En C no se puede pasar un array completo como argumento a una función; en su lugar, se pasa automáticamente un puntero.

EJEMPLO:

El siguiente programa muestra como una función recibe un vector y la dimensión del vector en sus parámetros y retorna el promedio de sus componentes, también muestra como una función permite ingresar datos a un vector y como una función puede imprimir un vector.

```
#include <stdio.h>

void leeVector(int a[],int n,char t);
void escribeVector(int a[],int n);
double promedio(int x[],int tamano);

int main(){
   int z[5],y[25];

    printf("Hola este tiene funciones dentro\n");
    leeVector(z,8,'p');
    leeVector(y,4,'j');
    escribeVector(z,8);
    printf("El promedio es %lf",promedio(z,8));
    escribeVector(y,4);
    printf("El promedio es %lf",promedio(y,4));
    getche();
}
```

```
double promedio(int x[],int tamano){
      double suma=0;

      int i;

      for(i=0;i<tamano;i++)
         suma+=x[i];
      return (suma/tamano);
}
void leeVector(int a[],int n,char t){
int i;
for(i=0;i<n;i++){
 printf("Entre el elemento %c[ %d ]=",t,i);
  scanf("%d",&a[i]);
}
}
void escribeVector(int a[],int n){
  int i;
  for (i=0;i<n;i++){
     printf("\n%d",a[i]);
  }
}
```

El anterior ejemplo muestra como cuando se usa un array como argumento de una función, se escribe el nombre del arreglo y lo que se pasa es su dirección.

19.8 Paso de Arrays bidimensionales a funciones

Para declarar parámetros que son matrices, solo hay que especificar el nombre de la matriz seguida de [ ][numero de columnas], no hay necesidad de especificar el número de filas.

Para pasar array bidimensionales a funciones, en la llamada a la función se pone el nombre del array sin índices.

EJEMPLO:

El siguiente programa muestra tres funciones, que pasan por referencia matrices. La función leematriz, permite almacenar m*n elementos en una matriz A cualquiera, la función sumaMatrices recibe dos matrices A y B y calcula en una matriz C la suma de A y B.

```
#include <stdio.h>
#define FILAS 10
#define COL 10

void meterDatos(double y[][COL],int m, int n,char x);
void escribirMatriz(double x[][COL],int m, int n,char z[]);
void sumaMatrices(double x[][COL], double y[][COL],double c[][COL],int m, int n);

int main(int argc, char *argv[])
{  double  a[FILAS][COL],b[FILAS][COL],c[FILAS][COL];
   int i,j,m,n;
   m=3;
   n=3;
   meterDatos(a,m,n,'a');
   meterDatos(b,m,n,'b');
   sumaMatrices(a,b,c,m,n);
   escribirMatriz(a,m,n,"Matriz A");
   escribirMatriz(b,m,n,"Matriz B");
   escribirMatriz(c,m,n,"Matriz C");
  getche();
}
void meterDatos(double y[][FILAS],int m, int n,char x){
    int i,j;
    for (i=0;i<m;i++){
        for(j=0;j<n;j++){
            printf("\n\t Ingrese %c [%d,%d]",x,i,j);
            scanf("%lf",&y[i][j]);
        }
    }
}


void escribirMatriz(double y[][COL],int m, int n,char z[]){
    int i,j;
    printf("\n\t %s\n",z);
    for (i=0;i<m;i++){
        for(j=0;j<n;j++){
            printf("\t%lf",y[i][j]);
        }
        printf("\n");
    }
}
void sumaMatrices(double x[][COL], double y[][COL],double c[][COL],int m, int n){
   int i,j;
   for (i=0;i<m;i++){
```

```
        for(j=0;j<n;j++){
            c[i][j]=x[i][j] + y[i][j];
        }
        printf("\n");
    }

}
```

19.9 El uso de los punteros como parámetros para procesar vectores

El siguiente ejemplo muestra una forma alternativa de procesar desde una función un vector.

El programa permite almacenar n elementos en un vector y luego mediante una función convierte las posiciones del vector que son pares en uno y las impares en cero.

Estudie las funciones transformaVector, transformaVector2.

```
#include <stdio.h>

int sumaVector(int x[],int n);
void transformaVector(int x[], int n);
void transformaVector2(int x[], int n,int y[]);
void leerVector(int  *x,int n);
void escribirVector(int  *x,int n);

int main(int argc, char *argv[])
{     int x[10],y[100];
      int n;
        printf("\t Cuantos datos va a entrar ");
        scanf("%d",&n);
        leerVector(x,n);
        printf("Primer x Vector\n");
        escribirVector(x,n);
        transformaVector2(x,n,y);
        printf("El x otra vez\n");
        escribirVector(x,n);
        printf("El y \n");
        escribirVector(y,n);
        getchar();
  return 0;
}
void leerVector(int  *x,int n){
 int i;
    for(i=0;i<n;i++){
           printf("\n entre [%d ]",i);
           scanf("%d",&x[i]);

     }
    fflush(stdin);
}
int sumaVector(int x[],int n){
    int suma,i;

    suma=0;
    for(i=0;i<n;i++)
       suma+=x[i];
}
void transformaVector(int x[], int n){
    int i;

     for (i=0;i<n;i++){
        if (x[i]%2==0)
           x[i]=1;
        else
           x[i]=0;

     }
}



void escribirVector(int  *x,int n){
 int i, j;

    for(i=0;i<n;i++){
          printf("\n%d",x[i]);
    }
}
```

Obsérvese la función leerVector, donde uno de los parámetros en un puntero a entero, nótese como en el cuerpo de dicha función se hace uso del parámetro como un vector, esta es una característica importante entre los vectores y las matrices.

Unidad académica: Ingenierías
Facultad: Facultad de Ingeniería Informática
Profesor: Carlos A. Rodriguez

Por último y para terminar este estudio del uso de los punteros en los parámetros de una función, mostraremos como utilizar de una manera más ágil los punteros, para pasar matrices.

El siguiente ejemplo muestra como pasar una matriz a una función en C.

Ejemplo:

Desarrolle una función que permita ingresar nxn elementos aleatorios a una matriz de orden nxn y la imprime, aquí se muestra el uso de los punteros.

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

#define  M 100
#define  N 100
void leerMatriz(int *x,int n,char nombre);
void escribirMatriz(int *x,int n,char t);

int main(int argc, char *argv[])
{   int x[M][N];
    int n;

    printf("\n Entre el valor de n");
    scanf("%d",&n);
    fflush(stdin);
    leerMatriz(x[0],n,'a');
    escribirMatriz(x[0],n,'a');
    getchar();
  return 0;
}
```

```
void leerMatriz(int *x,int n, char nombre){
    int i,j;

    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            /*
            printf("\n          Ingrese          %c
[%d,%d]",nombre,i,j);
            scanf("%d",(x+i*n+j));*/
            *(x+i*n+j)= rand() % (20+1);
        }
    }
}
void escribirMatriz(int *x,int n,char p){
    int i,j;
    time_t t;

    srand(time(&t));
    printf("\n La matriz %c es:\n",p);
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            printf("%6.2d",*(x+i*n+j));
        }
        printf("\n");
    }
}
```

Analice con su profesor el código que se encuentra resaltado con color azul.

19.10 El concepto de matriz como vector de puntero a punteros

Durante este curso se dijo que una matriz era un área rectangular que tenía un numero m determinado de filas y de n columnas que daban lugar a mxn celdas en las cuales podríamos almacén igual numero de datos del mismo tipo.

Para declararlos podríamos usar la siguiente sintaxis:

```
float a[3][3];
```

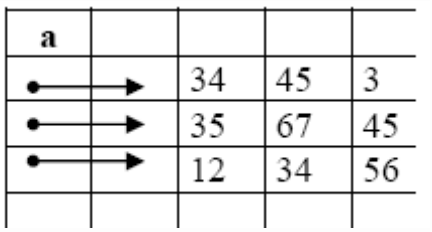
La cual gráficamente la representábamos así. Esa representación no es exacta para describir como C implementa las matrices.

a

	0	1	2
0	34	45	3
1	35	67	45
2	12	34	56

Los autores de C proponen que dada una matriz a de orden mxn, la veamos como un vector de m punteros que apuntan cada uno de ellos a m áreas de memoria que pueden almacenar n datos del tipo del cual se declaro la matriz.

Para el caso de la declaración anterior, el siguiente diagrama es una representación exacta de ese concepto.



El diagrama muestra como cada celda del vector a apunta al área de memoria en que está cada fila de la matriz. Observe como *cada a[i] es un puntero a enteros* y a su vez *a es un puntero a un área de memoria que almacena punteros*, es decir a punta a punteros.

El siguiente código trata de usar los punteros para mostrar este concepto:

```
#include <stdio.h>

int **matrizDinamica(int m,int n);
void entrarDatosMatriz(int **vector,int m,int n);
void escribirDatosMatriz(int **vector,int m,int n);

int main(int argc, char *argv[])
{ int **x,n,m;

  printf("\n De cuantas filas quiere la matriz");
  scanf("%d",&n);
  printf("\n De cuantas columnas quiere la matriz\n");
  scanf("%d",&m);
  x=matrizDinamica(m,n);
  entrarDatosMatriz(x,m,n);
  escribirDatosMatriz(x,m,n);
  free(*x);
  getch();
}
void entrarDatosMatriz(int **vector,int m,int n){
 int i,j;

 for(i=0;i<m;i++){
    for (j=0;j<n;j++){
       /*printf("\nIngrese a[%d,%d]",i,j);
       scanf("%d",&vector[i][j]);*/
       vector[i][j]=rand()%10;
    }
  }

}
void escribirDatosMatriz(int **vector,int m,int
n){
 int i,j;

 for(i=0;i<m;i++){
    for (j=0;j<n;j++){
       printf("\t%d",vector[i][j]);
    }
    printf("\n");
  }

}

int **matrizDinamica(int m,int n){
   int **matriz,i;

  for (i=0;i<m;i++){
     matriz=(int **)malloc(sizeof(int *)*m);
  }
  for (i=0;i<m;i++){
     matriz[i]=(int *)malloc(sizeof(int)*n);
  }
   return matriz;
}
```

Se le deja al estudiante la responsabilidad de discutir con sus compañeros el código anterior.

20 QUÉ DEVUELVE MAIN ( )

Cuando se usa una sentencia return en main( ), el programa devuelve un código de terminación al proceso que lo llamo, en este caso el sistema operativo. Para el sistema operativo un 0 indica que el programa ha terminado normalmente.

Todos los demás valores indican que se ha producido algún tipo de error.

20.1 Argumentos de main()

Turbo C soporta tres argumentos dentro de la función main(). Los dos primeros son: argc y argv, que permiten pasar información al programa de C mediante argumentos de línea de ordenes en el momento de invocar el programa desde el sistema operativo.

El parámetro argc contiene el numero de argumentos de la línea de ordenes y es un numero entero. Siempre vale uno por lo menos, ya que el nombre del programa cuenta como primer argumento.

El parámetro argv es un array de punteros a caracteres. Cada elemento del array apunta a un argumento de la línea de órdenes. Todos los argumentos de la línea de órdenes son cadenas; cualquier número tendrá que ser convertido por el programa al formato de número correcto.

Ejemplo: Escriba este programa en el editor de C.

```
#include<stdio.h>

main(int argc, char * argv[ ])
{   printf("El nombre del programa %s %s",argv[0],argv[1]);
    getchar();
}
```

Grábelo con el nombre mio.c, compile y cuando lo vaya a ejecutar y escriba ./mio.out mio

¿Qué escribe el programa en pantalla?

Ejemplo: Ejecute el siguiente programa.

```
#include<stdio.h>

main(int argc, char * argv[ ])
{   int ver, cuenta;

if (argc<2) {
printf("Debe introducir la cantidad a contar \n");
printf("en la línea de ordenes. Pruebe de nuevo.\n");
getchar();
return 1;
}
if (argc==3 && !strcmp(argv[2], "ver"))
ver=1;
else ver=0;

for (cuenta=atoi(argv[1]); cuenta; --cuenta)
if (ver)
printf("\n%d", cuenta);

printf("%c", '\a');
getchar();
return 0;
}
```

¿Qué hace el programa? Si después de grabarlo con el nombre curioso lo ejecutamos:

```
c:\tc\bin>curioso 13 ver
```

21 ESTRUCTURAS

En C una estructura es una colección de variables que se referencian bajo el mismo nombre. Una estructura proporciona un medio conveniente para mantener junta información relacionada.

Al crear o definir una estructura se forma una plantilla que puede usar el programador para definir una variable de tipo estructura. La variables que conforman la estructura son llamadas (elementos) de la estructura.

21.1 Declaración de estructuras y variables de tipo estructura

Las siguientes son dos maneras de declarar variables de tipo estructura:

```
struct nombreEstructura{
        tipo1    variable1;
        tipo2    variable2;
        .......
        tipon    variablen;
};
```

La declaración anterior se usa en C cuando se quiere definir una estructura: donde *nombreEstructura* es el identificador de la estructura, **struct** es la palabra que indica que se inicia la declaración de una estructura. La declaración de una estructura termina en punto y coma.

Ejemplo:

```
struct dire{
        char    nombre[30];
        char    dirección[30];
        char    ciudad[30];
        int     telefono;
        int     prefijo;
};
```

22 DEFINICIÓN DE VARIABLES DE TIPO ESTRUCTURA

Para definir una variable de tipo estructura se puede proceder en C así:

struct nombreEstructura variable1, variable1, variable2, variable3, ...

Ejemplo:

Struct dire cliente;

La variable en memoria se la puede imaginar así:

CLIENTE

NOMBRE DIRECCION CIUDAD TELEFONO PREFIJO

Del grafo anterior se debe sacar como conclusión que la variable cliente almacena

5 datos diferentes; que son NOMBRE, DIRECCION, CIUDAD, TELEFONO, PREFIJO.

Otra manera de declarar la misma variable es así:

También se pueden declarar variable en la definición de la estructura.

```
struct nombreEstructura{
    tipo1   variable1;
    tipo2   variable2;
    .......
    tipon   variablen;
}variable1, variable2, variable3,.....,variablen;
```

Ejemplo:

El siguiente es un ejemplo de cómo declarar una variable de nombre dir como una estructura directorio:

```
struct directorio{
        char    nombre[30];
        char    dirección[30];
        char    ciudad[30];
        int     telefono;
        int     prefijo;
}dire;
```

22.1 La palabra typedef

Se usa en C para definir sinónimos de tipos de datos definidos antes. Son de uso muy frecuente en C para crear sinónimos para estructuras, que permiten hacer declaraciones de variables de tipo estructura más abreviados.

Para declarar un sinónimo de una estructura se hace así:

```
typedef  struct nombreEstructura sinonimoEstructura;

o así:

typedef struct nombreEstructura{

    tipo1     variable1;
    tipo2     variable2;
    .....
    tipon     variablen;
};
```

Ejemplo:

La siguiente expresión muestra como declarar una variable de tipo estructura usando lo que se ha comentado hasta aquí, sobre el tema:

```
typedef struct dire directorio
```

Hemos declarado que directorio es un sinónimo de estructura dire.

Para definir variables de tipo directorio solo necesitamos escribir:

```
directorio    cliente;
```

EJEMPLO:

Se desea crear una variable de tipo estructura para almacenar un número complejo. Recuerde que un número complejo, tiene parte real, parte imaginaria.

Si entendimos lo anterior, entonces podemos escribir:

```
struct complejos{                    typedef struct {
        double preal;                        double preal;
        double  pimag;                       double  pimag;
  };                                   } complejo;
```

```
typedef struct complejos complejo;

Complejo     complejo1, complejo2;     complejo     complejo1, complejo2;
```

Para definir una variable complejo, escriba ahora:

```
complejo a;
complejo b;
```

22.2 Como se entran datos a un campo de una variable estructura

Para entrar datos a una variable de tipo estructura lo podemos hacer de dos maneras:

Por medio de una instrucción de lectura o por medio de una instrucción de asignación.

22.2.1 Entrando un dato con la instrucción scanf

Para entrar datos con está instrucción proceda así:

```
scanf("%s",&nombre_variable.nombre_campo);
```

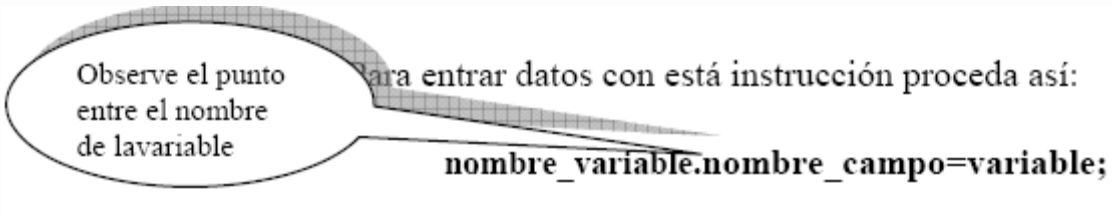
EJEMPLO:

La instrucción:

scanf("%lf",&a.p_real);

Permite entrar un número real a la variable p_real de la variable a

22.2.2 Entrando un dato con la instrucción de asignación



EJEMPLO:

El siguiente programa muestra como podemos utilizar una variable de tipo estructura en un problema de suma de complejos:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   struct complejos{
         double preal;
         double  pimag;
         }complejo1, complejo2, complejoSuma;

         printf("\n Entre la parte real de complejo 1");
          scanf("%lf",&complejo1.preal);
         printf("\n Entre la parte img de complejo 1");
         scanf("%lf",&complejo1.pimag);
         printf("\n Entre la parte real de complejo 2");
         scanf("%lf",&complejo2.preal);
         printf("\n Entre la parte img de complejo 2");
         scanf("%lf",&complejo2.pimag);
          complejoSuma.preal=complejo1.preal+complejo2.preal;
          complejoSuma.pimag=complejo1.pimag+complejo2.pimag;
         printf("\n La suma de ");
         printf("\n %5.2lf + %5.2lf i ", complejo1.preal, complejo1.pimag);
         printf("\n %5.2lf + %5.2lf i", complejo2.preal, complejo2.pimag);
         printf("\n %5.2lf + %5.2lfi",complejoSuma.preal,complejoSuma.pimag);
         fflush(stdin);
         getchar();
         return 0;
}
```

23 ARRAY DE ESTRUCTURAS

Se pueden declarar en C vectores o matrices de estructuras, para declarar variables de ese tipo basta con:

sinonimoEstructura nombre[tamaño];

Donde el tamaño es un número entero que define cuantas celdas tiene el vector.

Ejemplo:

La siguiente declaración define un vector de tipo Parse

```
parse x[10];
```

El siguiente código permite ingresar estructuras a las componentes del vector:

```
printf("\n Cual es le nombre del amigo ");
gets(x[totalAmigos].nombre);
printf("\n Cual es la cedula de %s ",x[totalAmigos].nombre);
scanf("%d",&x[totalAmigos].cedula);
fflush(stdin);
printf("\n Cual es el año de %s ",x[totalAmigos].nombre);
scanf("%lf",&x[totalAmigos].yearNac);
```

El siguiente código permite imprimir los datos que están almacenados en una estructura:

```
printf("\nNombre %s\n Cedula %d \n Año de Nacimiento:%lf", x[i].nombre , x[i].cedula , x[i].yearNac);
```

24 PUNTEROS A ESTRUCTURAS

Un puntero a una estructura es una variable que guarda la dirección de memoria donde se encuentra almacenada una estructura.

Para declarar una variable puntero a estructura se hace lo siguiente:

sinonimoEstructura *nombreVariable;

Ejemplo:

La variable teta es un puntero a una estructura de tipo Parse

```
Parse    *teta;
```

24.1 El operador _

Se utiliza en C para acceder a los miembros de una estructura mediante un puntero.

Si teta es una variable de tipo puntero a una variable de tipo Parse entonces:

teta _miembro1 le permite tener acceso a la información almacenada en el miembro (campo) uno de la estructura.

En los siguientes ejemplos se ilustra el manejo de los punteros a estructuras como parámetros a funciones y como desde ellas se puede acceder:

Ejemplo:

La siguiente función escribe una estructura; para que ella la escriba, recibe un puntero a una estructura:

```
void escribirEstructura(parse *x){
    printf("\nNombre %s\n Cedula %d", x->nombre, x->cedula);
}
```

Observe como se accede a los campos de la estructura.

Para llamar a la función se procede así:

escribirEstructura(&x[i]);

Donde x[i] es un vector que almacena estructuras, &x[i] es la dirección donde se encuentra la estructura.

25 ARCHIVOS EN C

25.1 ¿Qué es un archivo?

Es una estructura de datos que almacena información de manera permanente en un medio magnético.

25.2 Clasificación de los archivos

Los archivos se clasifican de dos maneras, la primera es por la forma en que se almacena la información, la segunda es por la manera en que se tiene acceso a la información en el disco.

25.3 Clasificación por la forma en que se almacena la información

25.3.1 Archivos de Texto

Son aquellos que almacenan la información en código ASCII, en el mundo de la informática también se conocen con el nombre de archivos planos. Un archivo de tipo texto es posible leer su contenido con el bloc de notas.

25.3.2 Archivos Binarios

Son aquellos que almacenan la información en código binario, por el contrario de los archivos texto, la información se puede leer usando el programa con que se creo.

De forma muy simple un archivo binario no es fácil de leer con el bloc de notas.

25.4 Clasificación por la forma en que se accede a la información

25.4.1 Archivos Secuenciales

Son aquellos que exigen que para leer cualquier información en particular hay que leer toda la información grabada antes de la que se quiere leer.

25.4.2 Archivos Aleatorios

Estos por el contrario permiten leer información en el archivo sin necesidad de leer la información anterior. Mediante un puntero se puede situar el computador en un punto especifico del archivo, y allí leer la información especifica y actualizarla.

25.5 Flujos

Un archivo en C cuando se abre se le asocia un flujo, eso significa que C ve el archivo como una secuencia de byte, que al final tiene una marca llamada Marcador de fin de Archivo.

26 ARCHIVOS ALEATORIOS EN C

En los próximos párrafos voy a explicar como se manipulan en un programa en C los archivos de acceso aleatorio.

26.1 El puntero File

Para poder manipular archivos en C, es necesario tener una variable del tipo de estructura File.

Para declarar una variable de tipo File se procede así:

```
File *nombrePunteroFile;
```

Ejemplo

En el código del programa ArchivoAleatorio.c que se anexa al final de este documento, se declara una variable de tipo puntero a una estructura File así.

```
FILE *ptroArchivo;
```

26.2 Apertura de un archivo aleatorio

Para poder tener acceso a un archivo, esto es poder escribir, leer o modificar información, se hace necesario abrir el archivo. Esto se hace posible mediante la función fopen, la cual devuelve un puntero a una estructura File.

El uso más general de la función se presenta en la siguiente sintaxis: ptroArchivo=fopen("nombre del archivo","modo")

Donde el nombre del archivo, es el identificador del archivo en el disco.

El modo es una letra, con la que en C se indica para que se abra el archivo, para los archivos aleatorios se tiene los siguientes modos:

Modo	Cuando se usa
Wb	Se utiliza para abrir un archivo de tipo binario, con acceso aleatorio, este modo se utiliza cuando se quiere crear un archivo para escritura. Si se utiliza esta opción para abrir un archivo que ya existe el archivo se borra y se perderá la información en el.
Ab	Se utiliza para abrir un archivo binario, para agregar información al final del archivo.
Rb	Se utiliza para abrir un archivo para lectura en modo binario.

Después de escribir el código que abre un archivo el programado debe validar que el puntero no tenga el valor NULL, si esto llega a ocurrir durante la ejecución de un programa eso quiere decir que el archivo no ha sido posible abrirlo.

Ejemplo:

El siguiente código muestra como se abre el archivo datosSecuencia.dat, en el programa archivosAleatorios.c y se guarda en el unas estructuras con información de amigos.

En la siguiente instrucción se abre para crearlo y escribir en el archivo por primera vez.

```
ptroArchivo=fopen("C:\\LenguajeC\\datosSecuencia.dat","ab")
```

En la siguiente instrucción se abre para leer la información.

```
ptroArchivo=fopen("C:\\LenguajeC\\datosSecuencia.dat","rb")
```

26.3 Escritura en un archivo aleatorio

Una vez que el archivo ha sido abierto se puede grabar la información al disco, para ello hay que usar la función fwrite. Dicha instrucción se usa de la siguiente manera:

fwrite(*direccionMemoriaVariable*,sizeof(*tipoDedatoagrabar*),cantidad, ptroArchivo);

Donde:
direccionMemoriaVariable: es la dirección donde se encuentra la variable que contiene lo que se quiere grabar en el disco.
tipoDedatoagrabar: es el tipo de dato de la variable que se va a escribir en el disco.
cantidad: es el numero de datos de ese tipo que se van escribir en el archivo

Ejemplo:

El siguiente código muestra como se graba en el archivo datosSecuencia.dat cada una de las estructuras almacenadas en un vector.

```
for(i=0;i<n;i++){
      fwrite(&z[i],sizeof(Parse),1,ptroArchivo);
}
```

26.4 Leer los datos de un archivo aleatorio

Para leer los datos de un archivo de acceso aleatorio se puede hacer de manera secuencial, leyendo uno por uno, o de manera aleatoria.

En los dos casos se usa fread la función tiene la siguiente sintaxis:

fread(*direccionMemoriaVariable,sizeof(tipoDedatoaleer), cantidad, ptroArchivo*);

Donde:

direccionMemoriaVariable: es la dirección donde se encuentra la variable en donde se va a poner lo que se va leer del disco
tipoDedatoagrabar: es el tipo de dato de la información que se va a leer del disco.
cantidad: es el numero de datos de ese tipo que se van leer en el archivo

Ejemplo:

El siguiente código lee de manera secuencia todos los datos en un archivo de acceso aleatorio binario.

```
fread(&z[i],sizeof(Parse),1,ptroArchivo);

do{
   printf("\n%-4d%-
12d%10s%10d",z[i].codigo,z[i].cedula,z[i].nombre,z[i].nacYear);
   i=i+1;
   fread(&z[i],sizeof(Parse),1,ptroArchivo);
}while( !feof(ptroArchivo));
```

La función fread, lee del archivo y almacena lo que va leyendo en un vector de estructuras.

La lectura se hace mientras la función *feof*, no encuentra el final del archivo.

26.4.1 Leer de manera aleatoria un dato en un archivo binario

La lectura secuencia es lenta pues hay que leer todos los datos antes de llegar al que se necesita, para evitar esto, se puede hacer que el puntero apunte a un lugar dentro del archivo donde se encuentra una información especifica, para ello se hace necesario utilizar la función *fseek.*

La función fseek, mueve el puntero de archivo a una posición donde se encuentra un dato específico dentro del archivo.

La función tiene los siguientes parámetros:

```
fseek(ptroArchivo, n*sizeof(tipodeDatoaleer),tipoMovimiento);
```

n: ***Es el numero de datos de un determinado tipo que hay que leer, antes de llegar al que se desea leer.***

tipoMovimiento: ***es una palabra en C que indica desde donde se va hacer la lectura en el archivo esa palabra puede ser:***

tipoMovimiento	Descripción
SEEK_SET	Indica que la búsqueda se debe hacer desde el principio del archivo.

SEEK_CUR	**Indica que la búsqueda se debe hacer desde la posición actual del puntero de archivo.**
SEEK_END	**Indica que la búsqueda se debe hacer desde el final del archivo.**

Ejemplo:

El siguiente código explica como se puede leer una estructura cualquiera en el disco, sabiendo en que posición se encuentra en el archivo aleatorio.

```
fseek(ptroArchivo,n*sizeof(Parse),SEEK_SET);
fread(&t,sizeof(Parse),1,ptroArchivo);
```

26.5 Realizando cambios en un archivo Aleatorio

Para actualizar los datos en un archivo aleatorio, hay que situarse en el punto dentro de un archivo en el cual se va a cambiar la información, y luego se debe escribir la información en ese lugar. Para ello hay que utilizar la instrucción fseek, y luego fwrite como ya se indico.

Ejemplo:

Las siguientes instrucciones cambian la información en un archivo, para una persona especifica:

Unidad académica: Ingenierías

Facultad: Facultad de Ingeniería Informática

Profesor: Carlos A. Rodriguez

```
fseek(ptroArchivo,n*sizeof(Parse),SEEK_SET);
fwrite(&t,sizeof(Parse),1,ptroArchivo);
```

27 BIBLIOGRAFIA

Al Kelley, Ira Pohl. Lenguaje C, Introducción a la Programación, Primera

Edición. México: Addison-Wesley IberoAmerica. 1989. p 392

Sharam, Hekmatpou, C++ Guía para Programadores en C, Primera Edición,

México, Prentice Hall. 1992. p 263.

Osvaldo Cairó, Fundamento de Programación Piensa en C, Primera Edición,

México, Pearson Prentice Hall. 2006. p375.

Deitel M Harvey, Deitel J Paul, Como Programar en C++ y Java, Cuarta Edición,

Pearson Prentice Hall. P 1152