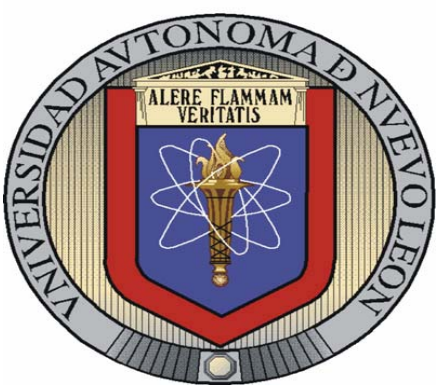


UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

PROGRAMACIÓN BÁSICA

Programación Estructurada con Lenguaje C

http://mx.geocities.com/ciberuniversitario/

Ing. Edgar Danilo Domínguez Vera
Ingeniero Administrador de Sistemas
Maestro en Ciencias de la Administración
con especialidad en Sistemas

Agosto de 2006

Í N D I C E

Aclaraciones:

1) El material de este folleto puede obtenerse en

http://mx.geocities.com/ciberuniversitario/

2) Los nombres de los programas obedecen al capítulo y a la secuencia en el que aparecen, por ejemplo: "**prog3-5.c**" es el quinto programa del capítulo #3.

D E D I C A T O R I A

Durante la vida de las personas, siempre existe el “amor de su vida”, aunque esta persona puede ir cambiando durante las diversas etapas de la vida, por ejemplo: mamá, papá, alguna novia, alguna amante, y aunque cada amor es diferente y se disfruta de diferente manera, sin lugar a dudas, el amor de un hijo no tiene comparación.

Para mi hijo Edgar Belisario y mi esposa Laura Maricela, por su apoyo y comprensión.

Edgar Danilo Domínguez Vera

Agosto del 2006

Capítulo #1
Introducción

1.1. PRESENTACIÓN

Saber programar es una herramienta imprescindible para cualquier profesionista, más en el área de la ingeniería, pues le permite solucionar problemas complejos mediante la realización de cálculos matemáticos que procesan grandes cantidades de datos. En los últimos años ha habido una mayor tendencia al uso de C entre programadores profesionales. Entre las muchas razones de la popularidad de C están las siguientes:

- ✓ C es un lenguaje de programación estructurado, de alto nivel, flexible y orientado a funciones.
- ✓ C tiene ciertas características de bajo nivel de las que sólo se dispone normalmente en ensamblador o en lenguaje máquina.
- ✓ Los programas escritos en C son transformados por los compiladores en pequeños programas objeto que se ejecutan eficientemente.
- ✓ Existen compiladores de C comerciales para la mayoría de las computadoras personales, minicomputadoras y grandes computadoras.
- ✓ C no depende en gran manera de la máquina. Los programas escritos en C se pueden llevar fácilmente de una computadora a otra.

Durante mi estancia como estudiante en la F.I.M.E.-U.A.N.L. aprendí a programar en FORTRAN, BASIC y COBOL, todos estos lenguajes eran no estructurados. Una vez que terminé la carrera y comencé a impartir cátedra, me correspondió enseñar PASCAL, QUICKBASIC y COBOL.

Tanto PASCAL como QUICKBASIC ya incluían características de la programación estructurada; la diferencia radica en que en la programación estructurada las instrucciones se ejecutan de manera secuencial y esta secuencialidad no se ve interrumpida en ningún momento.

Un lenguaje estructurado tiene además instrucciones **repetitivas y selectivas**; todas ellas con un principio y un fin bien claro y definido. La principal instrucción que rompe con la secuencialidad de un programa es la sentencia **goto**, por eso su uso ya ha sido descontinuado.

Ahora, en la F.I.M.E. ya impartimos Lenguaje C, Visual Fox Pro y Progress, todos ellos soportan la programación estructurada, aunque en el caso de Visual Fox Pro y Progress también soportan la Programación Orientada a Objetos.

La realización de este folleto, después de más 10 años de experiencia impartiendo cátedra de Lenguajes de Programación, he visto la necesidad de que el alumno cuente con un cuadernillo de ejercicios. Este folleto no pretende sustituir al libro de texto, simplemente es un apoyo que le permita, al estudiante, visualizar en forma práctica las instrucciones más importantes y utilizadas para comprender y aprender la filosofía de la **Programación Estructurada.**

Hoy, agosto del 2005, después un año de haber trabajado en la Cd. de México en el Comité Administrador del Programa Federal de Construcción de Escuela (CAPFCE), como subgerente de informática me he propuesto en revisar este folleto con el objetivo preparar el terreno para cambiar el plan de estudios de esta materia y adoptar al Java como el lenguaje de programación a enseñar.

Atentamente
I.A.S. Edgar Danilo Domínguez Vera
Maestro en Ciencias de Administración con especialidad en Sistemas
e-mail: danilo_mx@yahoo.com

1.2. DEFINICIÓN DE PROGRAMACIÓN DE COMPUTADORAS.

Es muy difícil dar una definición de lo que es la programación por lo que comenzaré por decir que la programación no es una ciencia exacta, aquí no hay soluciones únicas, un problema puede ser resuelto por programación, sin embargo, puede haber programas diferentes que solucionan el mismo problema.

En la programación no hay reglas estrictas porque no se trata de aplicar una fórmula. Es inútil tratar de aprender a programar por memorización. En la programación hay herramientas y el objetivo es usar esas herramientas con un orden lógico que nos ayuden a resolver un problema.

Mucho se ha discutido acerca de que la programación es un arte. Hay algo de verdad en lo anterior pero también es ciencia porque trata de entender la realidad y procura desarrollar una solución mediante técnicas y métodos ya existentes.

Por lo tanto, podemos decir que **la programación de computadoras es la habilidad de interpretar un problema; llevar ese problema a un nivel de abstracción tal que permita desarrollar un algoritmo de solución que posteriormente pueda convertirse en un código que sea interpretable por una computadora**.

1.3. HISTORIA DEL LENGUAJE "C"

El lenguaje C es un lenguaje de alto nivel que combina el poder de un lenguaje ensamblador con la facilidad de uso y portabilidad de un lenguaje de alto nivel. Fue desarrollado en 1972 en los laboratorios de Bell de AT & T. Fue diseñado por Dennis Ritchie como un lenguaje en el cual se escribió el sistema operativo UNIX. Originalmente fue usado para programar sistemas operativos. Sin embargo, con el transcurso de los años debido a su flexibilidad y su poder, junto con la disponibilidad de compiladores C de alta calidad, para todo tipo de computadoras de todos tamaños, se ha hecho popular en la industria, para una gran cantidad de aplicaciones.

Un lenguaje estándar es el que describe todas las construcciones y especificaciones de su sintaxis para que puedan ser ejecutadas en cualquier computadora.

El lenguaje C estándar usualmente llamado ANSI fue adoptado por el Instituto de Estándares Nacionales Americanos (ANSI) en 1989.

1.4. MÉTODO A SEGUIR PARA LA SOLUCIÓN DE PROBLEMAS

Ya que una computadora no piensa, para que realice un trabajo útil debemos proporcionar una serie de instrucciones, las cuales forman un programa.

Pero programar implica más que una lista de instrucciones. La solución de problemas es un componente crucial de la programación.

Antes de escribir un programa para la solución de un problema en particular, debemos considerar cuidadosamente todos los aspectos del problema y luego, desarrollar y organizar una solución.

Antes de hacer tus propios problemas debes aplicar el siguiente método para la solución del problema propuesto.

1. **Especificar los requerimientos**. Esto consiste en establecer el problema y entenderlo claramente, así como determinar con precisión lo que se requiere para su solución. Se debe descartar todo lo que no es importante y llegar a la raíz del mismo, y si después de esto, el problema no queda totalmente definido debemos pedir más información a la persona que quiere resolverlo.
2. **Análisis**. El análisis consiste en identificar las entradas del problema (datos conocidos), las salidas deseadas (datos que se desean conocer) y cualquier requisito o restricción adicional para lograr la solución. Identificar qué información es proporcionada por los datos del problema y qué resultados se deben computarizar y desplegar.
3. **Diseño**. Lo siguiente es desarrollar una lista de pasos a seguir para la solución del problema llamada ALGORITMO y verificar que el algoritmo resuelva el problema como se intenta. Escribir el algoritmo es la parte más difícil del proceso de solución de problemas.
4. Una vez que se tenga el algoritmo hay que verificar que sea correcto antes de seguir adelante.
5. **Implementación**. Implementar el algoritmo como programa. Se requiere conocer un lenguaje de programación ya que cada paso del algoritmo se convierte a una o varias líneas de código en el lenguaje seleccionado.
6. **Verificación y Prueba**. Probar el trabajo completo y verificar que trabaja como se esperaba usando diferentes conjuntos de datos.

1.5. MÉTODOS PARA LA ELABORACIÓN DE ALGORITMOS.

Hay dos métodos para la elaboración de algoritmos: **Diagramas de flujo y pseudo código**.

Para la elaboración de diagramas de flujo se utilizan una serie de símbolos que representan una acción computacional, tales como entrada de datos, impresión de datos, operaciones matemáticas, selección de alternativas, repetición de pasos, etc.

Es conveniente tomar en cuenta las siguientes **reglas generales para la elaboración de diagramas de flujo**:

1. Utilice símbolos estandarizados
2. Desarrolle el diagrama de flujo de tal forma que se lea de arriba hacia abajo y de izquierda a derecha siempre que sea posible. No cruce líneas de flujo. Use puntas de flechas para indicar dirección.
3. Mantenga el diagrama de flujo claro, legible, simple. Deje suficiente espacio entre los distintos símbolos. Si la solución a un problema es larga y compleja, divídala en varios diagramas de flujo.
4. Escriba mensajes sencillos para describir los distintos pasos a lo largo del diagrama de flujo.
5. Escriba en forma clara y legible los símbolos.

El pseudo código es una especie de lenguaje de programación que permite escribir: entrada de datos, impresión de datos, operaciones matemáticas, selección de alternativas, repetición de pasos etc.

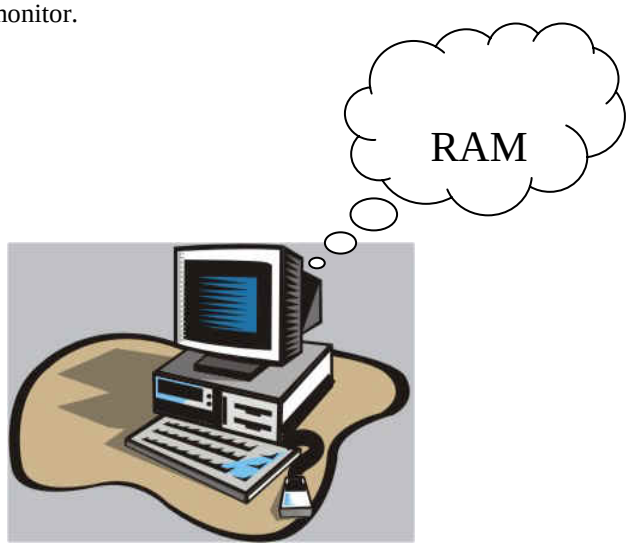
Recuerde que todas las computadoras digitales, independientemente de su tamaño, son básicamente dispositivos electrónicos que pueden transmitir, almacenar y manipular información (datos).

1.6. EJECUCIÓN DE UN PROGRAMA.

La ejecución de un programa supone lo siguiente:

1. Un conjunto de información conocida, los datos de entrada, se introducen en la computadora (desde un teclado, un disquete, un disco duro, etc.) y se almacenan en una porción de la memoria de ésta, tradicionalmente en la memoria RAM.

   Para entender lo que es la memoria RAM observe la figura donde se representa a una computadora que está pensando, de aquí se deduce que la memoria RAM es un medio de almacenamiento de datos volátil ya que cuando se apaga la computadora, toda información que había en la memoria RAM se pierde, a menos de que la hayamos almacenado en otro dispositivo como un disco.
2. Los datos de entrada se procesarán para producir ciertos resultados deseados que se les llama los datos de salida.
3. Los datos de salida, y probablemente algunos de los datos de entrada, se imprimirán en papel o se presentarán en un monitor.



Por otro lado, el procesamiento de datos mediante un programa puede representarse con la siguiente figura.

Se introducen datos a la memoria RAM (desde el teclado), ahí mismo de procesan y posteriormente se genera un resultado (se presentan en el monitor o en papel).

1.7. EL COMPILADOR DE TURBO C.

Un compilador es un traductor de líneas de código de un programa, de tal forma que la computadora "entiende" las instrucciones dadas.

El Compilador de Turbo C maneja dos pantallas, la primera tiene un fondo de color azul y se conoce como **pantalla de edición** como la que aparece en la siguiente figura, y la segunda tiene un fondo de color negro y se conoce como **pantalla de ejecución**.

En la pantalla de edición el programador escribe la codificación de un programa en lenguaje C y en la pantalla de ejecución el programador podrá ver la introducción de datos a la memoria RAM y los resultados que el programa arroja.

El Compilador Turbo C permite detectar los errores de sintaxis que un programa tiene, éstos se corrigen, se graba el programa y se vuelve a compilar; esta acción se repite hasta que el compilador ya no detecte errores de sintaxis, una vez que ya no hay errores, se ejecuta o se corre el programa.

En un programa puede haber **tres tipos de errores**: los de sintaxis, los de ejecución y los de lógica. El compilador únicamente detecta los errores de sintaxis, es decir, aquellos errores que no respetan el formato de las reglas básicas que el lenguaje ordena, es algo parecido a los errores de ortografía, que es cuando una palabra no se escribe según la reglas del idioma en cuestión.

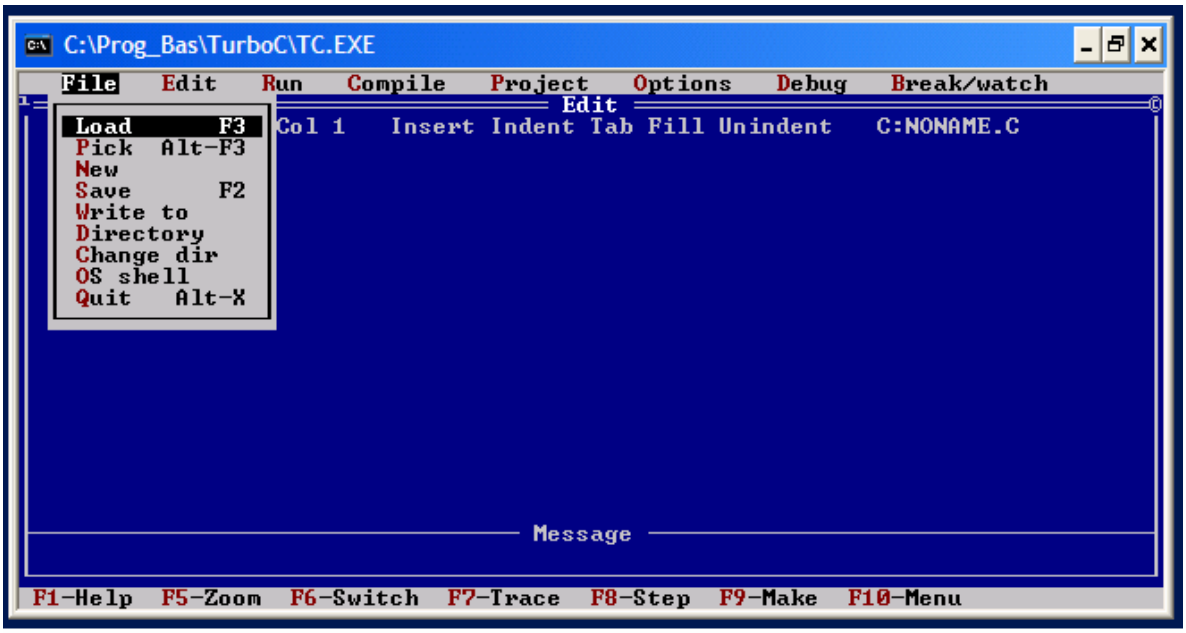
Los errores de ejecución y de lógica se dejarán para otro momento.

La pantalla de edición de Turbo C cuenta con varios menús

1. File
2. Edit
3. Run
4. Compile
5. Project
6. Options
7. Debug
8. Break/Watch

Cada uno de estos menús ofrece una serie de opciones que pueden ser seleccionadas, por ejemplo, tal como se puede ver en la siguiente figura, **File** tiene las siguientes opciones:

1. Load F3
2. Pick Alt-F3
3. New
4. Save F2
5. Write to
6. Directory
7. Change Dir
8. OS Shell
9. Quit Alt-X



Es común que en la pantalla de edición, el programador desee cortar y pegar una o varias líneas de escritura, por lo que se le pide que tenga en cuenta las siguientes recomendaciones.

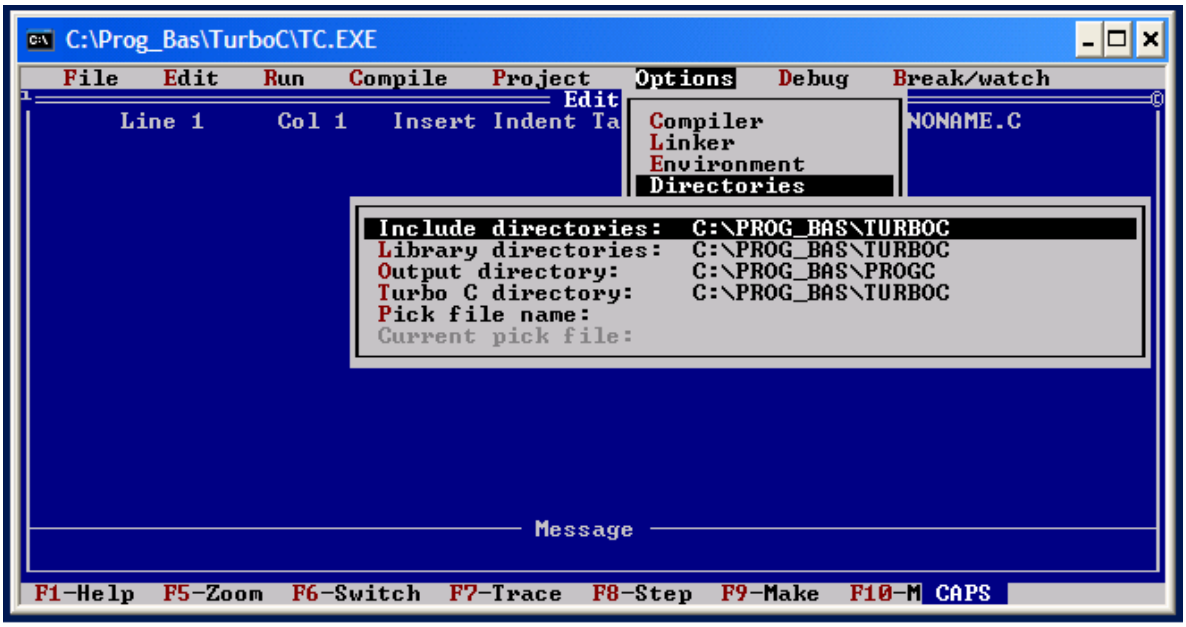
Si se quiere copiar o mover de lugar algunas líneas de código debe seguir estos pasos.

1) Colocar el cursor al inicio de lo que se quiere copiar o mover y dar simultáneamente las teclas <CONTROL> <K> <B>
2) Después, colocar el cursor al final de lo que se quiere copiar o mover y dar simultáneamente las teclas <CONTROL> <K> <K>; al hacer esto, se iluminará el texto; si el texto no se ilumina deberá empezar desde el paso 1.
3) Finalmente, colocar el cursor en el lugar donde se quiere copiar o mover. De esta forma, si se desea **copiar** deberá dar simultáneamente las teclas <CONTROL> <K> <C>, pero si desea **mover** el texto, deberá dar simultáneamente las teclas <CONTROL> <K> <V>.

1.8. CONFIGURACIÓN DEL COMPILADOR DE TURBO C.

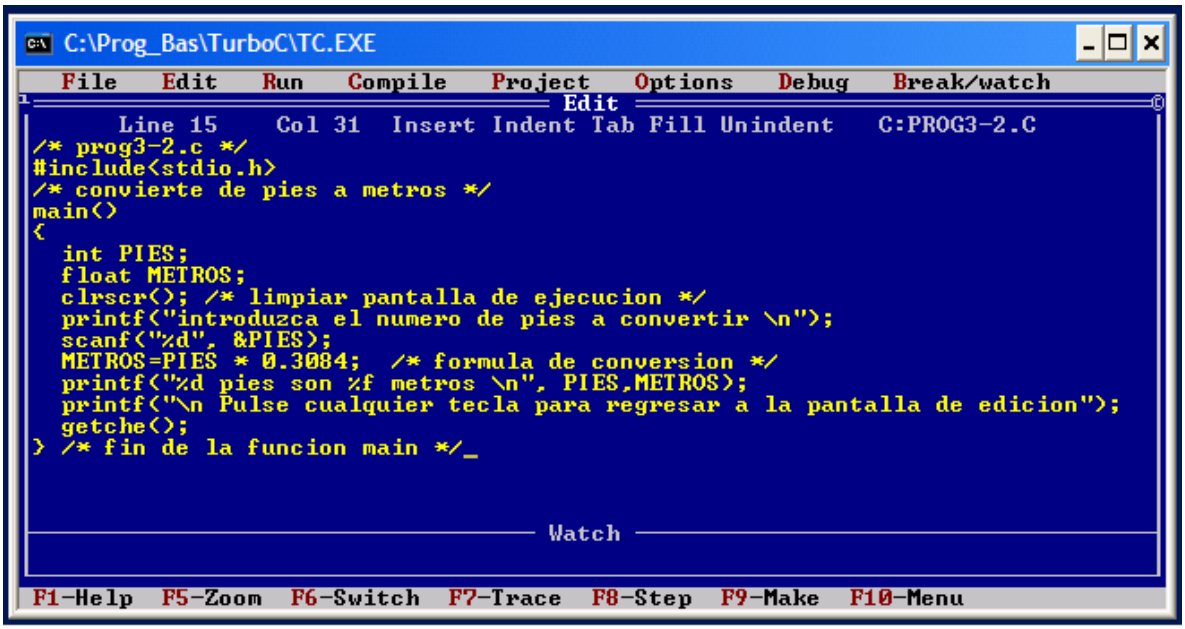
Para que el COMPILADOR de turbo C pueda funcionar adecuadamente se recomienda los siguiente:

1. Crear en el disco “C” una carpeta que se llame “PROG_BAS”.
2. Dentro de la carpeta “PROG_BAS”, haga dos carpetas, una se llamará “TURBOC” y otra se llamará “PROGC”.
3. En la carpeta de que llama “TURBOC” deberá copiar los archivos del editor de textos.
4. En la carpeta que se llama PROGC, ahí almacenará la codificación de sus programas.
5. Posteriormente, en el menú **Options**, en la opción de **Directories**, deberá escribir lo que aparece en la siguiente figura.

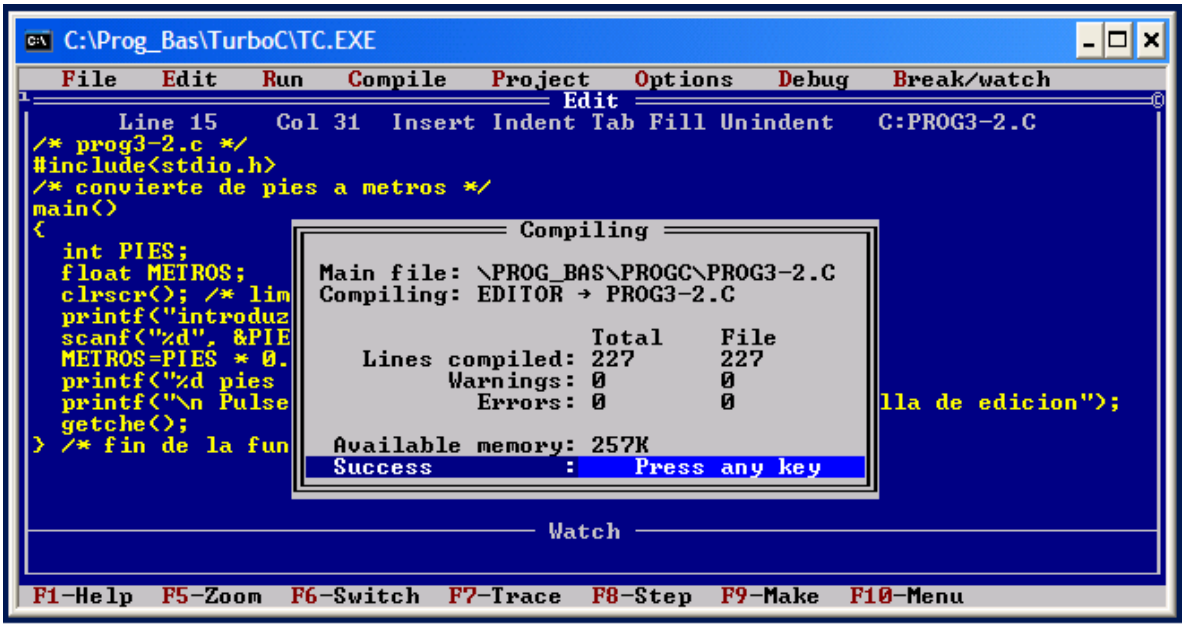


1.9. PASOS PARA EJECUTAR (CORRER) UN PROGRAMA.

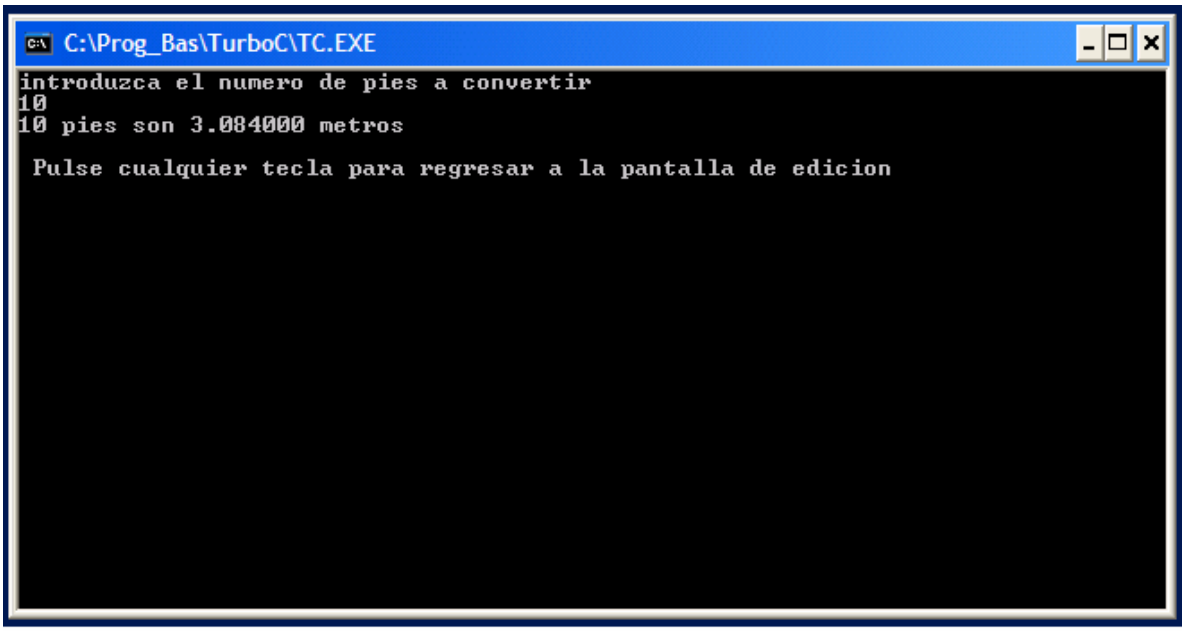
1. Escribir el código en la pantalla de edición.
2. grabar el programa en un disco y asignar un nombre, el cual debe ser de máximo ocho caracteres, sin espacios, acepta letras, números y guiones medios, tal y como se muestra en la siguiente figura.



3. Compilar hasta quitar todos los errores de sintaxis, para esto tendrá que corregir, grabar el programa y volver a compilar hasta que ya no halla errores. Para compilar el programa deberá activar la primera opción que está en el menú **Compile**. Cuando ya no hay errores de sintaxis aparecerá una caja de diálogo como en la siguiente figura



4. Correr el programa y verificar los resultados en la pantalla de ejecución. Para ejecutar el programa deberá activar la primera opción que está en el menú **Run**. Los resultados del programa aparecerán en la ventana de ejecución como en la siguiente figura.



Capítulo #2
Variables, Constantes, Operadores y Expresiones

Las variables se utilizan para expresar una solución generalizada a un problema, por el contrario **las constantes** dan una solución particularizada. Esto es, la siguiente operación:

$$507 + 8\ 9 = 596$$

está compuesta por dos constantes los números 507 y 89, y que por estar en una ecuación matemática se les llama **operandos**, mismos que al sumarse generan una tercer constante que es 596. Además, la operación está compuesta por dos **operadores**, el signo de más y el igual. La anterior **expresión**, es una solución particular que involucra a dos números. Sin embargo, para representar una solución generalizada tendríamos que utilizar variables, entonces tendríamos lo siguiente:

$$C = A + B$$

De esta forma, la **expresión** anterior involucra a **tres variables** (operandos), estas son A, B y C, de esta manera, se representa una solución generalizada ya que se puede dar diversos valores para A y B, generando un valor diferente para C.

En un programa computacional, se hace uso de variables, constantes y operadores. De esta forma, se genera un programa que puede solucionar muchos problemas de un mismo tipo, esto se debe gracias a las variables.

2.1. LAS VARIABLES

Ahora bien, **las variables deben recibir un nombre para poder identificarlas**, obviamente, a estos nombres se les llama identificadores. En Lenguaje C un identificador puede tener de uno a varios caracteres (letras, números o subrayados). **El primer caracter debe ser una letra o un subrayado**. Aquí hay ejemplos de identificadores correctos e incorrectos. No se recomienda que los nombres de identificadores tengan más de 32 caracteres porque lenguaje C ya no reconoce los caracteres posteriores.

Correcto	**Incorrecto**
Contador	8contador
Prueba23	Hola!tu
Alto_balance	Alto..balance

2.1.1. Tipos de Datos

Las variables se deben declarar antes de ser utilizadas en un programa, generalmente al principio del mismo ó de una función y antes de cualquier proposición ejecutable. **Una declaración de variable notifica sus propiedades**. Consta de un nombre de tipo y uno o varios nombres de identificadores.

Los tipos de datos en C son:

Tipo	Descripción	Ancho en bits	Bytes	Rango
char	Carácter	8	1	-128 a 127
int	Entero	16	2	-32,768 a 32,767
float	Real simple precisión	32	4	3.4 E-38 a 3.4E+38
double	Real doble precisión	64	8	1.7E-308 a 1.7E+308
void	Nulo o Vacío	0		Sin valores

a) Se usan variables tipo caracter (**char)** para guardar caracteres ASCII, éstos pueden ser letras, números o caracteres especiales, sin embargo, una variable **char** que contiene números no se puede utilizar en operaciones matemáticas algebraicas.
b) Las variables tipo enteras (**int)** se usan para cantidades numéricas enteras, es decir, que no necesitan decimales, con este tipo de variables se controlan los ciclos repetitivos como **for()** y **do{..}while()**.
c) Se usan variables reales (también llamadas de punto flotante) de simple y de doble precisión (**float** y **double**) para valores numéricos que requieran fracción decimal o cuando la aplicación requiera de valores grandes. La diferencia entre **float** y **double** es la magnitud del número más grande (o más pequeña) que pueden almacenar, esto es, una variable **double** puede almacenar un número aproximadamente diez veces más grande que un **float**.
d) Se usan variables tipo **void** para ayudar a mejorar la comprobación de tipos, se discutirá más adelante.

2.1.2. Modificadores del Tipo de Datos

Excepto para **void,** los tipos básicos tienen varios modificadores que los preceden. Se usa un modificador para alterar el significado de un tipo base y que pueda encajar con las necesidades de diversas situaciones. Los modificadores son:

signed
unsigned
long
short

Se pueden aplicar los modificadores **signed**, **unsigned**, **long** y **short** para los tipos base **char** (caracter) e **int** (entero). Sin embargo, se puede aplicar también **long** a **double.**

Enseguida, se muestra una lista de ejemplo de la forma en cómo se declaran variables.

```
long int matricula;
char nombre[30], hora[3];
float calficacion1, calificacion2, promedio;
int grupo, salon, frecuencia_semana;
```

En el ejemplo anterior tenemos la variable **matricula** como larga entera (**long int**), es decir, esta variable no acepta decimales porque su tipo base es entera (**int**), pero además se le agregó el modificador del tipo de datos de largo (**long**). Con esto es posible dar valores con 6 o 7 dígitos. Si la variable **matricula** fuera solamente entera (**int**), no podríamos dar valores grandes porque estaríamos fuera de rango (-32,768 a 32,767).

Tenemos la variable **nombre** de tipo caracter (**char**) con una longitud de 30, esto implica que podemos manejar nombres con máximo 30 posiciones, contando letras y espacios en blanco, aunque es conveniente utilizar sólo 29 de ellos para dejar un caracter de control propia de lenguaje C. Posteriormente se explicará su uso.

Dado que la variable **hora** aceptará valores como M1, M2,M3…V1,V2..N1,N6, entonces es de tipo caracter con tres posiciones, dejando la última posición para el caracter de control.

La variables **calificacion1**, **calificacion2** y **promedio** pueden presentarse con decimales, por eso se declararon como reales (**float**).

La variable **grupo** aceptará valores como 01,02,03…15 o más pero en ningún momento aceptará valores mayores a 32,767 por lo tanto es de tipo entero (**int**).

La variable **salon** recibirá valores como 1101, 1102, 2101, 2202, 3101…etc. y por el mismo motivo que la anterior variable, es de tipo entero (**int**).

Finalmente, la variable **frecuencia_semana** también es entera e indica cuántas horas de clase de esa materia se imparten por semana, por ejemplo, Programación Básica y Programación Avanzada I y II tienen una frecuencia de 3 horas-clase por semana, hay otras materias que serán de 2 ó 5 horas-clase, pero ninguna se imparte 2.5 o 3.40 horas-clase por semana, es decir, no hay fracciones de hora.

2.1.3 Turbo C es sensible al Tamaño de las Letras

Turbo C es sensible al tamaño de las letras, es decir, en el ejemplo anterior tenemos a la variable **matricula**, escrita con letras minúsculas, sin embargo, turbo C no es capaz de reconocer a **Matricula** o **MATRICULA** como la misma variable, de este modo, se debe tener cuidado que durante toda la codificación de un programa, la variable se utilice de la misma forma de cómo fue declarada.

2.2. LAS CONSTANTES

En Turbo C las constantes se refieren a los valores fijos que el programa no puede cambiar; ejemplos de constantes en la solución de un problema hay muchos, tal como: la gravedad (9.8 m/s^2 ó 32 ft/s^2), el valor de PI (3.1416), etc. De esta forma, las constantes pueden ser de cualquier tipo básico. Las constantes caracter están encerradas entre comillas simples. Por ejemplo, ‘a’, ‘%’. Las constantes enteras son números sin componente decimal y las constantes reales requieren el uso del punto y la fracción decimal.

Tipo de dato	Constante de Ejemplo
char	‘a’, ‘\n’, ‘9’
int	1, 123, 2100,-234
long int	37950, -51452
short int	10, -12, 90
unsigned int	10000, 987, 17953
float	123.23, 4.3e-3
double	123.23, 12312333, -0.9876324, 1.0E100

Las constantes dentro de un programa en Lenguaje C se declaran con la instrucción **#define** seguido del nombre de la constante en mayúscula y el valor que tomará dicha constante.

Sintaxis #define nombre valor

Ejemplos: #define MILLAS 0.62137
#define PI 3.1416
#define MAX_LONGITUD 100

2.2.1. Constantes Hexadecimales y Octales

Lenguaje C permite especificar constantes enteras en hexadecimal u octal en vez de decimal. Una constante en hexadecimal debe comenzar por **0x** (un cero seguido por una equis). Una constante octal empieza con cero. Estos son algunos ejemplos:

```
int hex=0xFF
int oct=011
```

La primer constante hexadecimal es equivalente a 255 en decimal y la segunda constante en octal equivale a 9 decimal.

2.2.2. Constantes de Cadena

Turbo C soporta otro tipo de constante además de las de tipos de datos predefinidos: **la cadena de caracteres**. Una cadena es un conjunto de caracteres que se encierran entre comillas dobles. Por ejemplo, “**esto es una prueba**”. **No se debe confundir las cadenas con los caracter**, esto es, una constante caracter se encierra en comillas simples: ‘a’, sin embargo, “a” es una cadena que contiene una sola letra.

2.2.3. Constantes de Caracteres de Diagonal Invertida

El uso de comillas simples para encerrar todas las constantes de caracteres funciona para la mayoría de los caracteres imprimibles, pero algunos, como retorno de carro es imposible introducir por teclado. Por esta razón , Lenguaje C proporciona las constantes especiales de caracter de diagonal invertida

Código	Significado
\b	Retroceso
\f	Alimentación de hoja
\n	Nueva línea
\r	Retorno de carro
\t	Tabulador horizontal
\"	Doble comilla
\'	Simple comilla
\0	Nulo
\\	Diagonal invertida
\v	Tabulador vertical
\a	Alerta
\N	Constante octal (N es constante octal)
\x	Constante hexadecimal

Nota: La constante **\n** significa avance de línea y retorno de carro, en pocas palabras, es la tecla <ENTER>.

2.3. OPERADORES

Un operador es un símbolo que le dice a la computadora que realice manipulaciones matemáticas o lógicas específicas. **Lenguaje C tiene tres clases generales de operadores: aritméticos, relacionales y lógicos, y sobre bits**. Además tiene algunos operadores especiales para tareas particulares

2.3.1. Operadores Aritméticos

Los operadores aritméticos son:

Operador	Acción
-	Resta
+	Suma
*	Multiplicación
/	División
%	Módulo división (residuo)
--	Decremento. Menos unario
++	Incremento. Más unario.

Estos operadores se pueden subdividir en **unarios** (++, - -) y **binarios** (+,-,*,/, %)

Hay dos operadores binarios que se refieren a una división: **/ y %**, sin embargo, cuando se aplica la división (/) a enteros o caracter el resultado se trunca, por ejemplo, 10/3 será igual a 3 en la división entera.

El operador módulo de **la división (%) recoge el resto (residuo)** de una división entera y tiene como restricción que no se puede usar sobre tipos **float** o **double.**

El operador + + añade un uno a su operando y, - - le resta uno, por lo tanto las siguientes operaciones son equivalentes.

X + + es equivalente a X = X + 1
X - - es equivalente a X = X –1

Ahora bien, estos operadores pueden preceder o seguir al operando, es decir

X=X+1 se puede dar como X++ o ++X

En el primer caso se denomina posincremento y en el segundo caso se denomina preincremento, de esta manera si tenemos:

```
X = 10
Y = ++X
```

El valor de Y sería 11, porque C incrementa primero X y después lo asigna a Y, pero si tuviéramos

```
X = 10
Y = X + +
```

El valor de Y sería 10 y después se incrementaría X. En ambos casos, X estaría a 11, la diferencia está en el momento en que se hace, sí antes o después de la asignación.

2.3.1.1 Reglas para la Evaluación de Expresiones Aritméticas

1. Todas las subexpresiones deben ser evaluadas por separado. Las subexpresiones dentro de paréntesis tienen prioridad por encima de todas las demás.
2. **La regla de prioridad de los operadores** dice que los operadores en la misma subexpresión son evaluados en el siguiente orden
   unario ++, - - primero
   binario *, /, % enseguida
   binario +, - al último
3. La regla de la asociatividad dice que **los operadores unarios** en la misma subexpresión y en el mismo nivel de prioridad, tales como ++ y - -, **son evaluados de derecha a izquierda** (asociatividad a la derecha). Por otro lado, **los mismos operadores binarios** en la misma subexpresión y en el mismo nivel de prioridad, tales como + y -, **son evaluados de izquierda a derecha** (asociatividad a la izquierda).
4. **Para alterar los niveles de prioridad** se pueden usar paréntesis. De esta forma, las subexpresiones dentro de paréntesis anidados se evalúan de las más interna a la más externa

2.3.2. Operadores Relacionales y Lógicos

Los operadores relacionales se refieren a las relaciones que los valores pueden tener unos con otros, y **los lógicos** se refieren a la manera en que tienen lugar estas relaciones. La clave de los conceptos relacionales y lógicos es la idea de falso y verdadero. **En Lenguaje C, verdad es cualquier valor distinto de cero, mientras que cero es falso**. De esta forma, **las expresiones que usan operadores relacionales y lógicos devolverán 0 para falso y 1 para verdadero**.

Se usan los operadores relacionales para determinar las relaciones de una cantidad con otra. Siempre devuelven 1 ó 0, dependiendo del resultado de la prueba.

A Continuación tenemos los operadores a los que nos hemos referido

Operadores Relacionales

Operador	Acción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
= =	Igual
! =	No igual

Operadores Lógicos

Operador	Acción
&&	AND
||	OR
!	NOT

Los operadores lógicos se usan para soportar las operaciones básicas de AND, OR, NOT de acuerdo con la siguiente tabla de verdad que usa 1 para verdad y 0 para falso.

p	Q	p AND q	p OR q	NOT p
0	0	0	0	1
0	1	0	1	1
1	1	1	1	0
1	0	0	1	0

Los operadores relacionales y lógicos tienen menor prioridad que los aritméticos. Esto significa que C evalúa una expresión como **10 > 1 + 12** como si se escribiera **10 > (1+12)** , el resultado de esta expresión es cero, es decir, falso. Así, 5==1 generará un cero que es falso, y 5>1 generará un uno, que es verdadero.

Lenguaje C permite combinar varias operaciones en una expresión como se muestra aquí:
10 > 5 && ! (10 < 9) || 3<= 4, esta expresión será verdad.
solución: 10>5 =1; 10<9=0 pero cambia a 1 por ! y finalmente 3<=4 = 1, entonces tenemos que:
1 && 1 || 1 que equivale a verdad

La prioridad relativa de los operadores relacionales y lógicos es la siguiente:

Más alto ↓ Más bajo

	!
	>, >=, <, <=
	= =, !=
	||, &&

2.3.3. Operador de Asignación

El operador de asignación es el signo igual (=), este operador es el que se utiliza para referirse a fórmulas matemáticas, no debe confundirse con el doble igual (= =) el cual es un operador relacional.

Es conveniente recordar la diferencia entre una fórmula y una ecuación matemática. Una fórmula es aquella en cual una variable está completamente despejada, por el contrario, una ecuación puede tener operadores y operandos en ambos lados de la igualdad.

Ejemplo de fórmulas matemáticas

Fuerza = masa * aceleracion
Area = base * altura / 2
Volumen = alto *largo * ancho

Ejemplo de ecuaciones matemáticas

$3x^2 + 3y = 2k + 5$
$C^2 = A^2 + B^2$
$Vf^2 = 2*gravedad*altura$

En Lenguaje C cuando se trata de fórmulas matemáticas, antes de un operador de asignación debe haber una variable completamente despejada y no puede haber operadores de ningún tipo, porque entonces ya no sería una fórmula sino una ecuación matemática.

2.4. EXPRESIONES ARITMÉTICAS

Los operadores, constantes y variables constituyen las expresiones aritméticas ó fórmulas. Una expresión en lenguaje C es cualquier combinación válida de estas piezas.

Cuando se mezclan constantes y variables de diferentes tipos en una expresión, lenguaje C las convierte al mismo tipo. El compilador convertirá todos los operandos al tipo de operando más grande según la base de esta operación, tal como se describe en estas reglas de conversión de tipos.

a) Todos los **char** y **short int** se convierten a **int**. Todos los **float** a **double.**
b) Para todo par de operandos, lo siguiente ocurre en secuencia. Si uno de los operandos es un **long double**, el otro se convierte a **long double.** Si uno de los operandos es **double**, el otro se convierte a **double.** Si uno de los operandos es **long**, el otro se convierte a **long**. Si uno de los operandos es **unsigned**, el otro se convierte en **unsigned.**

Se puede forzar a una expresión a ser de un tipo específico usando la construcción llamada ***cast*** . El formato general de un ***cast*** es:

(tipo de dato) expresión

donde *tipo de dato* es uno de los estándar de datos de lenguaje C. Por ejemplo si X es un entero y se quiere uno asegurar que la computadora evaluaría las expresión X/2 como de tipo **float** para garantizar un componente fraccional, se escribiría

(float) X / 2

Aquí el *cast* (**float**) se asocia con X, que provoca que la computadora convierta al **2** a **float** también y el resultado será **float**. Sin embargo, se debe ser cuidadoso si se trata de escribir el ***cast*** de esta forma

(float) (X/2)

En este caso, la computadora trae una división entera y convierte el resultado de esa a **float.**

Los ***cast*** son útiles cuando la variable que controla un ciclo, como **for()** la cual debe ser forzosamente entera, se requiere para realizar una operación de tipo **float**, por ejemplo:

```
int I;
float K;
for(I=1; I<=100; I++)
{
  K = (float) I/3;
}
```

2.5. TRANSFORMACIÓN DE FÓRMULAS ALGEBRAICAS A FÓRMULAS COMPUTACIONALES EN "C".

Fórmula	En C se escribe
D=A+B+C	D=A+B+C
A=BC	A=B*C
$C=\frac{AB}{D}$	C=A*B/D
$C=\frac{A}{BD}$	C=A/(B*D)

$m=\frac{y-b}{x-c}$ m=(y-b)/(x-c)

$d = b^2 - 4ac$ d =b*b-4*a*c o d=pow(b,2)-4*a*c

$K=\frac{1}{1+\sqrt{X}}$ K=1/(1+sqrt(X))

Capítulo #3
Operaciones de Entrada-Salida de Datos

3.1. ESTRUCTURA GENERAL DE UN PROGRAMA EN LENGUAJE 'C'

Antes de seguir adelante, daremos un panorama general de la estructura de un programa en lenguaje C. La primera parte de un programa en C son las **directivas del preprocesador** que empiezan con el símbolo "**#**" y que son utilizadas por el preprocesador para que el compilador sepa que son instrucciones que debe obtener de las bibliotecas o archivos que tiene lenguaje C, listos para usarlos, tales como **<stdio.h>** que proporciona a C las bibliotecas estándar de entrada y salida.

La directiva de preprocesador **<stdio.h>** va a ser una de las directivas que va a estar presente en la mayoría de nuestros programas, generalmente al principio de éstos, su significado es **standard input-output**, y permite que un programa en lenguaje C utilice los **estándares de entrada-salida** de la computadora que son: el monitor y el teclado, entre otros.

La siguiente parte del programa es **la declaración de variables y constantes**. En esta sección se establecen **los nombres de los identificadores así como su tipo de datos y en algunos casos su longitud**.

En la última sección del programa se escriben las instrucciones, indicando paso a paso el algoritmo definido para dar solución a un problema. Cabe recordar que lenguaje C es un lenguaje orientado a funciones, donde la función principal es llamada **main()** (principal), **esta función estará presente en todos los programas que se realicen**.

Enseguida, se dará un pequeño programa de ejemplo y se le pide al alumno que "corra ó ejecute" el programa.

```
 /* prog3-1.c */
#include <stdio.h>
main()
{
   int edad;
   edad=37;
   printf("Mi edad es de %d años \n",edad);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

El anterior programa se desglosa de la siguiente manera

a) la primer línea **/* prog3-1.c */** es un comentario. Los comentarios sirven en un programa para ayudar, a una persona que vea el código, a entenderlo. Antes del comentario se coloca "**/*** ", enseguida se escribe el comentario, y al final del mismo se coloca "*/". Los comentarios pueden ir en cualquier parte de un programa y no tienen ejecución alguna. En este caso particular, el comentario de este programa nos ayuda a saber que el nombre del programa es **prog3-1.c**.
b) La segunda línea **#include <stdio.h>**, es la directiva del preprocesador que ya explicamos en párrafos anteriores.
c) La tercer y cuarta línea, **main()** y **{** , indican el inicio del programa..
d) La quinta línea **int edad;** , indica que se declara una variable que se utilizará en el programa, esta variable se llama edad y es de tipo entero, es decir, no aceptará números decimales.
e) La sexta línea **edad=37;** , indica que a la variable edad se le asigna el valor de 37, es decir, se le da el valor de 37, esto dentro de la memoria RAM.

f) La séptima línea **printf("Mi edad es de %d años \n",edad);**. **printf()** es una función que envía datos, desde la memoria RAM, a la pantalla de la computadora. El mensaje, que está encerrado entre comillas, aparecerá en el monitor de la computadora, el **%d** es un especificador de formato que permite imprimir valores enteros. El **\n** es una constante de caracter de diagonal invertida que indica nueva línea

g) La octava línea **getche()**, es una función que en este caso se utiliza para detener la pantalla de ejecución hasta que pulse alguna tecla y así poder ver los resultados del programa. De esta manera, los resultados se verán en la pantalla de ejecución. Si no se utiliza, el programa cambiará automáticamente de la pantalla de ejecución a la pantalla de edición y el usuario no verá el resultado del programa a menos que pulse simultáneamente las teclas <ALT><F5>.

h) La última línea **" } /* fin de la función main() */ "**, la llave "}" indica el final de la función **main()**, en este caso particular indica también, el fin de la codificación del programa, y el comentario es para recordar que es el fin de la función **main()**.

i) Todas las líneas que terminan con punto y coma son instrucciones y/o funciones de lenguaje C, sin embargo, se debe tener cuidado porque no todas las líneas de código llevan punto y coma.

3.2. ESPECIFICADORES DE FORMATO

Para poder visualizar los valores de las variables se deben utilizar funciones de salida de datos como el **printf()**, éste debe ir acompañado de especificadores de formato los cuales varían según el tipo de datos que se desee visualizar.

Código	Tipo de dato que visualiza
%d	Decimal (entero).
%i	Decimal.
%f	Real o punto flotante de simple precisión.
%c	Un solo caracter.
%s	Cadena de Caracteres.
%e	Notación científica.
%g	Usa %e o %f, la que sea más corta.
%ld	Long o largas enteras.
%u	Decimal sin signo.
%o	Octal.
%p	Puntero.
%%	Imprime un signo %.
%n	El argumento asociado será un puntero entero en el que pondrá el número de caracteres escritos.
%x	Hexadecimal.

3.3. ENTRADA Y SALIDA DE DATOS.

Los datos pueden almacenarse en la memoria RAM, en variables de dos formas diferentes, por operaciones de asignación o por el uso de alguna instrucción de entrada de datos como **scanf()**.

El bloque o símbolo utilizado en diagramas de flujo para representar una acción de **entrada de datos**, es decir, **la introducción de un valor desde el teclado a la memoria RAM de la computadora**, es el siguiente:

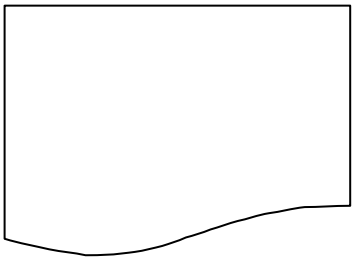
Cuando se usa la función **scanf()**, ésta lee el valor de un dispositivo de entrada, casi siempre del teclado, y lo almacena en la celda asignada en la memoria RAM para recibir dicho valor. La sintaxis de esta función es:

SINTAXIS: **scanf** (“cadena de control”, lista de argumentos);

A continuación, tenemos el bloque o símbolo que se utiliza para representar una **operación matemática o de asignación,** esta operación se realiza dentro de la memoria RAM**;** el bloque es un rectángulo como el siguiente:

La operación matemática o de asignación se ejecuta en la memoria RAM, para que esto sea posible, las variables de una fórmula aritmética deben tener un valor.

Por otro lado, **para poder visualizar valores de variables en el monitor de la computadora (salida de datos)**, se debe utilizar la función **printf()**, la cual se representa en diagrama de flujo de la siguiente manera:



Aunque esta visualización se muestra, generalmente, en el monitor de la computadora también puede ser en una impresora o un disco. La sintaxis de esta función es:

SINTAXIS: **printf** (“cadena de control”, lista de argumentos);

3.4. EJEMPLOS DE PROGRAMAS CON ENTRADA-SALIDA DE DATOS Y EXPRESIONES ARITMETICAS.

A continuación se presentan algunos ejemplos de programas donde se pondrá en práctica toda la teoría explicada hasta estos momentos. Se recomienda al maestro hacer primeramente, en el pizarrón, el diagrama de flujo correspondiente y simultáneamente explicar el código a sus alumnos. Finalmente, los alumnos deben practicar en las computadoras con el software apropiado.

```
/* prog3-2.c */
#include<stdio.h>
/* convierte de pies a metros /
main()
{
   int PIES;
   float METROS;
   clrscr(); /* limpiar pantalla de ejecución */
   printf("introduzca el numero de pies a convertir \n");
   scanf("%d", &PIES);
   METROS = PIES * 0.3084;     /* formula de conversion */
   printf("%d pies son %f metros \n", PIES, METROS);
   printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

El programa **prog3-2.c** tiene dos variables: **PIES** y **METROS,** la primera es de tipo entero (**int**), la segunda es real de simple precisión (**float**). La función **clrscr()** se utiliza para limpiar la pantalla de ejecución donde se visualizan los resultados del programa. Como se podrá observar, en el **scanf()** se escribe el amperson "**&**" , este carácter es necesario para que funcione adecuadamente, por el momento comentaremos que indica la dirección de memoria donde se almacenó el valor de la variable, sin embargo, por la complejidad de su conceptaulización se dejará para posteriores explicaciones.

En lenguaje C es posible tener en las expresiones o fórmulas a variables de diferente tipo como está en este programa. La limitante de **prog3-2.c** es que no se puede hacer conversiones de pies con número reales, solamente enteros. Además, cuando se ejecute el programa se verá un resultado con 6 o 7 decimales. Para quitar la limitante y permitir sólo 2 decimales en el resultado, se tiene el siguiente programa **prog3-3.c**.

```
/* prog3-3.c */
#include<stdio.h>
/* convierte de pies a metros */
main()
{
   float PIES, METROS;
   clrscr();  /* limpiar la pantalla de ejecucion */
   printf("introduzca en numero de pies a convertir \n");
   scanf("%f", &PIES);
   METROS = PIES * 0.3084;     /* formula de conversion */
   printf("%8.2f pies son %8.2f metros \n", PIES, METROS);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Las diferencias entre **prog3-2.c** y **prog3-3.c** se reflejan en la declaración de variables, ahora ambas son de tipo real de simple precisión (**float**). En el segundo **printf()** podemos observar, en el especificador de formato, que hay un **8.2** entre **%** y **f**, esto indica que los valores que se verán en los resultados serán de máximo 8 caracteres, los cuales serán de máximo cinco números en la parte entera y dos en la parte decimal, el punto decimal también se contabiliza entre los 8 caracteres. Con esto aparecerán sólo dos decimales en el resultado.

A continuación, tenemos el programa **prog3-4.c** que calcula el área de un cuadrado.

```
/* prog3-4.c */
#include<stdio.h>
/* calcula el area de un cuadrado */
main()
{
   float AREA,LADO;
   clrscr();
   printf("Introduzca la longitud del lado \n");
   scanf("%f",&LADO);
   AREA = LADO * LADO;
   printf("El area de un cuadrado con lado = %8.2f es %8.2f \n",LADO,AREA);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

En seguida, con el programa **prog3-5.c** calcularemos el área de un triángulo. El cálculo del área está limitada por la longitud de los lados, esto es, si LADO1=31, LADO2=45 y LADO3=22 centímetros, entonces estos lados si pueden formar un triángulo, pero si LADO1= 98, LADO2=45 y LADO3=12 centímetros, entonces estos lados no pueden formar un triángulo. En Conclusión, si la sumatoria de los lados más pequeños es mayor al lado más grande, se puede formar un triángulo. Para el primer caso 31+22=53, dado que 53 es mayor a 45 se forma el triángulo, pero en el segundo caso donde 45+12=57 y dado que 57 es menor que 98, no se puede formar el triángulo.

La fórmula para determinar el área es:

$$\text{Área} = \sqrt{\text{LADOS}(\text{LADOS} - \text{LADO1})(\text{LADOS} - \text{LADO2})(\text{LADOS} - \text{LADO3})}$$

$$\text{donde, LADOS} = (\text{LADO1}+\text{LADO2}+\text{LADO3})/2$$

Por lo tanto al ejecutar el programa **prog3-5.c**, si se dan tres lados que no forman un triángulo se generará un error. La función **sqrt()** se utiliza para calcular la raíz cuadrada, misma que necesita de la directiva del preprocesador llamada <**math.h**> que se escribió al principio del programa indicando el uso de una función matemática.

```
/* prog3-5.c */
#include <stdio.h>
#include <math.h>
/* calcula el area de triangulo en base a sus lados */
main()
{
  float LADO1,LADO2,LADO3,AREA,OPERACION,LADOS;
  clrscr();
  printf("Dame la longitud del lado # 1 \n");
  scanf("%f",&LADO1);
  printf("Dame la longitud del lado # 2 \n");
  scanf("%f",&LADO2);
  printf("Dame la longitud del lado # 3 \n");
  scanf("%f",&LADO3);
  LADOS=(LADO1+LADO2+LADO3)/2;
  OPERACION=LADOS*(LADOS-LADO1)*(LADOS-LADO2)*(LADOS-LADO3);
  AREA=sqrt(OPERACION);
  printf("El area del triangulo es %8.2f \n",AREA);
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

En otro ejemplo tenemos el programa **prog3-6.c** en el cual encontrará la definición de la constante PI con valor de 3.1416, así mismo, encontrará la función **pow()** que permite elevar cualquier número a cualquier potencia.

```
/* prog3-6.c */
#include <stdio.h>
#include <math.h>
#define PI 3.1416
/* calcula el area de un circulo */
main()
{
   float RADIO,AREA;
   clrscr();
   printf("Dame el radio del circulo \n");
   scanf("%f",&RADIO);
   AREA=PI*pow(RADIO,2);
   printf("El area de un circulo con radio %8.2f es %8.2f \n",RADIO,AREA);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Por otro lado, en el programa **prog3-7.c** se plantea un problema de caída libre, en el cual una niña deja caer su cartera desde el último piso de la torre de Sears en Chicago cuya altura es de 1,454 pies. Calcule la velocidad de impacto al llegar al suelo.

$$\mathbf{V} = \sqrt{2gh}$$

Donde *V* es la velocidad final, *g* es la gravedad equivalente a 32 pies/seg^2 y *h* es la altura de la torre igual a 1,454 pies

En este programa encontramos la definición de una constante llamada GRAVEDAD con un valor de 32, también se realiza una raíz cuadrada con **sqrt()** y se incluye la directiva del preprocesador **<math.h>**

```
/* prog3-7.c */
#include <stdio.h>
#include <math.h>
#define GRAVEDAD 32
/* resuelve un problema de caida libre */
main()
{
   float ALTURA, VALOR,VEL_FINAL;
   clrscr();
   printf("Dame la altura de la torre \n");
   scanf("%f",&ALTURA);
   VALOR= 2*GRAVEDAD*ALTURA;
   VEL_FINAL=sqrt(VALOR);
   printf("La Velocidad Final es %8.2f pies/seg \n",VEL_FINAL);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Otro problema será resuelto con el programa **prog3-8.c** el cual hace la conversión de grados Fahrenheit a centígrados. La fórmula de conversión es **C= (5/9) (F-32).**

```
/* prog3-8 */
#include <stdio.h>
/* convierte de grados fahrenheit a grados centigrados */
main()
{
   float CENT,FAHR;
   clrscr();
   printf("Dame los grados fahrenheit a convertir \n");
   scanf("%f",&FAHR);
   CENT=(5.0/9.0)*(FAHR-32);
   printf("%8.2f grados fahrenheit equivale %8.2f grados centigrados \n",FAHR,CENT);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Finalmente, en el programa **prog3-9.c** se resuelve el siguiente problema. Un avión que se encuentra volando a una altura de 10,000 pies y a 150 millas por hora, desarrolla velocidades relativas con respecto al viento de 300 millas por hora en la parte superior del ala y 80 millas por hora debajo del ala. Si el área del ala es de 160 pies^2 , ¿Cuál será la fuerza perpendicular a ella?

Suponiendo que la densidad a 10,000 pies de altura es p=0.001756 slugs/pies^3 . La fórmula es

$$\text{Fuerza} = A * \frac{1}{2} p (V1^2 - V2^2)$$

Donde ***A*** es el área de la ala del avión: 160 pies^2 ; ***p*** es la densidad a 10,000 pies de altura: 0.001756; ***V1*** es la velocidad relativa del viento en la parte superior del ala: 300 millas por hora, ***V2*** es la velocidad relativa del viento en la parte inferior del ala: 80 millas por hora. El factor que debe usarse para convertir las millas por hora a pies por segundo es 44/30.

```
/* prog3-9.c */
#include<stdio.h>
#include<math.h>
/* resuelve un problema de aviacion */
#define DENSIDAD 0.001756
main()
{
   float AREA,V1,V2,FUERZA;
   clrscr();
   printf("Dame el area del ala \n");
   scanf("%f",&AREA);
   printf("Dame la velocidad relativa del viento en la parte superior del ala \n");
   scanf("%f",&V1);
   printf("Dame la velocidad relativa del viento en la parte inferior del ala \n");
   scanf("%f",&V2);
   /* convertir las millas por hora a pies por segundo */
   V1=(44.0/30.0)*V1;
   V2=(44.0/30.0)*V2;
   FUERZA= AREA * (1.0/2.0) * DENSIDAD * (pow(V1,2)-pow(V2,2));
   printf("la fuerza del viento perpendicular al ala es de %8.2f slugs-ft/seg2 \n",FUERZA);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

3.5. ENTRADA Y SALIDA DE CARACTERES Y CADENAS DE CARACTERES.

Existen otras funciones semejantes a **printf()** y **scanf()**, que permiten la introducción de datos a la memoria RAM de la computadora desde el teclado o enviar datos, desde la memoria RAM, al monitor. Estas funciones son:

Función	**Operación**
getchar()	Lee un caracter desde el teclado, espera retorno de carro. <enter>
getche()	Lee un caracter con eco, no espera retorno de carro. <enter>
getch()	Lee un caracter sin eco, no espera retorno de carro. <enter>
putchar()	Escribe un caracter en la pantalla.
gets()	Lee una cadena de caracteres desde el teclado.
puts()	Escribe una cadena de caracteres en la pantalla.

La principal característica de las funciones anteriores es que los valores insertados son caracteres y/o cadenas de caracteres, mismas que pueden almacenar cualquier caracter incluyendo números. Aun cuando en la variables caracter se pueden almacenar números, su utilización, generalmente, es para almacenar letras, espacios en blancos y cualquier otro carácter.

Otra consideración que se debe tener presente es que cuando se almacenan números en variables de tipo caracter no se pueden realizar operaciones algebraicas. Considere los siguientes ejemplos.

Ejemplo #1:	
1. Sean operador1, operador2 y resultado variables de tipo enteras.	**int operador1, operador2, resultado;**
2. Se les asigna el valor de 40 y 20 respectivamente	**operador1=40** **operador2=20**
3. Se realiza una operación	**resultado = operador1+operador2**
4. La solución a la anterior operación es	**resultado = 60**

Ejemplo #2:	
1. Sean operador1, operador2 y resultado variables de tipo carácter	**char operador1[5], operador2[5], resultado[5];**
2. Se les asigna el valor de 40 y 20 respectivamente	**operador1=”40”** **operador2=”20”**
3. Se realiza la operación	**resultado = operador1+operador2**
4. La solución a la anterior operación es	**resultado = ”4020”**

En el ejemplo #1 se realizó una operación algebraica, en el ejemplo #2 se realizó una concatenación. De hecho podemos ver que en la asignación del ejemplo #2, los valores se escribieron entre comillas en un intento por denotar que no es una asignación algebraica, además se incluyó “[5]” en la tres variables lo cual significa que pueden almacenar hasta cinco caracteres.

La asignación y concatenación de caracteres no se realizan en lenguaje C como se muestra en este ejemplo, por lo que la forma adecuada se dejará para más adelante.

En el programa **prog3-10.c** encontrará un ejemplo del uso de **gets()** y **puts()**, donde **gets()** se utiliza para insertar el nombre de una persona

```
/* prog3-10.c */
#include<stdio.h>
/* envia un saludo al usuario */
main()
{
  char NOMBRE[30];
  clrscr();
  printf("Dame tu nombre \n");
  gets(NOMBRE);
  printf("¡Hola!, %s, ¿Como Estas? \n",NOMBRE);
  puts("¡Soy yo!, tu computadora...");
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

1. La línea correspondiente a **char NOMBRE[30]** significa que se declara una variable llamada **NOMBRE**, de tipo carácter, que permite insertar hasta 30 caracteres, por este motivo podemos decir que la variable **NOMBRE** es una cadena de caracteres, también conocida como **string**. Ahora bien, aunque esta variable permite insertar hasta 30 caracteres, se debe tomar en cuenta que lenguaje C coloca al final de una cadena de caracteres un terminador nulo, esto significa que si usted corre el programa e inserta 30 o más caracteres el programa producirá un error, por lo que recomendamos insertar máximo 29 caracteres.
2. La línea que se refiere a **gets(NOMBRE);**, es la instrucción que permite insertar, desde el teclado, el nombre de alguna persona.
3. La línea correspondiente a **printf("¡Hola!, %s, ¨¿Como Estas? \n",NOMBRE)**, es la instrucción que permite ver el nombre y un mensaje en la pantalla de su computadora. **%s** es un especificador de formato que permite visualizar una cadena de caracteres.
4. La línea que se refiere a **puts("!Soy yo!, tu computadora...");. puts** es una función que únicamente permite enviar una cadena de caracteres al monitor de la computadora, no permite visualizar el valor de una variable.

Ahora bien, aunque en este programa utilizamos **gets()** para introducir el nombre, desde el teclado, a la memoria RAM de la computadora, esto no impide que utilicemos **scanf()**. Esto lo puede verificar en el siguiente programa llamado **prog3-11. C**, donde se ha sustituido **gets()** por **scanf(),** y sustituimos **puts()** por **printf()**.

```
/* prog3-11.c */
#include<stdio.h>
/* envia un saludo al usuario */
main()
{
  char NOMBRE[30];
  clrscr();
  printf("Dame tu nombre \n");
  scanf(" %[^\n]",NOMBRE);
  printf("¡Hola!, %s, ¿Como Estas? \n",NOMBRE);
  printf("¡Soy yo!, tu computadora...");
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

En la línea **scanf(" %[^\n]", NOMBRE);,** bien podría usarse el especificador de formato **%s** en lugar de **%[^\n].** La diferencia va estar en que con **%s**, al correr el programa, no se podrá dar en espacios en blanco entre el nombre y los apellidos, y con **%[^\n]** si podrá hacerse. Para ver esta situación se le pide que ejecute el programa haciendo los cambios correspondientes.

La diferencia entre **puts()** y **printf()**, está en que **puts()** sólo puede imprimir una cadena de caracteres, no puede imprimir variables, **printf()** si puede hacerlo. Por lo tanto, SERÍA INCORRECTO ESCRIBIR: **puts("¡Hola!, %s, ¿Cómo Estas? \n", NOMBRE),** ya que esta línea implica imprimir el valor de la variable **NOMBRE.**

En el último programa de este capítulo se verá el uso de **getchar()** y **getche()**, el especificador de formato **%c** y cómo convertir de letras mayúsculas a minúsculas y viceversa. Observe el programa **prog3-12.c**. En el caso de la función **toupper()** que convierte de minúscula a mayúscula, si se le pulsa una letra que ya es mayúscula no produce ningún cambio, lo mismo sucede con **tolower()** que convierte de mayúscula a minúscula, si se le inserta una letra que ya es minúscula no produce ningún cambio. En ambas funciones, si se inserta un caracter que no es una letra, tampoco se hará ningún cambio. Al ejecutar el programa observará la diferencia entre **getchar()** y **getche()**.

```
/* prog3-12.c */
#include<stdio.h>
#include<ctype.h>
/* imprime una letras minusculas y mayusculas */
main()
{
  char MIN,MAY,CMIN,CMAY;
  clrscr();
  puts("\n Dame un letra Minuscula \n");
  MIN=getche();
  puts("\n Dame un letra Mayuscula \n");
  MAY=getchar();
  CMIN=toupper(MIN); /* devuelve el equivalente en mayuscula */
  CMAY=tolower(MAY); /* devuelve el equivalente en minuscula */
  printf("\n La mayuscula de %c es %c \n", MIN, CMIN);
  printf("\n La minuscula de %c es %c \n", MAY, CMAY);
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

La directiva del preprocesador **"ctype.h"** se requiere en este programa porque se utilizan las funciones **"toupper()"** y **"tolower()"**.

3.6 DEFICIENCIAS EN LOS PROGRAMAS

La mayor parte de los programas realizados en este capítulo presentan algunas deficiencias, las cuales en la medida que avance el curso deberán ser corregidas. La mayor parte de éstas se deben a la posible inserción de datos que arrojarán resultados incongruentes o que no tienen significados en la realidad, por ejemplo:

a) En los programas **prog3-2.c** y **prog3-3.c** que hacen la conversión pies a metros, si se insertan números negativos el resultado que arrojaría el programa no tiene ningún sentido en la realidad.
b) Lo mismo sucede en programa **prog3-4.c**, que calcula el área de un cuadrado, pues al insertar un número negativo, el resultado no tiene sentido alguno.
c) En el caso del programa **prog3-5.c**, que calcula el área de un triángulo; si se insertan datos que no forman un triángulo, **el programa generará un error**, pues la función que calcula la raíz cuadrada: **sqrt()**, no puede generar un resultado si el radical es negativo.
d) En el programa **prog3-6.c**, que calcula el área de un círculo; si se inserta un radio negativo, el programa arrojará un resultado sin sentido.
e) Lo mismo sucede en programa **prog3-7.c** que calcula la velocidad final de un objeto en caída libre cuando éste llega al suelo; si se inserta una altura negativa, el resultado no tiene sentido.
f) En el caso del programa **prog3-8.c**, que realiza la conversión de la temperatura de grados farenheit centígrados, no se presenta problema alguno, porque la conversión de valores negativos si tienen sentido en la realidad.

g) Finalmente, en el programa **prog3-9.c** que calcula la fuerza perpendicular en la ala de un avión, se presenta problemas similares, pues si se inserta una área negativa del ala del avión, los resultados no tienen sentido alguno en la realidad.

   En algunos de estos programas, el programador deberá decidir qué hacer cuando se inserten valores de cero, pues por ejemplo, no se puede calcular el área de un cuadro con lado=0, porque no existe un cuadrado con esas dimensiones.

Por otro lado, es necesario que el alumno sepa diferenciar entre lo que significa **que un programa arroje resultados sin sentido ó que genere un error**, ya que son situaciones completamente diferentes. Se dejará que el maestro aborde el tema y explique esto, así como que el mismo alumno experimente con los programas. Para esto se deberá explicar las diferencias entre errores de sintaxis, de lógica y de ejecución.

3.7 PROBLEMAS PROPUESTOS.

a) Realice un programa que eleve al cuadrado y al cubo cualquier número, y que imprima el número junto a su cuadrado y su cubo.
b) Antes de despegar un avión, el piloto anuncia el tiempo estimado de vuelo en minutos, realice un programa que le ayude a determinar el porcentaje de avance del vuelo, teniendo como dato conocido al tiempo transcurrido del vuelo en minutos y el tiempo estimado de vuelo.
c) Un maestro desea determinar el porcentaje de aprobados y reprobados en un grupo; sólo sabe cuántos alumnos han aprobado y cuántos han reprobado. Realice un programa que le ayude a calcular estos porcentajes.
d) Una maestra midió la altura de Juanito al principio y al final de año escolar. Realice un programa que le ayude a determinar el porcentaje de crecimiento de Juanito.

Capítulo #4
Estructuras Selectivas
if() y switch()

La programación estructurada maneja instrucciones **selectivas, repetitivas y secuenciales**. A estas instrucciones se les llama estructuras porque tienen un principio y un fin bien definido. La finalidad de la programación estructurada es que las instrucciones se ejecuten **secuencialmente**.

Existe una instrucción llamada **goto**. El uso que se le da a esta instrucción es para alterar la secuencia de ejecución normal de un programa. Anteriormente, cuando no existían los lenguajes estructurados, la sentencia **goto** era muy utilizada. Sin embargo, desde que este tipo de lenguajes aparecieron, la instrucción **goto** quedó relegada y su uso descontinuado. La razón de esto es porque rompe la estructuración de un programa, haciéndolo más difícil de entender y de modificar. De hecho, utilizar **goto**, en un programa, es sinónimo de un programa obsoleto y de mala calidad. Por lo tanto, no se recomienda su uso.

Aquí surge la polémica, ¿Cuándo un programa es de buena calidad?. Para empezar, este tema está más ampliamente explicado en la rama de la ingeniería que se denomina **Ingeniería de Software**. Solamente diremos que **el buen programa es aquél que es fácil de entender, fácil de modificar y que arroja los resultados correctos**. Por lo tanto, cuando hagamos un programa debemos hacernos la siguiente consideración: Si este programa que estoy haciendo, lo ve una persona que tiene conocimientos medios-avanzados acerca del lenguaje, ¿será capaz de entenderlo?.

4.1 SENTENCIA if()

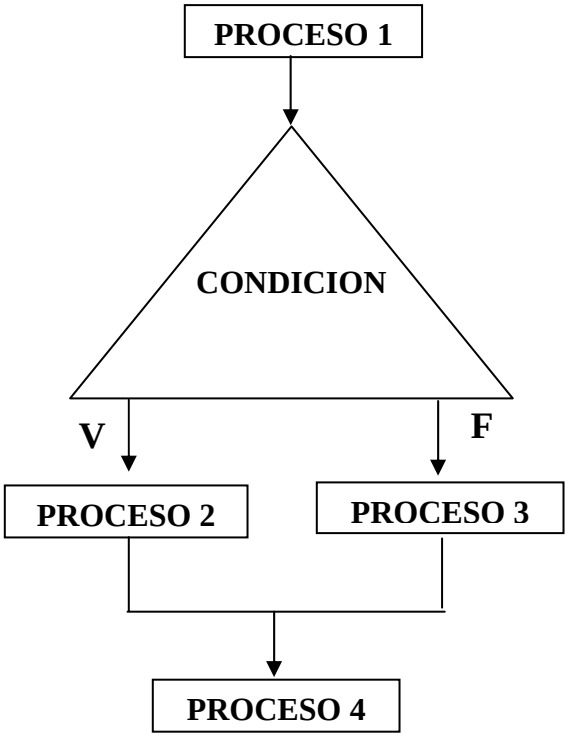
Volviendo al tema de las estructuras selectivas. Estas son utilizadas en un programa **cuando tenemos que decidir entre dos caminos (alternativas) ó más**.

En el caso de **if()** se utiliza cuando tenemos que elegir entre dos caminos: **Falso y Verdadero**. Muchas son las situaciones, dentro de la programación, que enfrentan esta situación, por ejemplo, un alumno sólo puede estar aprobado o reprobado, una persona sólo puede ser del sexo masculino o femenino, una mercancía puede estar entregada o no, una mujer puede estar embarazada o no, etc. en fin, hay muchas situaciones que son binarias, es decir, sólo tienen dos alternativas.

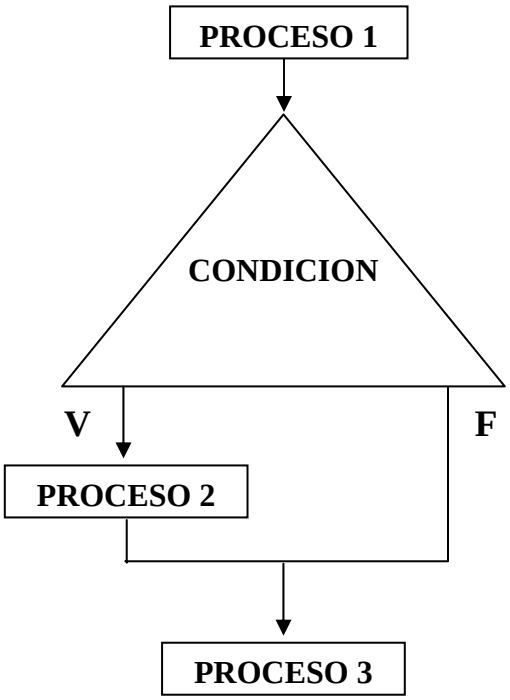
Dentro de la instrucción **if()** haremos uso de los **operadores relacionales y los lógicos**. Durante este punto abordaremos programas que se enfrentan a situaciones binarias.

La sentencia **if()** escoge entre dos caminos, según sea la condición: **Falso o Verdadero**. Ahora bien, es necesario recalcar que el **if()** puede tener solamente parte verdadera, mientras que la parte falsa se puede omitir. La parte falsa del **if()** comienza con la cláusula **else**, de este modo **tanto la parte falsa como la parte verdadera están delimitados por llaves { }**.

La figura para representar la sentencia if(), en diagramas de flujo, es la siguiente:



****



Lo que se debe destacar de las figuras anteriores es que en ambos casos **las líneas de falso y verdadero se unen para continuar un mismo camino**. La primer figura tiene procesos por falso y verdadero y en la segunda sólo hay un proceso por la parte verdadera, la parte falsa no tiene proceso alguno.

En el primer programa de este capítulo **prog4-1.c** se trata de conocer la situación académica de un alumno en una materia según las calificaciones en dos exámenes parciales, de este modo, encontrará que si el **PROMEDIO** es mayor o igual que 70, la variable **SITUACION** tendrá el valor de **APROBADO**, pero si el **PROMEDIO** es menor a 70, la variable **SITUACION** tendrá el valor de **REPROBADO**. En este caso en particular, el **if()** tiene parte falsa y parte verdadera.

Observe que la variable **SITUACION** es del tipo carácter con máximo 30 caracteres, así mismo, observe que para asignar valor a la variable **SITUACION** se utilizó la función **strcpy()**, que significa copiar una cadena de caracteres: **STRING COPY**. Para que esta función trabaje adecuadamente se debe agregar, al principio del programa, la directiva del preprocesador <**string.h**>.

```
/* prog4-1.c */
#include<stdio.h>
#include<string.h>
/* programa que determina, en base a las calificaciones de los parciales,
el promedio del alumno y si esta aprobado o reprobado */
main()
{
   long int MATRICULA;
   char NOMBRE[30], SITUACION[30];
   float PARCIAL1, PARCIAL2, PROMEDIO;
   clrscr();
   printf("Dame la matricula \n");
   scanf("%ld",&MATRICULA);
   printf("Dame el nombre \n");
   scanf(" %[^\n]",NOMBRE);
   printf("Dame la calificacion del primer parcial \n");
   scanf("%f",&PARCIAL1);
   printf("Dame la calificacion del segundo parcial \n");
   scanf("%f",&PARCIAL2);
   PROMEDIO=(PARCIAL1+PARCIAL2)/2;
   if(PROMEDIO>=70)
   {
      strcpy(SITUACION,"APROBADO");
   }
   else
   {
      strcpy(SITUACION,"REPROBADO");
   } /* fin del if */
   printf("El alumno con matricula %ld Se llama %s \n",MATRICULA,NOMBRE);
   printf("tiene promedio de :%8.2f y esta %s\n", PROMEDIO,SITUACION);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

En el siguiente programa **prog4-2.c** encontrará un **if()** que compara si dos valores son iguales. Observe que la comparación se realiza con doble igual (= =).

```
/* prog4-2.c */
#include<stdio.h>
/* determina si dos valores son iguales */
main()
{
  int VALOR1,VALOR2;
  clrscr();
  printf("Dame el primer numero \n");
  scanf("%d",&VALOR1);
  printf("Dame el segundo numero \n");
  scanf("%d",&VALOR2);
  if(VALOR1==VALOR2)
  {
     printf("Los numeros son iguales\n");
  }
  else
  {
     printf("Los numeros son diferentes\n");
  } /* fin del if */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

En seguida tenemos el programa **prog4-3.c** en él encontrará un **if()** que compara si dos cadenas de caracteres son iguales. Este programa es muy delicado, por lo que le recomendamos ser muy cuidadoso, por ejemplo, si la primer cadena es **FIME** y la segunda **fime**, el resultado será que las cadenas son diferentes porque una tiene letras mayúsculas y la otra minúsculas, esto es, la computadora no reconoce que es lo mismo. Por otro lado, si la primer cadena es **FIME** y la segunda **FIME**, pero por error, al teclear la segunda cadena, dimos un espacio en blanco después de la **E**, el resultado será que las cadenas son diferentes. Por eso, al correr el programa, asegúrese de teclear exactamente iguales las dos cadenas.

También observe que la comparación se realizó con la función **strcmp()**, que significa comparación de cadenas: **STRING COMPARE**. El signo de admiración significa una negación. Observe que no se utilizan los signos de igual como en el anterior programa.

```
/* prog4-3.c */
#include<stdio.h>
/* determina si dos cadenas de caracteres son iguales */
main()
{
  char CADENA1[30],CADENA2[30];
  clrscr();
  printf("Dame la primer cadena \n");
  scanf(" %[^\n]",CADENA1);
  printf("Dame la segunda cadena \n");
  scanf(" %[^\n]",CADENA2);
  if(!strcmp(CADENA1,CADENA2))
  {
     printf("Las cadenas son iguales\n");
  }
  else
  {
     printf("Las cadenas son diferentes\n");
  } /* fin del if */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

A continuación tenemos el programa **prog4-4.c** el cual hace una comparación para determinar, a partir de dos números, cuál es mayor. **Si se introducen dos números iguales el programa arrojará un resultado incongruente**. Como ejercicio, se dejará al alumno que modifique el programa, de forma que no se presente esta incongruencia.

```
/* prog4-4.c */
#include<stdio.h>
/* determina, a partir de dos valores, cual es mayor */
main()
{
  int VALOR1,VALOR2;
  clrscr();
  printf("Dame el primer numero \n");
  scanf("%d",&VALOR1);
  printf("Dame el segundo numero \n");
  scanf("%d",&VALOR2);
  if(VALOR1>VALOR2)
  {
     printf("El valor1: %d es mayor que el valor2: %d\n",VALOR1,VALOR2);
  }
  else
  {
     printf("El valor2: %d es mayor que el valor1: %d\n",VALOR2,VALOR1);
  } /* fin del if */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

Ahora tenemos el programa **prog4-5.c** el cual hace una comparación entre dos cadenas para determinar cuál es mayor. **Si se teclean dos cadenas exactamente iguales, el programa arrojará un resultado incongruente**. Al igual que el programa anterior, se dejará al alumno la modificación del programa para que no se presente esta incongruencia.

```
/* prog4-5.c */
#include<stdio.h>
/* determina, a partir de dos cadenas de caracteres, cual es mayor */
main()
{
  char CADENA1[30],CADENA2[30];
  int RESULTADO;
  clrscr();
  printf("Dame la primer cadena \n");
  scanf(" %[^\n]",CADENA1);
  printf("Dame la segunda cadena \n");
  scanf(" %[^\n]",CADENA2);
  RESULTADO=strcmp(CADENA1,CADENA2);
  if(RESULTADO>0)
  {
     printf("La cadena1: %s es mayor a la cadena2: %s \n",CADENA1,CADENA2);
  }
  else
  {
     printf("La cadena2: %s es mayor a la cadena1:%s \n",CADENA2,CADENA1);
  } /* fin del if */
  printf("Resultado %d \n", RESULTADO);
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

Auxiliados con los programas **prog4-5.c** y **prog4-6.c** analizaremos como funciona internamente el **if()**. De esta manera, cuando se ejecuta un **if()**, se realiza una operación que puede devolver un valor de cero o diferente de cero.

Observe que El programa **prog4-6.c** es el mismo que el **prog4-2.c** pero está modificado. Aquí se agregó la variable **RESULTADO**, y la operación **RESULTADO=VALOR1==VALOR2**. De tal forma que si **VALOR1** es igual a **VALOR2**, **RESULTADO** tendrá valor de uno, si no es así, **RESULTADO** tendrá valor igual a cero.

De este modo, para el **if(), si la operación da resultado de uno es verdadero, si da resultado diferente a uno es falso**.

```
/* prog4-6.c */
#include<stdio.h>
/* determina si dos valores son iguales */
/* este programa es el mismo que prog4-2.c pero modificado */
main()
{
  int VALOR1,VALOR2, RESULTADO;
  clrscr();
  printf("Dame el primer numero \n");
  scanf("%d",&VALOR1);
  printf("Dame el segundo numero \n");
  scanf("%d",&VALOR2);
  RESULTADO=VALOR1==VALOR2;
  if(RESULTADO)
  {
     printf("Los numeros son iguales\n");
  }
  else
  {
     printf("Los numeros son diferentes\n");
  } /* fin del if */
  printf("Resultado %d \n", RESULTADO);
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

Si en el anterior programa cambiáramos la fórmula y tuviéramos **RESULTADO=VALOR1>VALOR2**, donde **VALOR1=53** y **VALOR2=21**, el **RESULTADO** sería igual uno, pero si **VALOR1=23** y **VALOR2=59**, el **RESULTADO** sería igual a cero. **Todo esto es válido cuando se trata de variables numéricas**.

Sin embargo, cuando se trata de una comparación de cadenas de caracteres resulta lo siguiente:

RESULTADO=strcmp(CADENA1, CADENA2)

En estas condiciones tenemos tres posibles resultados
Primero, la variable **RESULTADO** es igual cero cuando las cadenas son iguales.
Segundo, la variable **RESULTADO** es mayor a cero cuando la CADENA1 **es mayor a la** CADENA2.
Y finalmente, la variable **RESULTADO** es menor a cero cuando la CADENA1 **es menor a la** CADENA2.

Por otro lado, si CADENA1=”A” , CADENA2=”a” y RESULTADO=strcmp(CADENA1,CADENA2), la variable resultado sería menor que cero porque el código ASCII de **A=65** y el código ASCII de **a=97.** Por lo tanto **“a es mayor que A”.**

4.1.1 . Funciones de Cadena de Caracteres.

Con el siguiente programa **prog4-7.c** se verá el uso de **funciones de cadena**, en este caso la función **isdigit()** que determina si una cadena es un dígito (número) o no.

```
/* prog4-7.c */
#include<stdio.h>
#include<ctype.h>
/* determina si un caracter es un digito (numero) */
main()
{
  char CARACTER;
  clrscr();
  printf("Dame un caracter \n");
  CARACTER=getchar();
  if(isdigit(CARACTER))
  {
     printf("Si es digito: %c \n",CARACTER);
  }
  else
  {
     printf("No es digito: %c \n",CARACTER);
  } /* fin del if */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

Además de isdigit(), strcmp() y strcpy(), se dejará como ejercicio para que el alumno investigue y utilice las siguientes funciones de cadena:

isalnum()
isalpha()
iscntrl()
isgraph()
islower()
isprint()
ispunct()
isspace()
isupper()
isxdigit()
strcat()
strchr()

En el siguiente programa **prog4-8.c**, se verá el uso del **%** como operador matemático binario. Es necesario recalcar, que no se debe confundir el signo del porciento (**%**) cuando se usa dentro de la función **scanf()** o **printf()** y cuando se usa en una operación matemática. Esto es, cuando se usa dentro de la función **scanf()** o **printf()** este signo es parte del especificador de formato, pero cuando se usa dentro de una operación matemática este signo es una operación de módulo (residuo) de una división entre números enteros.

De este modo, si tuviéramos lo siguiente:

Resultado=25/5 entonces Resultado=5
Resultado=25%5 entonces Resultado=0
Resultado=25/4 entonces Resultado=6.25
Resultado=25%4 entonces Resultado=1

Para resolver el siguiente problema del programa **prog4-8.c**, el cual determina si un número es par ó impar, se utilizó, en la fórmula matemática, el módulo (%) del número dado dividiéndolo entre dos. Posteriormente, dentro del **if()** se invirtió la salida con el signo de exclamación (**!**). Un inconveniente de este programa es que cuando el número analizado sea cero se imprimirá un resultado incongruente.

```
/* prog4-8.c */
#include<stdio.h>
/* determina si un numero es par o impar */
main()
{
   int NUMERO,RESULTADO;
   clrscr();
   printf("Dame un numero \n");
   scanf("%d",&NUMERO);
   RESULTADO=NUMERO%2;
   if(!RESULTADO)
   {
      printf("El numero %d es par \n",NUMERO);
   }
   else
   {
      printf("El numero %d es impar \n", NUMERO);
   } /* fin del if */
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Hasta el momento hemos utilizado el **if()** con parte falsa y verdadera. Como dijimos anteriormente, hay ocasiones que el **if()** puede tener solamente parte verdadera. Para ver este ejemplo desarrollaremos el programa **prog4-9.c**, el cual tiene la siguiente redacción.

Se desea calcular el pago del recibo de luz de una persona. Los datos de entrada son: Número de Medidor, Cantidad de Kilowatts con consumidos, costo del kilowatt y saldo anterior. Si no se pagó el recibo anterior habrá un recargo de quince pesos. La manera de saber si el recibo anterior se pagó o no, es preguntar si el saldo anterior es mayor a cero. Por lo tanto, el pago se calcula multiplicando la cantidad de kilowatts consumidos por el costo del kilowatt, y si el recibo anterior no fue pagado se agrega el recargo y el saldo anterior.

```
/* prog4-9.c */
#include<stdio.h>
/* determina el pago del recibo de luz */
main()
{
   long int MEDIDOR;
   float KILOWATT,COSTO_KWT,SALDO_ANTERIOR, PAGO;
   clrscr();
   printf("Dame el numero de medidor \n");
   scanf("%ld",&MEDIDOR);
   printf("Dame la cantidad de kilowatts consumidos \n");
```

```
   scanf("%f",&KILOWATT);
   printf("Dame el costo por kilowatt \n");
   scanf("%f",&COSTO_KWT);
   printf("Dame el saldo anterior\n");
   scanf("%f",&SALDO_ANTERIOR);
   PAGO=KILOWATT*COSTO_KWT;
   if(SALDO_ANTERIOR>0)
   {
      PAGO=PAGO+15+SALDO_ANTERIOR;
   } /* fin del if */
   printf("El Pago a realizar por el medidor # %ld es $ %8.2f \n",MEDIDOR,PAGO);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

4.2 if() 's ANIDADOS

Los **if()**'s anidados se refiere al hecho de que hay una instrucción **if()** dentro de otra instrucción **if()**. Para entender este concepto analizaremos el programa **prog4-10.c** en el que podemos ver que por la parte falsa del primer **if()** hay otro **if()**. Otro detalle que se debe observar es que los dos **if()** tienen parte falsa y verdadera, esto no siempre sucede así. Este programa es el mismo que el **prog4-4.c** pero está modificado.

```
/* prog4-10.c */
#include<stdio.h>
/* determina, a partir de dos valores, cual es mayor */
main()
{
  int VALOR1,VALOR2;
  clrscr();
  printf("Dame el primer numero \n");
  scanf("%d",&VALOR1);
  printf("Dame el segundo numero \n");
  scanf("%d",&VALOR2);
  if(VALOR1==VALOR2)
  {
       printf("No se puede determinar cual numero es mayor porque son iguales \n");
  }
  else
  {
       if(VALOR1>VALOR2)
       {
           printf("El valor1: %d es mayor que el valor2: %d\n",VALOR1,VALOR2);
       }
       else
       {
           printf("El valor2: %d es mayor que el valor1: %d\n",VALOR2,VALOR1);
       } /* fin del if(VALOR1>VALOR2) */
  } /* fin del if(VALOR1==VALOR2) */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

A continuación tenemos el programa **prog4-11.c** es el mismo que **prog4-5.c** pero modificado para mostrar el uso de los **if()** anidados.

```
/* prog4-11.c */
#include<stdio.h>
/* determina, a partir de dos cadenas de caracteres, cual es mayor */
main()
{
  char CADENA1[30],CADENA2[30];
  int RESULTADO;
  clrscr();
  printf("Dame la primer cadena \n");
  scanf(" %[^\n]",CADENA1);
  printf("Dame la segunda cadena \n");
  scanf(" %[^\n]",CADENA2);
```

```
  RESULTADO=strcmp(CADENA1,CADENA2);
  if(!RESULTADO)
  {
      printf("No se puede determinar cual cadena es mayor porque son iguales \n");
  }
  else
  {
      if(RESULTADO>0)
      {
         printf("La cadena1: %s es mayor a la cadena2: %s \n",CADENA1,CADENA2);
      }
      else
      {
         printf("La cadena2: %s es mayor a la cadena1:%s \n",CADENA2,CADENA1);
      } /* fin del if(RESULTADO>0) */
  } /* fin del if(!RESULTADO) */
  printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main() */
```

Continuando con los ejemplos de **if()** anidados tenemos el programa **prog4-12.c**, que es el mismo programa **prog4-8.c** pero modificado.

```
/* prog4-12.c */
#include<stdio.h>
/* determina si un numero es par o impar */
main()
{
   int NUMERO,RESULTADO;
   clrscr();
   printf("Dame un numero \n");
   scanf("%d",&NUMERO);
   if(NUMERO==0)
   {
       printf("No se puede determinar si el numero es par o impar porque es CERO \n");
   }
   else
   {
       RESULTADO=NUMERO%2;
       if(!RESULTADO)
       {
            printf("El numero %d es par \n",NUMERO);
       }
       else
       {
            printf("El numero %d es impar \n", NUMERO);
       } /* fin del if(!RESULTADO) */
   } /* fin del if(NUMERO) */
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Por otro lado, tenemos el programa **prog4-13.c** que determina el grado del acero cuando se conocen los valores para T1 Y T2. De esta manera, el acero se considera de grado 1 si T1 excede a 0.95 y T2 excede a 0.75; de grado 2 si T1 excede a 0.95 pero T2 no excede a 0.75; y de grado 3 si T1 no es mayor que 0.95.

```
/* prog4-13.c */
#include<stdio.h>
/* determina el grado del acero */
main()
{
   float T1,T2;
   clrscr();
   printf("Dame el valor de T1 \n");
   scanf("%f",&T1);
   printf("Dame el valor de T2 \n");
   scanf("%f",&T2);
```

```
   if(T1>0.95 && T2>0.75)
   {
       printf("El grado del acero es #1 \n");
   }
   else
   {
       if(T1>0.95 && T2<0.76)
       {
           printf("El grado del acero es #2 \n");
       } /* fin del if(T1>0.95 && T2<0.76) */
       else
       {
            if(T1<0.96)
           {
                printf("El grado del acero es #3 \n");
            } /* fin del if(T1<.096) */
       } /* fin del if(T1>0.95 && T2<0.76) */
   } /* fin del if(T1>0.95 && T2>0.75) */
printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
getche();
} /* fin de la funcion main() */
```

En el anterior programa **prog4-13.c** se utilizó, dentro de algunos **if()**, el operador lógico **AND** (doble amperson) **&&**, el cual se utiliza para comprobar una doble condición. No se debe confundir el amperson que se utiliza dentro del **scanf()** porque tienen diferente función.

En seguida tenemos el programa **prog4-14.c** que calcula los pagos de los recibos de luz en base a las siguientes tarifas:

14 KWH o menos	$30 pesos
Los siguientes 51 KWH	$0.50 por KWH
Exceso sobre 65 KWH	$0.25 por KWH

```
/* prog4-14.c */
#include<stdio.h>
/* calcula los pagos de los recibos de luz */
main()
{
   float KWH,PAGO,EXCESO;
   clrscr();
   printf("Dame los Kilowatts-hora consumidos\n");
   scanf("%f",&KWH);
   if(KWH<=14)
   {
       PAGO=30;
   }
   else
   {
       if(KWH>65)
       {
          EXCESO=KWH-65;
          PAGO=30+51*0.50+EXCESO*0.25;
       }
       else
       {
          EXCESO=KWH-14;
          PAGO=30+EXCESO*.50;
       }  /* fin del if(KWH>65) */
   } /* fin del if(KWH<=14) */
   printf("El pago debe ser de $%8.2f pesos\n",PAGO);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Para poner fin a este punto de los **if()** anidados, en el programa **prog4-15.c** se resuelve un problema para determinar si tres longitudes pueden formar un triángulo.

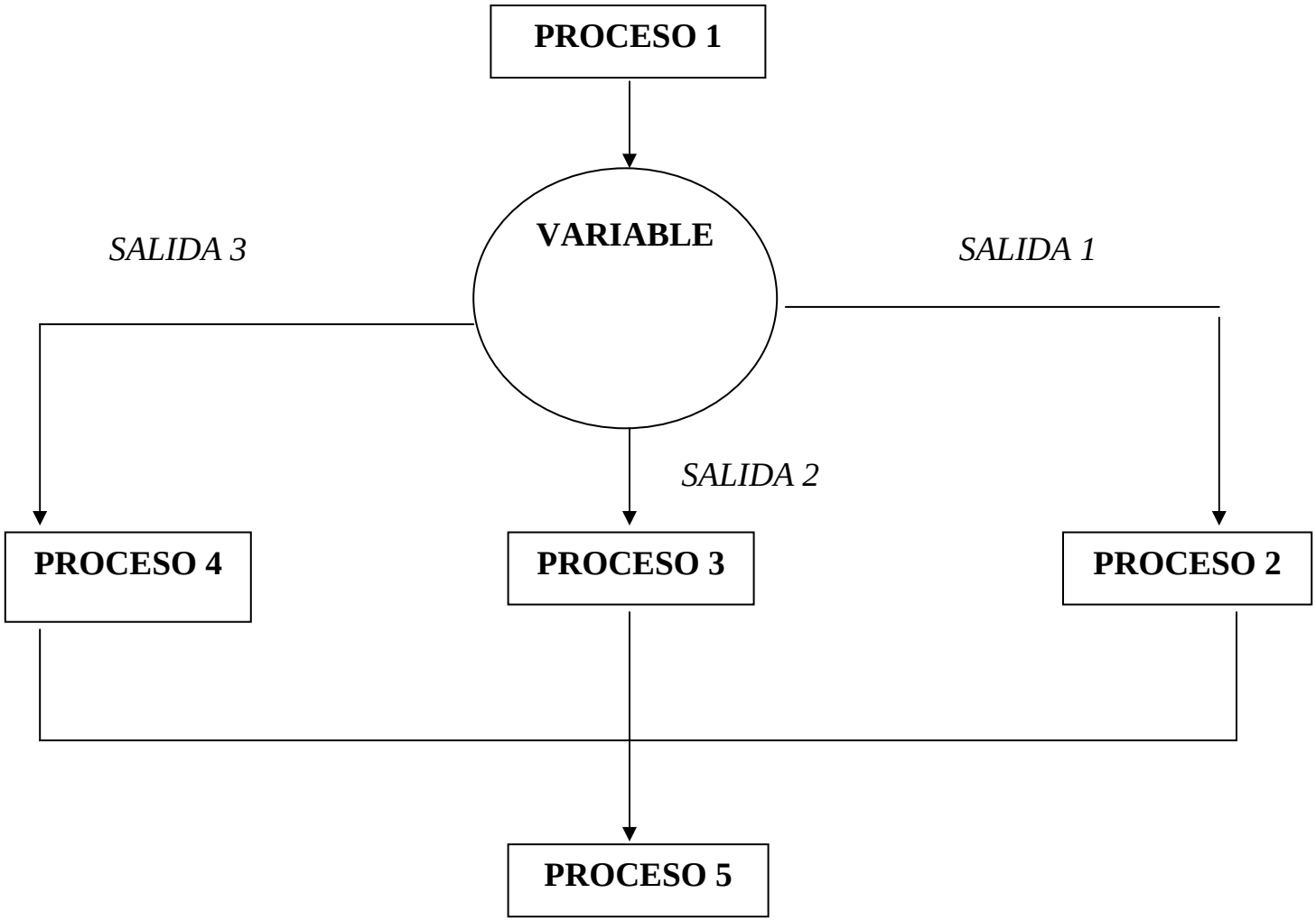
```
 /* prog4-15.c */
#include<stdio.h>
/* determina si tres lados puede formar un triangulo */
main()
{
 float LADO1,LADO2,LADO3,LADOS,LARGO;
 clrscr();
 printf("Dame lalongitud del lado 1\n");
 scanf("%f",&LADO1);
 printf("Dame lalongitud del lado 2\n");
 scanf("%f",&LADO2);
 printf("Dame lalongitud del lado 3\n");
 scanf("%f",&LADO3);
 if(LADO1!=0 && LADO2!=0 && LADO3!=0)
 {
   if(LADO1==LADO2)
   {
      if(LADO1>LADO3)
      {

         LADOS=LADO2+LADO3;
         LARGO=LADO1;
       }
       else
       {
         LADOS=LADO1+LADO2;
         LARGO=LADO3;
       } /* fin del if(LADO1>LADO3) */
   } /* fin del if(LADO1==LADO2) */

   if(LADO1==LADO3)
   {
      if(LADO1>LADO2)
      {

         LADOS=LADO2+LADO3;
         LARGO=LADO1;
       }
       else
       {
         LADOS=LADO1+LADO3;
         LARGO=LADO2;
       } /* fin del if(LADO1>LADO2) */
   } /* fin del if(LADO1==LADO3) */

   if(LADO2==LADO3)
   {
      if(LADO2>LADO1)
      {

         LADOS=LADO1+LADO3;
         LARGO=LADO2;
       }
       else
       {
         LADOS=LADO2+LADO3;
         LARGO=LADO1;
       } /* fin del if(LADO2>LADO1) */
   } /* fin del if(LADO2==LADO3) */


   if(LADO1!=LADO2 && LADO1!=LADO3 && LADO2!=LADO3)
   {
       if(LADO1>LADO2 && LADO1>LADO3)
       {
           LADOS=LADO2+LADO3;
           LARGO=LADO1;
       }
```

```
       else
       {
           if(LADO2>LADO1 && LADO2>LADO3)
           {
               LADOS=LADO1+LADO3;
               LARGO=LADO2;
           }
           else
           {
               if(LADO3>LADO1 && LADO3>LADO2)
               {
                     LADOS=LADO2+LADO1;
                     LARGO=LADO3;
               } /* fin del if(LADO3>LADO1 && LADO3>LADO2) */
           } /* fin del if(LADO2>LADO1 && LADO2>LADO3) */
       } /* fin del if(LADO1>LADO2 && LADO1>LADO3) */
  }/* fin del if(LADO1!=LADO2 && LADO1!=LADO3 && LADO2!=LADO3)*/

  if(LADOS>LARGO)
  {
    printf("Si pueden formar un triangulo \n");
  }
  else
  {
     printf("No pueden formar un triangulo \n");
  } /* if(LADOS>LARGO) */
 }
else
{
   printf("\n\nUno o mas de los numeros insertados son CERO\n");
   printf("por lo tanto no es triangulo\n");
} /* fin del if(LADO1!=0 && LADO2!=0 && LADO3!=0) */
printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
getche();
} /* fin de la funcion main() */
```

4.3 SENTENCIA switch()

La sentencia **switch()** se utiliza cuando tenemos dos opciones ó más a elegir, así como dos ó más posibilidades a escoger según un determinado problema.

La figura para representar la sentencia switch(), en diagramas de flujo, es la siguiente:



Nota: Todas las salidas se unen para seguir un mismo camino.

En el siguiente programa **prog4-16.c** podemos elegir entre varias opciones para calcular el área de diversas figuras geométricas.

```
/* prog4-16.c */
#include<stdio.h>
#include<math.h>
#define PI 3.1416
/* programa menu para obtener el area de diversas figuras */
main()
{
    int OPCION;
    float RADIO,AREA,LADO,BASE,ALTURA;
    clrscr();
    gotoxy(20,5); printf("Menu para obtener el Area de diversas figuras\n");
    gotoxy(10,8); printf("1.-Circulo");
    gotoxy(10,10); printf("2.-Esfera");
    gotoxy(10,12); printf("3.-Cuadrado");
    gotoxy(10,14); printf("4.-Triangulo");
    gotoxy(15,20); printf("Elija una opcion: ");
    scanf("%d",&OPCION);
    switch(OPCION)
    {
        case 1:
        {
            clrscr();
            printf("Dame el radio \n");
            scanf("%f",&RADIO);
            AREA=PI*pow(RADIO,2);
            printf("El area del circulo con radio=%8.2f es %8.2f\n",RADIO,AREA);
            break;
        } /* fin del case 1 */
        case 2:
        {
              clrscr();
              printf("Dame el radio \n");
              scanf("%f",&RADIO);
              AREA=4*PI*pow(RADIO,2);
              printf("El area de la esfera con radio=%8.2f es %8.2f\n",RADIO,AREA);
              break;
        } /* fin del case 2 */
        case 3:
        {
              clrscr();
              printf("Dame la longitud del lado \n");
              scanf("%f",&LADO);
              AREA=LADO*LADO;
              printf("El area del cuadrado con lado=%8.2f es %8.2f\n",LADO,AREA);
              break;
        } /* fin del case 3 */
        case 4:
        {
              clrscr();
              printf("Dame la longitud de la Base\n");
              scanf("%f",&BASE);
              printf("Dame la Altura\n");
              scanf("%f",&ALTURA);
              AREA=BASE*ALTURA/2.00;
              printf("El area del triangulo con base=%8.2f, altura=%8.2f es
%8.2f\n",BASE,ALTURA,AREA);
              break;
        } /* fin del case 4 */
        default:
        {
              printf("La opcion elegida no esta disponible\n");
        } /* fin del default */
    } /* fin del switch */
    printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
    getche();
} /* fin de la funcion main() */
```

4.4 PROBLEMAS PROPUESTOS.

Al realizar los siguientes programas establezca mecanismos para impedir el cálculo de operaciones matemáticas que arrojen resultados irreales o erróneos.

a) Auxiliado de las estructuras selectivas, modifique los programas del capítulo #3. Elimine algunas **deficiencias señaladas en el apartado correspondiente**. Tome como ejemplo, el código siguiente que se refiere **prog3-2.c**. Observe las modificaciones que se incluyeron, en el caso del **if(),** no permite calcular la conversión con valores negativos.

```
/* prog3-2.c */
/* Programa modificado para eliminar deficiencias utilizando if() */
#include<stdio.h>
/* convierte de pies a metros */
main()
{
   int PIES;
   float METROS;
   clrscr();
   printf("introduzca en numero de pies a convertir \n");
   scanf("%d", &PIES);
   if(PIES<=0)
   {
     printf("el numero insertado es cero o menor a cero \n");
     printf("no tiene sentido hacer conversiones con este numero\n");
     printf("si desea intentar de nuevo debera de ejecutar otra vez el programa\n");
   }
   else
   {
     METROS = PIES * 0.3084;     /* formula de conversion */
     printf("%d pies son %f metros \n", PIES, METROS);
   } /* fin del if */
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
 } /* fin de la funcion main() */
```

b) Usted conoce el espacio total en un disco duro y también conoce el espacio ocupado. Realice un programa que calcule el porcentaje de ocupación.

c) Usted está leyendo un libro. Realice un programa que le ayude a calcular el porcentaje de avance de lectura en cualquier momento.

d) Se ha dejado caer, desde el reposo, una billetera de una torre. Realice un programa que le ayude a calcular el porcentaje de distancia recorrido a los 2.3 segundos.

Los siguiente problemas propuestos son los mismos del capítulo 3, ahora deberá eliminar las deficiencias correspondientes, utilizando las estructuras selectivas.

e) Antes de despegar un avión, el piloto anuncia el tiempo estimado de vuelo en minutos, realice un programa que le ayude a determinar el porcentaje de avance del vuelo, teniendo como dato conocido al tiempo transcurrido del vuelo en minutos y el tiempo estimado de vuelo en minutos.

f) Un maestro desea determinar el porcentaje de aprobados y reprobados en un grupo; sólo sabe cuántos alumnos han aprobado y cuántos han reprobado. Realice un programa que le ayude a calcular estos porcentajes.

g) Una maestra midió la altura de Juanito al principio y al final de año escolar. Realice un programa que le ayude a determinar el porcentaje de crecimiento de Juanito.

Capítulo #5
Estructuras Repetitivas
for(), do{…}while() y while(){…}

Las estructuras repetitivas permiten que una serie de pasos, instrucciones o funciones se repitan con una secuencia predeterminada.

Existen dos clases de estructuras repetitivas, las que se **controlan por centinela y las que lo hacen por contador**. La diferencia es que en las estructuras controladas por centinela no se conoce de antemano las veces en que se repetirá un proceso y en las controladas por contador, esto sí se conoce.

Lenguaje C tiene dos funciones repetitivas que se controlan por centinela y son: **do{…}while() y while(){…}**, en las cuales **el proceso se repite mientras la condición es verdadera**, sin embargo, entre estas dos, la diferencia es que en el caso del **do{…}while()**, la condición para repetir el proceso está al final del mismo, y en el caso del **while() {…}**, la condición está el principio. Aquí surge la pregunta obvia, ¿Cuándo y en qué caso utilizar una ó la otra?. La respuesta es que se debe ver la situación particular de cada programa ya que en algunos casos es indiferente cuál se utilice, pero en otros, es más útil una que otra. Esto se puede determinar con la experiencia que cada programador tenga.

Por otro lado, **lenguaje C tiene una función que se controla por contador** en este caso es el **for()**.

Es necesario comentar que en una buena parte de los problemas a que se enfrenta un programador, se pueden utilizar ambas estructuras, es decir, las controladas por centinela o por contador, sin embargo, no son pocas las ocasiones en que sólo se puede utilizar una de las dos. Por ejemplo, analizaremos el problema a que se refiere el programa **prog5-1.c**.

Finalmente, antes de pasar a ejemplos de programas en este capítulo, haremos una importante aclaración. Esta es que, así como existen **if()**'s anidados, es decir un **if()** dentro de otro **if()**, también existen **for()** anidados, **do{…}while()** y **while(){…}** anidados etc. Es necesario tener esto en mente porque aunque en este folleto no halla programas que exhiban estos ejemplos, dichas estructuras anidadas existen y son de uso común.

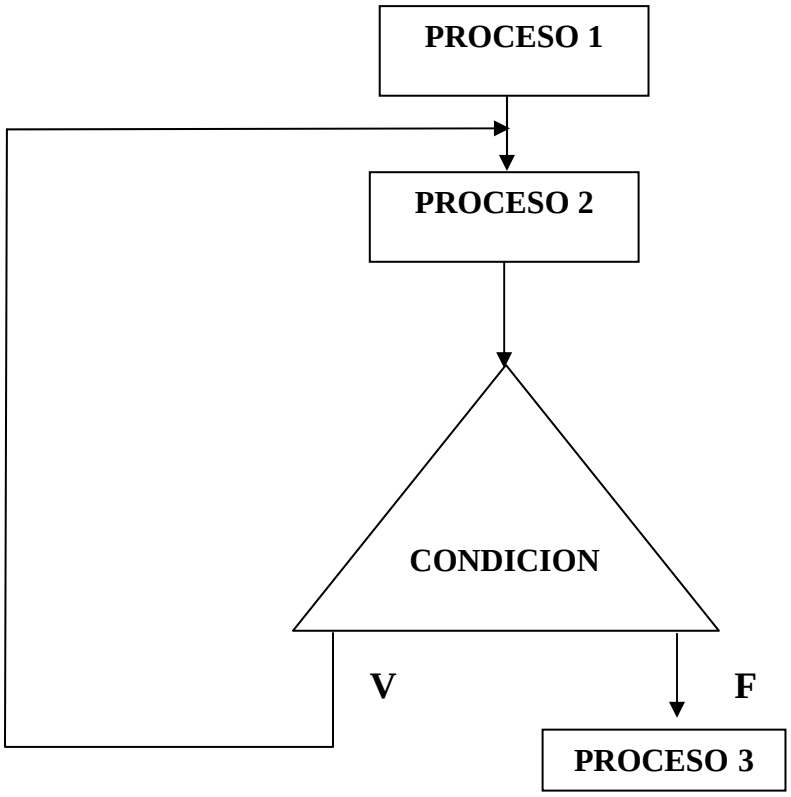
5.1 CICLO do{…}while(condición);

Esta sentencia es un ciclo repetitivo controlado por centinela donde la condición para que se repita el proceso se encuentra al final del mismo. Esta estructura se utiliza cuando no se conoce de antemano cuántas veces se repetirá un proceso que está inmerso en este ciclo. Es conveniente recordar que **el proceso se repetirá mientras la condición sea verdadera**. Por ejemplo:

```
do{
        Instrucción 1;
        Instrucción 2;
        Instrucción 3;
        Instrucción 4;
        Instrucción 5;
}while(centinela!=0)
```

El anterior ciclo puede entenderse así: Repita las instrucciones 1,2,3,4 y 5 mientras centinela sea diferente a cero.

La secuencia para representar la sentencia **do{…}while(condición);**, en diagramas de flujo, es la siguiente:



De la anterior figura cabe mencionar **que el falso y el verdadero NO se unen para continuar un solo camino**. Esta es la principal diferencia con el **if()**, que utiliza el mismo símbolo (un triángulo) pero en el **if(),** el falso y el verdadero sí se unen.

Es importante señalar que algunos autores prefieren utilizar un **triángulo** para señalar una estructura selectiva como lo es el **if()** y un **rombo** para referirse a una estructura repetitiva como lo son **do{…}while(condición) y while(condición){…},** sin embargo, esa distinción es innecesaria pues con la experiencia en la programación, se puede determinar de antemano, con relativa facilidad, cómo habrá de codificarse. Se recomienda analizar, dentro de este capítulo, cómo las estructuras selectivas y repetitivas se complementan para la validación de campos o variables.

Entonces comenzaremos con una serie de programas que utilicen el **do{…}while(condición);**

En el programa **prog5-1.c** se tiene la siguiente situación: Se deja caer una pelota desde una altura de determinada (en metros), rebota y cada vez su altura es de dos tercios de su altura en el rebote anterior. Elabore un programa que determine el número del rebote en el que la altura de la pelota es igual o inferior cincuenta centímetros. Además, deberá imprimir el número de cada rebote y su correspondiente altura.

Para resolver este problema, necesariamente utilizaremos una estructura de repetitiva controlada por centinela ya que no se puede utilizar la controlada por contador; de las dos disponibles utilizaremos **do{…}while()** por ser la que tiene la condición al final del proceso repetitivo y se ajusta más a nuestro problema.

Se utiliza una estructura controlada por centinela porque no sabemos cuántas veces se tendrá que calcular la nueva altura de la pelota después de cada rebote ya que esto depende de la altura de la que se deja caer la pelota. Por lo tanto, no sabemos cuántas veces se repetirá el proceso, entonces, para este caso en particular, utilizar una estructura controlada por centinela es la única opción para resolver el problema. Observe la siguiente codificación.

```
/* prog5-1.c*/
#include<stdio.h>
/* Pelota que rebota 2/3 partes de su altura anterior */
main()
{
  int REBOTE;
  float ALTURA;
  clrscr();
  REBOTE=0;
  printf("Dame la altura inicial de donde se deja caer la pelota, en metros\n");
  scanf("%f",&ALTURA);
  if(ALTURA>0)
  {
     printf("Rebote Altura\n");
     do{
          REBOTE=REBOTE+1;
          ALTURA=ALTURA*2/3;
          /* imprimir todos los rebotes con sus alturas */
          printf("%d   %8.2f\n",REBOTE,ALTURA);
        }while(ALTURA>0.50);
        printf("Despues del rebote # %d, la altura es %8.2f\n",REBOTE,ALTURA);
  }
  else
  {
     printf("La altura insertada es cero o menor a cero \n");
     printf("no tiene sentido hacer calculos con este numero \n");
     printf("si dese intentar de nuevo debera de ejecutar, otra vez, el programa\n");
  } /* fin del if */

  printf("\n\n pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
} /* fin de la funcion main */
```

Otro ejemplo lo tenemos en el programa **prog5-2.c** el cual es una adaptación del programa **prog3-5.c**., es este caso, al final del programa se pregunta si se desea continuar; si la respuesta es afirmativa deberá insertarse el número 1.

```
/* prog5-2.c */
#include <stdio.h>
#include <math.h>
/* calcula el area de triangulo en base a sus lados */
main()
{
  float LADO1,LADO2,LADO3,AREA,OPERACION,LADOS;
  int OPCION;
  do{
       clrscr();
       printf("Dame la longitud del lado # 1 \n");
       scanf("%f",&LADO1);
       printf("Dame la longitud del lado # 2 \n");
       scanf("%f",&LADO2);
       printf("Dame la longitud del lado # 3 \n");
       scanf("%f",&LADO3);
       LADOS=(LADO1+LADO2+LADO3)/2;
       OPERACION=LADOS*(LADOS-LADO1)*(LADOS-LADO2)*(LADOS-LADO3);
       AREA=sqrt(OPERACION);
       printf("El area del triangulo es %8.2f \n",AREA);
       printf("\n \n ¨Deseas analizar otro triangulo? \n");
       printf("Pulsa el numero 1 para continuar o cualquier otro numero para terminar\n");
       scanf("%d",&OPCION);
   }while(OPCION==1);
} /* fin de la funcion main() */
```

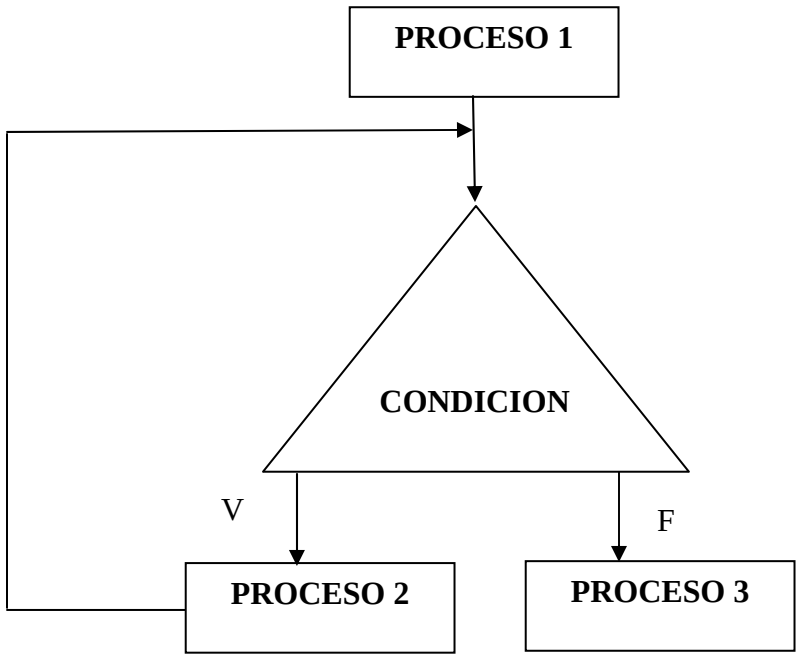
A continuación tenemos el programa **prog5-3.c**, que es una adaptación del programa **prog3-8.c**. En esta ocasión, también se pregunta al final del programa si se desea continuar y si la respuesta fuere afirmativa deberá insertarse una "S", ya sea mayúscula o minúscula. Por lo tanto, observe como en la condición del **while()** se colocó una doble condición de **"S" mayúscula ó "s" minúscula** con el objetivo de no generar errores cuando se pulse una u otra. Además, se debe negar la condición con (**!**) para que funcione adecuadamente

```
/* prog5-3.c */
#include <stdio.h>
#include <string.h>
/* convierte de grados fahrenheit a grados centigrados */
main()
{
   float CENT,FAHR;
   char OPCION[2];
   do{
       clrscr();
       printf("Dame los grados fahrenheit a convertir \n");
       scanf("%f",&FAHR);
       CENT=(5.0/9.0)*(FAHR-32);
       printf("%8.2f grados fahrenheit equivale %8.2f grados centigrados \n",FAHR,CENT);
       printf("¨Deseas continuar haciendo conversiones? \n");
    printf("\n\n Pulsa la letra 'S' para continuar o cualquier otra letra para terminar\n");
       scanf(" %[^\n]",OPCION);
   }while( !strcmp(OPCION,"S") || !strcmp(OPCION,"s"));
}/* fin de la funcion main() */
```

5.2 CICLO while(condición){ …}

El modo de operación de este ciclo es semejante al **do{…}while(condicion)**; . La diferencia está en el lugar que ocupa la condición. En el **do{…}while(condicion);** el proceso siempre se ejecutará por lo menos una vez porque la condición se encuentra al final del proceso. Por el contrario, con el ciclo **while(condición){…}** puede suceder que el proceso no se ejecute ni siquiera una vez porque la condición se encuentra al principio del proceso.

La secuencia para representar la sentencia **while(condición){...}**, en diagramas de flujo, es la siguiente:



Veamos el siguiente programa **prog5-4.c**. Observe como antes de **while(OPCION==1)** tuvimos que incluir la operación de asignación **OPCION=1**, esto con la finalidad de que el proceso se ejecute por lo menos una vez. Si no se hubiera incluido esta operación el proceso nunca se ejecutaría. Compare cuidadosamente con el Programa **prog5-2.c**.

```
/* prog5-4.c */
#include <stdio.h>
#include <math.h>
/* calcula el area de triangulo en base a sus lados */
main()
{
  float LADO1,LADO2,LADO3,AREA,OPERACION,LADOS;
  int OPCION;
  OPCION=1;
  while(OPCION==1)
  {
       clrscr();
       printf("Dame la longitud del lado # 1 \n");
       scanf("%f",&LADO1);
       printf("Dame la longitud del lado # 2 \n");
       scanf("%f",&LADO2);
       printf("Dame la longitud del lado # 3 \n");
       scanf("%f",&LADO3);
       LADOS=(LADO1+LADO2+LADO3)/2;
       OPERACION=LADOS*(LADOS-LADO1)*(LADOS-LADO2)*(LADOS-LADO3);
       AREA=sqrt(OPERACION);
       printf("El area del triangulo es %8.2f \n",AREA);
       printf("\n \n ¨Deseas analizar otro triangulo? \n");
       printf("Pulsa el numero 1 para continuar o cualquier otro numero para terminar\n");
       scanf("%d",&OPCION);
   } /* fin del while */
} /* fin de la funcion main() */
```

5.3 VALIDACIÓN DE CAMPOS O VARIABLES.

La validación de campo o variables es asegurarse que el ingreso de datos tiene un rango o forma que no puede violentarse porque entonces se cae en un error o inconsistencia. Por ejemplo, si estamos capturando calificaciones, éstas sólo deben de estar entre cero y cien, de otra forma se vuelve un dato incongruente para el contexto en el que se está utilizando.

Por otro lado, se mencionó que algunos autores prefieren utilizar un **triángulo** para señalar una estructura selectiva como lo es el **if()** y un **rombo** para referirse a una estructura repetitiva como lo son **do{…}while(condición); y while(condición){…}.**

Sin embargo, el autor de este folleto considera que esa distinción es innecesaria, pues con la experiencia en la programación, se puede determinar de antemano, con relativa facilidad, cómo habrá de codificarse dicho triángulo.

Por ejemplo, cuando en un triángulo de diagrama de flujo, el falso y el verdadero se unen, invariablemente la codificación será una estructura selectiva, es decir, es un **if()**, pero, cuando no se unen, la codificación corresponderá a una estructura repetitiva, en este caso, **do{…}while(condición); ó while(condición){…}.** Sin embargo, se debe ser muy cuidadoso, porque hay ocasiones en que un triángulo de condición deberá codificarse en ambas formas, esto es, como estructura selectiva y repetitiva.

Para sostener esta afirmación, se analizará el siguiente programa **prog5-5.c** que es el mismo que el **prog4-1.c**, sin embargo se han quitado algunas deficiencias.

En este caso se desea calcular el promedio de la calificación a partir de dos calificaciones parciales; el programa está mejorado pues no permitirá que se inserten, a la memoria RAM de la computadora, calificaciones menores a cero ó mayores a cien. Si se inserta una calificación inválida, el programa imprimirá en el monitor un mensaje de error y pedirá que se inserte otra calificación, de esta manera, el programa no avanzará hasta que se tecleé una calificación válida.

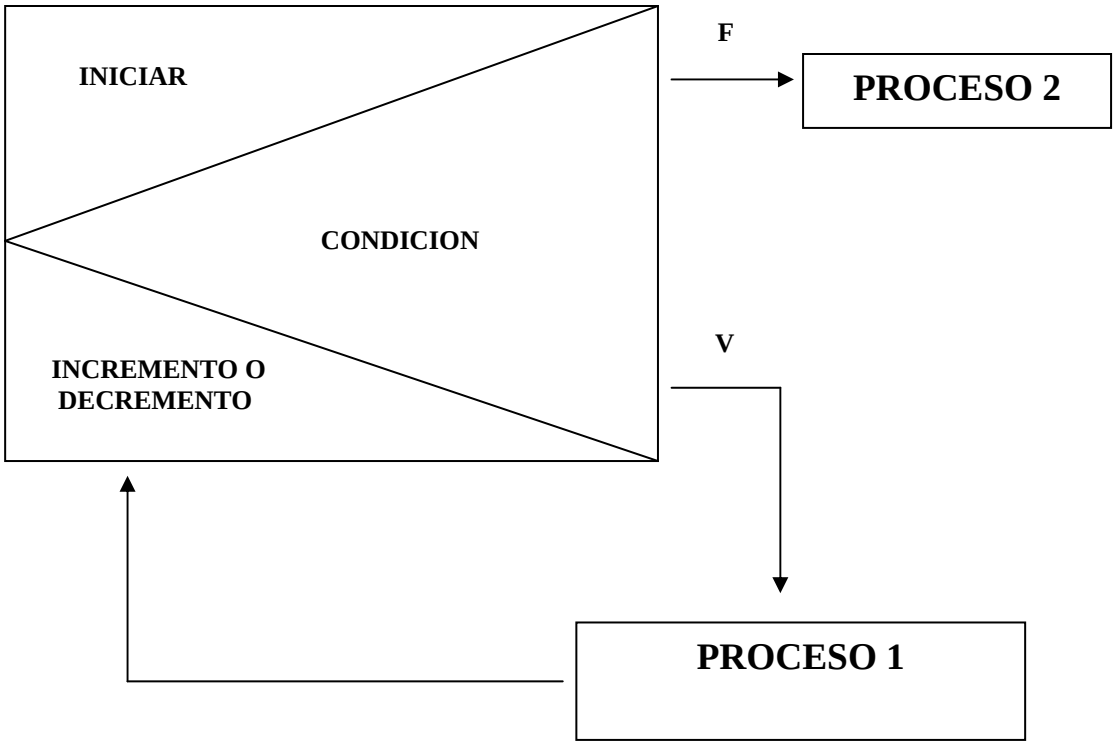
Observe la codificación y verá que se utilizó un **if()** para imprimir el mensaje de error y un **do{…}while(condición);** para que se pueda pedir nuevamente otra calificación.

```
/* prog5-5.c */
#include<stdio.h>
#include<string.h>
/* programa determina, en base a las calificaciones de los parciales,
   el promedio del alumno y si esta aprobado o reprobado */

/* En este programa se observa como se complementan las estructuras
   selectivas y repetitivas */

main()
{
   long int MATRICULA;
   char NOMBRE[30], SITUACION[30];
   float PARCIAL1, PARCIAL2, PROMEDIO;
   clrscr();
   printf("Dame la matricula \n");
   scanf("%ld",&MATRICULA);
   printf("Dame el nombre \n");
   scanf(" %[^\n]",NOMBRE);
   do{
       printf("Dame la calificacion del primer parcial \n");
       scanf("%f",&PARCIAL1);
       if(PARCIAL1<0 ||PARCIAL1>100)
       {
        printf("Calificacion del Primer parcial Invalida,");
        printf("favor de insertar otro numero\n\n");
       } /* fin del if del PARCIAL1 */
      }while(PARCIAL1<0 ||PARCIAL1>100);

    do{
       printf("Dame la calificacion del segundo parcial \n");
       scanf("%f",&PARCIAL2);
       if(PARCIAL2<0 ||PARCIAL2>100)
       {
        printf("Calificacion del Segundo parcial Invalida,");
        printf("favor de insertar otro numero\n\n");
       } /* fin de if del PARICIAL2 */
      }while(PARCIAL2<0 ||PARCIAL2>100);
   PROMEDIO=(PARCIAL1+PARCIAL2)/2;
   if(PROMEDIO>=70)
   {
      strcpy(SITUACION,"APROBADO");
   }
   else
   {
      strcpy(SITUACION,"REPROBADO");
   } /* fin del if */
   clrscr();
   printf("El alumno con matricula %ld Se llama %s \n",MATRICULA,NOMBRE);
   printf("tiene promedio de :%8.2f y esta %s\n", PROMEDIO,SITUACION);
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
} /* fin de la funcion main() */
```

Este mecanismo de validación se utiliza para impedir que se inserte a la computadora, información que no tiene sentido para una organización, por ejemplo, los meses del año sólo pueden ser del 1 al 12; el año de nacimiento de una persona no puede ser mayor al año que transcurre; la edad de una persona no puede ser negativa ni más grande de lo lógicamente aceptable, etc.

5.4 SENTENCIA for()

La función **for()** es una estructura repetitiva que se controla por contador y **la figura que se utiliza para representarla en diagramas de flujo es:**



Es conveniente recordar que en las estructuras controladas por contador, **si se conoce de antemano las veces en que un proceso se repetirá, y sólo se repetirá mientras la condición es verdadera**.

La codificación de la anterior figura es:

```
for(INICIAR; CONDICION; INCREMENTO o DECREMENTO)
{
  PORCESO1;
} /* fin del for */
```

Observe que INICIAR está separada de CONDICION con un punto y coma, lo mismo sucede con CONDICION e, INCREMENTO o DECREMENTO

Así mismo, se puede apreciar un apartado de iniciar que es la parte donde se coloca el valor inicial del contador. En el apartado de condición se coloca el número de veces en que se repetirá el proceso1. El apartado de incremento o decremento indica el número en que el contador, se incrementará o decrementará, con cada repetición de Proceso1. Por último, el Proceso2 se realizará después de que se haya cumplido la condición. Por ejemplo: La siguiente instrucción:

```
for(Contador=1; Contador<25 ; Contador++)
{
        instrucción 1
        instrucción 2
        instrucción 3
        instrucción 4
        instrucción 5
} /* fin del for */
```

El anterior ciclo puede leerse de la siguiente forma: Realiza las instrucciones 1,2,3,4,y 5 desde que el contador vale uno mientras que sea menor a veinticinco, con incrementos de uno.

Con el programa **prog5-6.c** resolverá la siguiente ecuación cuadrática para cuando "X" toma diez valores diferentes a partir que X=15 con incrementos de 3. La ecuación es la siguientes: $Y=6X^2+4X-3$, en este programa se utiliza la función for(), que es una estructura repetitiva controlada por contador.

```
/* prog5-6.c */
#include<stdio.h>
#include<math.h>
/* imprime la tabulacion de la ecuacion cuadratica */
/* y=6x2+4x-3 */
main()
{
  float x,y;
  int I;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Tabular la ecuacion cuadratica y=6x2+4x-3 \n");
  printf("introduzca el valor de X \n");
  scanf("%f",&x);
  for(I=1;I<=10;I++)
  {
      y=6*pow(x,2)+4*x-3;  /* ecuacion cuadratica a tabular */
      printf("Cuando X=%8.2f, Y= %8.2f \n", x,y);
      x=x+3;
  } /* fin del for */
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

Con el programa **prog5-7.c** se resolverá el programa anterior pero ahora se utilizará una estructura repetitiva controlada por centinela.

```
/* prog5-7.c */
#include<stdio.h>
#include<math.h>
/* imprime la tabulacion de la ecuacion cuadratica */
/* y=6x2+4x-3 */
main()
{
  float x,y;
  int I;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Tabular la ecuacion cuadratica y=6x2+4x-3 \n");
  printf("introduzca el valor de X \n");
  scanf("%f",&x);
  I=1;
  do{
      y=6*pow(x,2)+4*x-3;  /* ecuacion cuadratica a tabular */
      printf("Cuando X=%8.2f, Y= %8.2f \n",x,y);
      x=x+3;
      I=I+1;
     }while(I<=10);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

Con los programas **prog5-8.c** y **prog5-9.c** vamos a determinar la sumatoria de los primeros cien números enteros, es decir, del uno al cien. En el **prog5-8.c** se utiliza una estructura repetitiva controlada por contador y en el **prog5-9.c** se utiliza una controlada por centinela.

```
/* prog5-8.c */
#include<stdio.h>
/* suma los primeros cien numeros enteros */
main()
{
  int SUMA,I;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Calcular la sumatoria del 1 al 100 \n");
  SUMA=0;
  for(I=1;I<=100;I++)
  {
      SUMA=SUMA+I;
  } /* fin del for */
  printf("La sumatoria es %d\n",SUMA);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

```
/* prog5-9.c */
#include<stdio.h>
/* suma los primeros cien numeros enteros */
main()
{
  int SUMA,I;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Calcular la sumatoria del 1 al 100 \n");
  SUMA=0;
  I=1;
  do{
      SUMA=SUMA+I;
      I=I+1;
    }while(I<=100);
  printf("La sumatoria es %d\n",SUMA);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

Como se verá, éstos últimos programas pueden resolverse con estructuras repetitivas controladas por contador y por centinela, esto no siempre es así, por lo que el alumno deberá aprender a utilizar ambas estructuras.

Los programas del **prog5-10.**c al **prog5-15.c**, resolverán algunos problemas con la estructura repetitiva controlada por contador, es decir, el **for()**, se dejará que el alumno practique haciéndolos de nuevo con la estructura controlada por centinela, si es que esto fuera posible.

El programa **prog5-10.c** permite leer un valor para “A”, luego calcula e imprime la suma de los ocho números: 1, 1+A, 1+2A, 1+3A…, 1+7A .

```
/* prog5-10.c */
#include<stdio.h>
/* suma 1,1+A, 1+2A, 1+3A+...+1+7A */
main()
{
  int SUMA,I,A;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Calcular la sumatoria 1,1+A, 1+2A, 1+3A+...+1+7A \n");
  printf("Dame el Valor de A\n");
  scanf("%d",&A);
  SUMA=1;
  for(I=1;I<=7;I++)
  {
      SUMA=SUMA+(1+I*A);
      printf("Cuando I=%d, SUMA=%d\n",I,SUMA);
  } /* fin del for */
  printf("La sumatoria es %d\n",SUMA);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

El programa **prog5-11.c** calcula el promedio de “N” números.

```
/* prog5-11.c */
#include<stdio.h>
/* obtiene el promedio de N numeros */
main()
{
  int I,N;
  float SUMA,NUMERO,PROMEDIO;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Obtener el promedio de N numeros \n");
  printf("¨Cuantos numeros son?\n");
  scanf("%d",&N);
  SUMA=0;
  for(I=1;I<=N;I++)
  {
      printf("Dame el numero %d: \n",I);
      scanf("%f",&NUMERO);
      SUMA=SUMA+NUMERO;
  } /* fin del for */
  PROMEDIO=SUMA/N;
  printf("Sumatoria=%8.2f y el promedio= %8.2f\n",SUMA,PROMEDIO);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

El programa **prog5-12.c** determina si un número es primo o no.

```
/* prog5-12.c */
#include<stdio.h>
/* Determinar si un numero es primo */
main()
{
  int I,N,NUMERO,PRIMO,RESIDUO;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Determinar si un numero es primo \n");
  printf("Dame el numero a analizar \n");
  scanf("%d",&NUMERO);
  RESIDUO=0;
  N=NUMERO/2;
  for(I=2;I<=N;I++)
  {
      RESIDUO=NUMERO%I;
      printf("Cuando I=%d, Residuo=%d \n",I,RESIDUO);
      if(RESIDUO==0)
      {
       I=N;
       PRIMO=0;
      } /* fin del if RESIDUO==0 */
  } /* fin del for */
  if(PRIMO==0)
  {
     printf("\n El Numero=%d No es primo \n",NUMERO);
  }
  else
  {
     printf("\n El Numero=%d Si es primo \n",NUMERO);
  } /* fin del if PRIMO==0 */
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

El programa **prog5-13.c** calcula el factorial de un número.

```
/* prog5-13.c */
#include<stdio.h>
/* Calcular el factorial de un numero */
main()
{
  int I,NUMERO;
  long int FACT;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Calcular el factorial de un numero \n");
  printf("Dame el numero \n");
  scanf("%d",&NUMERO);
  FACT=1;
  for(I=2;I<=NUMERO;I++)
  {
    FACT=FACT*I;
  } /* fin del for */
  printf("\n El Factorial de %d es: %ld \n",NUMERO,FACT);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

El programa **prog5-14.c** permite imprimir la tabla de multiplicar de cualquier número.

```
/* prog5-14.c */
#include<stdio.h>
/* Imprime la tabla de multiplicar de un numero */
main()
{
  int I,TABLA,RES;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Imprimir la tabla de multiplicar de un numero \n");
  printf("Dame el numero \n");
  scanf("%d", &TABLA);
  for(I=1;I<=10;I++)
  {
    RES=TABLA*I;
    printf("\n %d * %d = %d \n",TABLA,I,RES);
  } /* fin del for */
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

El programa **prog5-15.c** permite determinar el elemento mayor de una lista de “N” números.

```
/* prog5-15.c */
#include<stdio.h>
/* Determinar el elemento mayor de una lista de N numeros */
main()
{
  int I,N;
  float NUMERO,MAY;
  clrscr(); /* limpiar la pantalla de ejecucion */
  printf("Determinar el elemento mayor de una lista de N numeros \n");
  printf("¨¿Cuantos numeros son?\n");
  scanf("%d",&N);
  printf("Dame el elemento # 1: \n");
  scanf("%f",&NUMERO);
  MAY=NUMERO;
  for(I=2;I<=N;I++)
  {
    printf("Dame el elemento # %d: \n",I);
    scanf("%f",&NUMERO);
    if(NUMERO>MAY)
    {
       MAY=NUMERO;
    } /* fin del if */
  } /* fin del for */
  printf("\n El elemento mayor es: %8.2f \n",MAY);
  printf("\n Pulse cualquier tecla para regresar a la pantalla de edicion");
  getche();
}/* fin de la funcion main */
```

A continuación tenemos el programa **prog5-16.c** que es el mismo de **prog5-5.c**, pero ahora se puede calcular el promedio de varios alumnos en una misma corrida del programa.

```
/* prog5-16.c */
#include<stdio.h>
#include<string.h>
/* programa que determina, en base a las calificaciones de los parciales,
el promedio de varios alumnos y si estan aprobados o reprobados */
main()
{
   long int MATRICULA;
   char NOMBRE[30], SITUACION[30];
   float PARCIAL1, PARCIAL2, PROMEDIO;
   int I,N;
   clrscr();
   printf("¨Cuantos alumnos son?\n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
       clrscr();
       printf("Introduzca los datos del alumno # %d \n",I);
       printf("Dame la matricula \n");
       scanf("%ld",&MATRICULA);
       printf("Dame el nombre \n");
       scanf(" %[^\n]",NOMBRE);
       do{
         printf("Dame la calificacion del primer parcial \n");
         scanf("%f",&PARCIAL1);
         if(PARCIAL1<0 || PARCIAL1>100)
         {
            printf("Calificacion del Primer parcial Invalida,");
            printf("favor de insertar otro numero\n\n");
         } /* fin del if del PARCIAL1 */
       }while(PARCIAL1<0 || PARCIAL1>100);
       do{
         printf("Dame la calificacion del segundo parcial \n");
         scanf("%f",&PARCIAL2);
         if(PARCIAL2<0 || PARCIAL2>100)
         {
            printf("Calificacion del Segundo parcial Invalida,");
            printf("favor de insertar otro numero\n\n");
         } /* fin del if del PARCIAL2 */
       }while(PARCIAL2<0 || PARCIAL2>100);
       PROMEDIO=(PARCIAL1+PARCIAL2)/2;
       if(PROMEDIO>=70)
       {
            strcpy(SITUACION,"APROBADO");
       }
      else
      {
            strcpy(SITUACION,"REPROBADO");
      } /* fin del if */
      printf("El alumno %s tiene promedio:%8.2f esta %s\n", NOMBRE,PROMEDIO,SITUACION);
      printf("\n \n Pulse cualquier tecla para continuar\n");
      getche();
   } /* fin del for */
   printf("\n \n Pulse cualquier tecla para regresar a la pantalla de edicion\n");
   getche();
} /* fin de la funcion main() */
```

El siguiente programa es el **prog5-17.c** que es una adaptación del **prog4-14.c**, pero ahora se puede calcular el pago de varios recibos en una misma corrida del programa.

```
/* prog5-17.c */
#include<stdio.h>
/* calcula los pagos de los recibos de luz */
main()
{
   float KWH,PAGO,EXCESO;
   int I,N;
   clrscr();
   printf("¨Cuantos recibos de luz son? \n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
      clrscr();
      printf("Dame los Kilowatts-hora consumidos por el recibo #%d\n",I);
      scanf("%f",&KWH);
      if(KWH<=14)
      {
         PAGO=30;
      }
      else
      {
         if(KWH>65)
         {
             EXCESO=KWH-65;
             PAGO=30+51*0.50+EXCESO*0.25;
         }
         else
         {
             EXCESO=KWH-14;
             PAGO=30+EXCESO*.50;
         }  /* fin del if(KWH>65) */
      } /* fin del if(KWH<=14) */
      printf("El pago debe ser de $%8.2f pesos\n",PAGO);
      printf("\n \n Pulse cualquier tecla para continuar \n");
      getche();
   } /* fin del for */
} /* fin de la funcion main() */
```

5.5 LA SENTENCIA break;

La sentencia **break**; se utiliza para terminar la ejecución de ciclos o salir de una sentencia **switch()**. Se puede utilizar dentro de las sentencias **do{…}while(),while(), for()** o **switch()**.

5.6 LA SENTENCIA continue;

La sentencia **continue**; se utiliza para saltarse el resto de la pasada actual a través de un ciclo. El ciclo no termina cuando se encuentra una sentencia **continue**, sencillamente no se ejecutan las sentencias que se encuentran a continuación en él y se salta directamente a la siguiente pasada a través del ciclo. Se puede utilizar dentro de una sentencia **do{…}while(),while(), for()** o **switch()**..

5.7 PROBLEMAS PROPUESTOS.

a) Loenardo Fibonacci, un rico comerciante italiano del siglo XIII, introdujo una serie de números conocidos hoy en día como números de Fibonacci. Los primeros 16 de ellos son: 1,1,2,3,5,8,13,21,34,55,89,144,233,377,610. Cada número de una secuencia de números de fibonacci es la suma de los dos números que preceden inmediatamente al número considerado. Esto es:
1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21 …etc.
Esta secuencia tiene aplicaciones prácticas en botánica, teoría de redes eléctricas y otros campos. Realice el programa que determine los primeros 50 números de la serie de Fibonacci.

b) Realice un programa que calcule la sumatoria de la siguiente serie:
C=Σ ai bi= a1b1+a2b2+a3b3+a4b4, desde que i=1 hasta que i=4.

c) Una dama de edad avanzada desea pintar el piso del kiosco sin desperdiciar nada de pintura. Ella sabe por experiencia que se necesita un cuarto de pintura para cubrir 37 pies cuadrados de área. Si el piso del kiosco tiene diez pies de diámetro ¿qué cantidad de pintura debería comprar?. Realice un programa que resuelva el anterior problema.

d) Realice un programa que calcule los valores desde que N=1 hasta N=20 en incrementos de uno e imprima encabezados de lo siguiente:
N N^2 $\sqrt{N}$ $\sqrt{10N}$ N^3 $\sqrt[3]{N}$

e) Realice un programa que calcule la depreciación de un automóvil
Valor del Auto=$ 1,800,000
Vida=6
Rescate=$ 120,000
Imprima los siguientes encabezados
"Año Depreciación Depreciación Acumulada Valor Anual"
La fórmula de la depreciación anual constate para cada año de vida útil

$$\text{Depreciación} = \frac{\text{Costo} - \text{Valorde Re scate}}{\text{VidaUtil}}$$

f) En la fábrica de sillas Bibliomodel, el pago a sus empleados está basado en una tarifa diaria más un incentivo, el cual depende del número de sillas producidas durante el día. Calcule el salario de cada empleado. Por ejemplo, a un salario básico de $3.50 dólares por hora y con 60 centavos de incentivo por cada silla producida por encima de 50 unidades, un empleado que ensamble 76 sillas recibirá:
(3.50)(8)+(76-50)(0.60)=28.00+15.60=$43.60

g) Dado un conjunto de números (mínimo 20, máximo 50), realice un programa que:
1. Sume los números dados.
2. Muestre cuál es el número menor.
3. Muestre cuál es el número mayor.
4. Muestre el cantidad de valores iguales a 25.
5. Muestre la cantidad de valores mayores a 20 y menores a 70.

h) Utilizando el teorema de Pitágoras, realice un programa que determine las tercia de números menores a cien y que formen un triángulo rectángulo, por ejemplo, 05-12-13, 40-50-30 y 26-24-10, son tercias de números que cumplen dicha condición.

i) Asuma que ha invertido $1,000 pesos en una caja popular. La junta directiva le ha asegurado que usted duplicará su inversión en cada dos años. Elabore un programa que calcule su inversión cada año. El procedimiento deberá imprimir una lista semejante a la siguiente:

2 años	$ 200
4 años	$ 400
6 años	$ 800
8 años	$ 1,600
10 años	$ 3,200
12 años	$ 6,400
14 años	$12,800

j) Realice un programa dirigido a los niños de educación primaria. El programa deberá hacer diez preguntas sencillas de historia, biología o matemáticas. Así mismo, en cada pregunta se deberán establecer por lo menos 4 opciones de respuesta, si se inserta una opción invalida, el programa deberá advertir al niño para que escoja sólo entre las opciones ofrecidas. Después de cada respuesta, el programa deberá mostrar un registro de la cantidad de preguntas contestadas correctamente, de las incorrectas y de las que faltan por contestar. Al terminar deberá imprimir una calificación.

k) El número de sonido emitidos por un grillo en un minuto es una función de la temperatura. Como resultado de esto, ¡es posible determinar el nivel de temperatura utilizando al grillo como termómetro!. La fórmula de la función es: $t = \frac{N}{4} + 40$, donde *t* representa la temperatura en grados Fahrenheit y *n* representa el número de sonidos emitidos en un minuto. Elabore un programa que determine e imprima los valores para *t* cuando *n* toma los valores de 40,50,60,70,…,140,150.

l) El tiempo que requiere un satélite para dar una revolución completa alrededor de la tierra y a una determinada altura es una función de su velocidad. La fórmula para una altitud de 100 millas es $t = \frac{1.540}{s}$, donde *t* es el tiempo y *s* es la velocidad del satélite en miles de millas por hora. Elabore un programa que calcule e imprima *t* para los valores de *s* 18,19,20,…24.

Capítulo #6
Funciones Definidas por el Usuario sin Parámetros

Lenguaje C es un lenguaje orientado a funciones, de tal forma que **printf(), scanf(), for(), if(), switch(), do{…}while(), while(), getche(), main(), pow(), sqrt()**, etc. son funciones. A estas funciones se les llama predeterminadas porque ya existen en lenguaje C.

De esta manera, lenguaje C permite que el programador haga sus propias funciones, a éstas les llama funciones definidas por el usuario. Así mismo, dentro de estas funciones hay de dos tipos, las que tienen y las que no tienen parámetros.

En este capítulo utilizaremos las funciones definidas por el usuario sin parámetros por ser las más sencillas y fáciles, es recomendable que los alumnos interesados, estudien por cuenta propia las funciones definidas por el usuario con parámetros.

En el siguiente programa **prog6-1.c** que es el mismo que el **prog4-16.c** pero ahora está hecho con funciones. Aquí podemos encontrar las siguientes cosas:

En los programas anteriores las variables han sido declaradas dentro de la función **main()**, en este programa están declaradas afuera y para ser precisos antes de la función **main()**, cuando esto sucede a estas variables se les llama **variables globales**. Por lo tanto cuando, hay variables declaradas dentro de una función se les llama **variables locales** a la función.

En resumen, **en un programa puede haber variables locales y globales**. Su función, su uso y manejo tienen una diferencia notable por lo que es conveniente que los alumnos interesados en este tema lo estudien por cuenta propia.

Por otro lado, cuando se manejan funciones se deben considerar tres cosas: **La declaración (prototipo) de la función, la llamada a la función y la definición de la función**. Así mismo, se utilizaron líneas de comentario para identificar cada una de estas tres partes. Observe que tanto en la declaración como en la definición de la función se anota **void**, lo cual indica que las funciones no contienen parámetros, también se dice que estas funciones no retornan ningún valor.

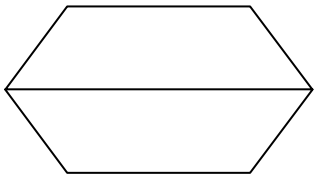
En la **definición de la función** se escriben las instrucciones que están dentro de la función, a esto también se le llama **cuerpo de la función**, estas instrucciones se ejecutarán cuando se llame a la función. Al final del cuerpo de la función se escribe la sentencia **return;** que indica el regreso a la posición de donde fue llamada la función.

Compare el programa **prog6-1.c** con el programa **prog4-16.c** para que observe las diferencias entre un programa con y sin funciones.

```
/* prog6-1.c */
#include<stdio.h>
#include<math.h>
#define PI 3.1416
/* programa menu para obtener el area de diversas figuras utilizando funciones*/
/* declaracion de las funciones */
void menu();
void circulo();
void esfera();
void cuadrado();
void triangulo();
/* declaracion de variables globales */
int OPCION;
float RADIO,AREA,LADO,BASE,ALTURA;
main()
{
   menu(); /* llamada a la funcion menu() */
   getche();
} /* fin de la funcion main() */

/* definicion de funciones */
void menu()
{
    clrscr();
    gotoxy(20,5); printf("Menu para obtener el Area de diversas figuras\n");
    gotoxy(10,8); printf("1.-Circulo");
    gotoxy(10,10); printf("2.-Esfera");
    gotoxy(10,12); printf("3.-Cuadrado");
    gotoxy(10,14); printf("4.-Triangulo");
    gotoxy(15,20); printf("Elija una opcion: ");
    scanf("%d",&OPCION);
    switch(OPCION)
    {
        case 1:
        {
           circulo(); /* llamada a la funcion circulo() */
           break;
        } /* fin del case 1 */
        case 2:
        {
           esfera(); /* llamada a la funcion esfera() */
           break;
        } /* fin del case 2 */
        case 3:
        {
           cuadrado(); /* llamada a la funcion cuadrado() */
           break;
        } /* fin del case 3 */
        case 4:
        {
           triangulo(); /* llamada a la funcion triangulo() */
           break;
        } /* fin del case 4 */
        default:
        {
            printf("La opcion elegida no esta disponible\n");
        } /* fin del default */
    } /* fin del switch */
    return;
} /* fin de la funcion menu() */

void circulo()
{
    clrscr();
    printf("Dame el radio \n");
    scanf("%f",&RADIO);
    AREA=PI*pow(RADIO,2);
    printf("El area del circulo con radio=%8.2f es %8.2f\n",RADIO,AREA);
    return;
} /* fin de la funcion circulo() */
void esfera()
```

```
{
    clrscr();
    printf("Dame el radio \n");
    scanf("%f",&RADIO);
    AREA=4*PI*pow(RADIO,2);
    printf("El area de la esfera con radio=%8.2f es %8.2f\n",RADIO,AREA);
    return;
} /* fin de la funcion esfera */


void cuadrado()
{
    clrscr();
    printf("Dame la longitud del lado \n");
    scanf("%f",&LADO);
    AREA=LADO*LADO;
    printf("El area del cuadrado con lado=%8.2f es %8.2f\n",LADO,AREA);
    return;
} /* fin de la funcion cuadrado */


void triangulo()
{
    clrscr();
    printf("Dame la longitud de la Base\n");
    scanf("%f",&BASE);
    printf("Dame la Altura\n");
    scanf("%f",&ALTURA);
    AREA=BASE*ALTURA/2.00;
    printf("El area del triangulo con base=%8.2f, altura=%8.2f es
%8.2f\n",BASE,ALTURA,AREA);
    return;
} /* fin de la funcion triangulo */
```

Las funciones pueden ser concebidas como programas pequeños dentro de un programa o mejor dicho subprogramas. **El símbolo que se utiliza para representar a una función es el siguiente:**



Programa 6.2

El Señor López ha recibido una oferta de empleo de la Compañía de juguetes “Ensueño” con la oportunidad elegir entre dos sistemas de pago. Puede recibir un salario mensual de $5,000 y $50 de aumento cada mes, o puede recibir un salario mensual de $5,000 con un aumento anual de $800. Realice un programa que determine el salario mensual para los siguientes tres años. El programa debe determinar los salarios acumulados después de cada mes y a partir de esta información determinar el mejor método de pago.

```
/* prog6-2.c */
```

```
#include<stdio.h>
/* Determinar cual opcion de salario es mejor */

/* declaracion de las funciones */
void datos();
void opcion_1();
void opcion_2();
void decision();


/* declaracion de variables globales */
float SALARIO_MENSUAL, SALARIO_MENSUAL2, AUMENTO_MENSUAL, AUMENTO_ANUAL;
float TOTAL_1, TOTAL_2,MESES;
int MES,CAMBIO_ANUAL, PERIODO;

main()
{
   TOTAL_1=0;
   TOTAL_2=0;
   datos();     /* llamada a la funcion datos() */
   opcion_1();  /* llamada a la funcion opcion_1() */
   opcion_2();  /* llamada a la funcion opcion_2() */
   decision();  /* llamada a la funcion decision() */
   printf("\n\n Pulse cualquier tecla para regresar a la pantalla de edicion\n");
   getche();
} /* fin de la funcion main() */

/* definicion de funciones */
void datos()
{
  do{
     clrscr();
     printf("Dame el salario mensual\n");
     scanf("%f",&SALARIO_MENSUAL);
     printf("Dame el aumento mensual\n");
     scanf("%f",&AUMENTO_MENSUAL);
     printf("Dame el aumento anual\n");
     scanf("%f",&AUMENTO_ANUAL);
     printf("¨A cuantos a¤os desea hacer el calculo?\n");
     scanf("%d",&PERIODO);
     SALARIO_MENSUAL2=SALARIO_MENSUAL;
     MESES=PERIODO*12;
     if(SALARIO_MENSUAL<=0 || AUMENTO_MENSUAL<=0 || AUMENTO_ANUAL<=0 || PERIODO<=0)
     {
       printf("\n\n Uno o mas datos insertados NO son validos,");
       printf("\n favor de revisarlos e intentar de nuevo\n");
       printf(" Pulse cualquier tecla para continuar\n");
       getche();
     } /* fin del if */
  }while(SALARIO_MENSUAL<=0 || AUMENTO_MENSUAL<=0 || AUMENTO_ANUAL<=0 || PERIODO<=0);
} /* fin de la funcion datos */

void opcion_1()
{
  clrscr();
  printf("Calculo de la Primera Opcion\n");
  printf("Mes   Salario Mensual  Acumulado\n");
  for(MES=1;MES<=MESES;MES++)
  {
      TOTAL_1=TOTAL_1+SALARIO_MENSUAL;
      printf("%d    %8.2f          %8.2f\n",MES,SALARIO_MENSUAL,TOTAL_1);
      SALARIO_MENSUAL=SALARIO_MENSUAL+AUMENTO_MENSUAL;
  } /* fin del for */
  printf("OPCION 1: TOTAL DE SUELDO FINAL DEL A¥O %d :$%12.2f \n",PERIODO,TOTAL_1);
  printf("Pulse cualquier tecla para continuar\n");
  getche();
  return;
} /* fin de la funcion opcion_1 */


void opcion_2()
```

```
{
  clrscr();
  printf("Calculo de la Segunda Opcion\n");
  printf("Mes   Salario Mensual  Acumulado\n");
  for(MES=1;MES<=MESES;MES++)
  {
      CAMBIO_ANUAL=CAMBIO_ANUAL+1;
      if(CAMBIO_ANUAL==13)
      {
       SALARIO_MENSUAL2=SALARIO_MENSUAL2+AUMENTO_ANUAL;
       CAMBIO_ANUAL=1;
      } /* fin del if */
      TOTAL_2=TOTAL_2+SALARIO_MENSUAL2;
      printf("%d    %8.2f          %8.2f\n",MES,SALARIO_MENSUAL2, TOTAL_2);
  } /* fin del for */
  printf("OPCION 2: TOTAL DE SUELDO FINAL DEL A¥O %d :$%12.2f \n",PERIODO,TOTAL_2);
  printf("Pulse cualquier tecla para continuar\n");
  getche();
  return;
} /* fin de la funcion opcion_2 */

void decision()
{
  clrscr();
  printf("OPCION 1: TOTAL DE SUELDO FINAL DEL AÑO %d :$%12.2f \n",PERIODO,TOTAL_1);
  printf("OPCION 2: TOTAL DE SUELDO FINAL DEL AÑO %d :$%12.2f \n",PERIODO,TOTAL_2);
  if(TOTAL_1==TOTAL_2)
  {
     printf("\n\n Ninguna opcion es mejor que la otra\n");
  }
  else
  {
     if(TOTAL_1>TOTAL_2)
     {
       printf("\n\n La mejor opcion es la numero 1\n");
     }
     else
     {
       printf("\n\n La mejor opcion es la numero 2\n");
     } /* fin del if TOTAL_1>TOTAL_2 */
  } /* fin del if TOTAL_1==TOTAL_2 */
  return;
} /* fin de la funcion decision */
```

Capítulo #7
Arreglos de Memoria

Un arreglo de memoria es una colección de variables del mismo tipo que se referencian utilizando un nombre común, así mismo, le permite a la computadora recordar valores pasados.

7.1 ARREGLOS DE MEMORIA UNIDIMENSIONALES

Con fines académicos, un arreglo de memoria unidimensional puede compararse con una columna de una hoja electrónica, por ejemplo, Excel. Sin embargo, en una columna de excel podemos insertar datos de diferente tipo: numéricos, alfabéticos o alfanuméricos.

Por el contrario, a un arreglo de memoria se le establece un tipo da datos y sólo aceptará eso. Es decir, si un arreglo de memoria es del tipo entero (**int**), solo aceptará números enteros con el rango correspondiente.

El formato general para la declaración de un arreglo unidimensional es

tipo nombre_variable [tamaño]

Aquí *tipo* declara el tipo de variable del arreglo. El *nombre_variable* es el nombre de la variable que es arreglo de memoria. El *tamaño* es la cantidad de elementos que guardará el arreglo. Por ejemplo, el siguiente arreglo llamado *lista* podrá almacenar 15 elementos enteros.

int Lista[15]; ⟶ esto puede compararse con una **columna** de excel.

No debe confundirse con la forma de declarar un variable de cadena de caracteres. Por ejemplo:

char nombre[30]; ⟶ esto puede compararse con una **celda** de excel**.**
char nombre[15][30]; ⟶ esto puede compararse con una **columna** de excel

En el primer caso se declaró una variable que es una cadena de caracteres y que puede almacenar hasta 30 caracteres.

En el segundo caso se declaró un arreglo de cadena de caracteres; este arreglo de memoria podrá almacenar hasta quince cadenas con 30 caracteres cada una.

Problema 7.1: Realice un programa que lea e imprima los elementos de una lista de "N" números. **Prog7-1.c**.

```
/* prog7-1.c */
#include<stdio.h>
/* declaracion de funciones */
void Leer_Elementos();
void Imprimir_Elementos();

/* declaracion de variables globales */
int I,N,Lista[100];
```

```
main()
{
   /* llamar a la funciones */
   Leer_Elementos();
   Imprimir_Elementos();
} /* fin de la funcion main() */

/* ------------------------------------ */
/* definicion de las funciones */
void Leer_Elementos()
{
   clrscr();
   printf("¿Cuantos elementos tiene la lista de numeros? \n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
       clrscr();
       printf("Dame el Elemento[%d] \n",I);
       scanf("%d",&Lista[I]);
   } /* fin del for */
   return;
} /* fin de la funcion Leer_Elementos() */

/* ------------------------------------ */
void Imprimir_Elementos()
{
  clrscr();
  for(I=1;I<=N;I++)
  {
     printf("Elemento[%d]=%d \n",I,Lista[I]);
  } /* fin del for */
  printf("\n \n Pulse cualquier tecla para continuar \n");
  getche();
  return;
} /* fin de la funcion Imprimir_Elementos() */
```

Problema7.2: Realice un programa que sume los elementos de una lista de “N” números. **Prog7-2.c**.

```
/* prog7-2.c */
#include<stdio.h>
/* declaracion de funciones */
void Leer_Elementos();
void Sumar_Elementos();
void Imprimir_Elementos();

/* declaracion de variables globales */
int I,N,Lista[100],Sumatoria;
main()
{
   /* llamar a las funciones */
   Leer_Elementos();
   Sumar_Elementos();
   Imprimir_Elementos();
} /* fin de la funcion main() */

/* ------------------------------------ */
/* defincion de las funciones */
void Leer_Elementos()
{
   clrscr();
   printf("¨Cuantos elementos tiene la lista de numeros? \n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
       clrscr();
       printf("Dame el Elemento[%d] \n",I);
       scanf("%d",&Lista[I]);
   } /* fin del for */
   return;
} /* fin de la funcion Leer_Elementos() */
```

```
/* ------------------------------------ */
void Sumar_Elementos()
{
  Sumatoria=0;
  for(I=1;I<=N;I++)
   {
       Sumatoria=Sumatoria+Lista[I];
   } /* fin del for */
 return;
} /* fin de la funcion Sumar_Elementos() */

/* ------------------------------------ */
void Imprimir_Elementos()
{
  clrscr();
  for(I=1;I<=N;I++)
  {
     printf("Elemento[%d]=%d \n",I,Lista[I]);
  } /* fin del for */
  printf("La sumatoria de los elementos es %d \n",Sumatoria);
  printf("\n \n Pulse cualquier tecla para continuar \n");
  getche();
  return;
} /* fin de la funcion Imprimir_Elementos() */
```

Problema 7.3: Realice un programa que determine el elemento mayor de una lista de “N” números. **Prog7-3.c**.

```
/* prog7-3.c */
#include<stdio.h>
/* declaracion de funciones */
void Leer_Elementos();
void Elemento_Mayor();
void Imprimir_Elementos();

/* declaracion de variables globales */
int I,N,Mayor,Lista[100];
main()
{
   /* llamar a la funciones */
   Leer_Elementos();
   Elemento_Mayor();
   Imprimir_Elementos();
} /* fin de la funcion main() */

/* ------------------------------------ */
/* definicion de funciones */
void Leer_Elementos()
{
   clrscr();
   printf("¿Cuantos elementos tiene la lista de numeros? \n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
       clrscr();
       printf("Dame el Elemento[%d] \n",I);
       scanf("%d",&Lista[I]);
   } /* fin del for */
   return;
} /* fin de la funcion Leer_Elementos() */
```

```
/* ------------------------------------ */
void Elemento_Mayor()
{
  Mayor=Lista[1];
  for(I=2;I<=N;I++)
  {
      if(Lista[I]>Mayor)
      {
         Mayor=Lista[I];
      } /* fin if(Lista[I]>Mayor) */
  } /* fin del for */
  return;
} /* fin de la funcion Elemento_Mayor() */

/* ------------------------------------ */
void Imprimir_Elementos()
{
  clrscr();
  for(I=1;I<=N;I++)
  {
     printf("Elemento[%d]=%d \n",I,Lista[I]);
  } /* fin del for */
  printf("El elemento mayor de esta lista es; %d \n",Mayor);
  printf("\n \n Pulse cualquier tecla para continuar \n");
  getche();
  return;
} /* fin del funcion Imprimir_Elementos() */
```

Problema 7.4: Realice un programa que haga la búsqueda secuencial de un número en una lista de "N" números e imprima si fue encontrado o no. Suponga que ningún elemento se repite en la lista. **Prog7-4.c**. En este programa se puede observar que la variable **Encontrado** se utiliza como una variable de control que implica que si **Encontrado=0** no se encontró el número buscado en la lista, pero si **Encontrado=1** implica que el número buscado si está en la lista.

Por otro lado, en la función **Busqueda()** se utilizó la sentencia **continue** que implica que termine el **for()** sin terminar su ejecución.

```
/* prog7-4.c */
#include<stdio.h>
/* declaracion de funciones */
void Leer_Elementos();
void Busqueda();
void Imprimir_Elementos();

/* declaracion de variables globales */
int I,N,Buscado,Encontrado,Lista[100];
char OPCION[2];
main()
{
   /* llamar a las funciones */
   Leer_Elementos();
   do{
       clrscr();
       Busqueda();
       printf("\n \n ¨Deseas hacer otra busqueda en la misma lista de numeros? \n");
       printf("Pulse la letra 'S' para continuar, o cualquier otra letra para terminar\n");
       scanf(" %[^\n]",OPCION);
   }while(!strcmp(OPCION,"S") || !strcmp(OPCION,"s"));
} /* fin de la funcion main() */
```

```
/* ------------------------------------ */
/* definicion de las funciones */
void Leer_Elementos()
{
   clrscr();
   printf("¿Cuantos elementos tiene la lista de numeros? \n");
   scanf("%d",&N);
   for(I=1;I<=N;I++)
   {
       clrscr();
       printf("Dame el Elemento[%d] \n",I);
       scanf("%d",&Lista[I]);
   } /* fin del for */
   return;
} /* fin de la funcion Leer_Elementos() */


/* ------------------------------------ */
void Busqueda()
{
  Encontrado=0;
  printf("Dame el numero buscado: \n");
  scanf("%d",&Buscado);
  for(I=1;I<=N;I++)
  {
     if(Buscado==Lista[I])
     {
       Encontrado=1;
       continue;
     } /* fin del if(Buscado==Lista[I]) */
  } /* fin del for */
  if(Encontrado==1)
  {
     printf("\n \n El numero SI fue encontrado \n");
  }
  else
  {
     printf("\n \n El numero NO fue encontrado \n");
  } /* fin if(Encontrado) */
  Imprimir_Elementos(); /* llamada a la funcion Imprimir_Elementos() */
  printf("\n \n Pulse cualquier tecla para continuar \n");
  getche();
  return;
} /* fin de la funcion Busqueda */

/* ------------------------------------ */
void Imprimir_Elementos()
{
  for(I=1;I<=N;I++)
  {
     printf("Elemento[%d]=%d \n",I,Lista[I]);
  } /* fin del for */
  return;
} /* fin de la funcion Imprimir_Elementos() */
```

Problema 7.5. Utilice Funciones Definidas por el Usuario sin Parámetros y arreglos de memoria unidimensionales. Se deja caer una pelota desde una altura de 50 metros, rebota y cada vez su altura es de dos tercios de su altura en el rebote anterior.

a) Elabore un programa que determine e imprima la altura de la pelota desde el primer rebote hasta que la altura de la pelota sea igual ó menor a un centimetro.
b) Determine e imprima en cuál número del rebote la altura de la pelota ya es igual o inferior a cincuenta centímetros.
c) Calcule e imprima la altura promedio de la pelota en todos los rebotes.
d) Calcule e imprima cuántos rebotes le proporcionan a la pelota una altura superior al promedio.

```
/* prog7-5.c */
/* Pelota que rebota 2/3 partes de su altura anterior */

#include<stdio.h>
/* declaracion de funciones */
void inciso_a();
void inciso_b();
void inciso_c();
void inciso_d();

/* declaracion de variables globales */
float ALTURA[1000],ALTO,ALT_PROM;
int I,REBOTE,POSICION,ARRIBA;

main()
{
   clrscr();
   /* llamar a las funciones */
   inciso_a();
   while(ALTO>0)
   {
     inciso_b();
     inciso_c();
     inciso_d();
     break; /* romper el ciclo while */
   } /* fin del while */
}  /* fin de la funcion main() */

/* ------------------------------------ */

/* definicion de las funciones */
void inciso_a()
{
  clrscr();

  REBOTE=0;
  ALTO=1;
  printf("Dame la altura inicial de donde se deja caer la pelota, en metros\n");
  scanf("%f",&ALTO);
  if(ALTO>0)
  {
     do{
         REBOTE=REBOTE+1;
         ALTO=ALTO*2.0/3.0;
         ALTURA[REBOTE]=ALTO;
       }while(ALTO>0.01);

      /* imprimir todos los rebotes con sus alturas */
      printf("\n\nImprimir los Rebotes y sus Alturas\n");
      printf("Rebote   Altura\n");
      for(I=1;I<=REBOTE;I++)
      {
        printf("%d      %8.4f\n",I,ALTURA[I]);
      } /* fin del for */

      printf("Respuesta a la Pregunta del inciso A\n");
      printf("\nTotal de Rebotes %d hasta alcanzar altura=1 cm. o menor\n",REBOTE);
  }
  else
  {
     printf("La altura insertada es cero o menor a cero \n");
     printf("no tiene sentido hacer calculos con este numero \n");
     printf("si dese intentar de nuevo debera de ejecutar, otra vez, el programa\n");
  } /* fin del if */
  printf("\n\n pulse cualquier tecla para continuar");
  getche();
  return;
} /* fin de la funcion inciso_a */
```

```
/* ------------------------------------ */

void inciso_b()
{
   clrscr();
   for(I=1;I<=REBOTE;I++)
   {
     if(ALTURA[I]<=0.50)
     {
       POSICION=I;
       I=REBOTE;
     } /* fin del if */
   } /* fin del for */

   printf("Respuesta a la Pregunta del inciso B\n");
   printf("\nEl rebote en que la altura es 50 cms o menor es %d,con altura
%8.4f\n",POSICION,ALTURA[POSICION]);
   printf("\n\n pulse cualquier tecla para continuar");
   getche();
   return;
} /* fin de la funcion inciso_b */


/* ------------------------------------ */
void inciso_c()
{
   clrscr();
   ALT_PROM=0;
   for(I=1;I<=REBOTE;I++)
   {
      ALT_PROM=ALT_PROM+ALTURA[I];
   } /* fin del for */
   ALT_PROM=ALT_PROM/REBOTE;
   printf("Respuesta a la Pregunta del inciso C\n");
   printf("La altura Promedio es %8.4f\n",ALT_PROM);
   printf("\n\n pulse cualquier tecla para continuar");
   getche();
   return;
} /* fin de la funcion inciso_c */


/* ------------------------------------ */

void inciso_d()
{
   clrscr();
   ARRIBA=0;
   for(I=1;I<=REBOTE;I++)
   {
      if(ALTURA[I]>ALT_PROM)
      {
        ARRIBA=ARRIBA+1;
      } /* fin del if */
   } /* fin del for */
   printf("Respuesta a la Pregunta del inciso D\n");
   printf("La altura Promedio es %8.4f\n",ALT_PROM);
   printf("Total de Rebotes %d\n",REBOTE);
   printf("Numero de rebotes superior al promedio: %d\n",ARRIBA);
   printf("\n\n pulse cualquier tecla para regresar a la pantalla de edicion");
   getche();
   return;
} /* fin de la funcion inciso_d */
```

7.2 PROBLEMAS PROPUESTOS: Arreglos de Memoria Unidimensionales

Problema 7.6. Realice un programa que imprima la tabla de multiplicar de cualquier número.

Problema 7.7. Realice un programa que haga la búsqueda secuencial de un número en una lista de "N" números e imprima si fue encontrado o no. Suponga que los elementos de la lista se pueden repetir, si se repite imprima el número de veces que está repetido.

Problema7.8. Realice un programa que sume los elementos de dos listas de números y guarde los valores en una tercera lista.

Problema 7.9. Utilice Funciones Definidas por el Usuario sin Parámetros. En dos arreglos unidimensionales se tiene la siguiente información de "N" personas. En el primero, se tiene la altura en metros. En el segundo, se tiene el peso en kilogramos.

En un tercer arreglo unidimensional deberá calcular el Índice de Masa Corporal (IMC) según la siguiente fórmula. **IMC=Peso/ Altura^2**

En un cuarto arreglo unidimensional deberá almacenar el resultado de acuerdo al IMC según la siguiente tabla:

IMC	**Resultado**
Menos de 19	Bajo en Peso
19 a 25	Peso Normal
25 a 30	Sobre Peso
Mayor de 30	Obesidad

a) Elabore un programa que determine e imprima el Índice de Masa Corporal (IMC) promedio.
b) Determine e imprima cuántas y cuáles personas están por encima o igual al promedio, y cuántas y cuáles por debajo del mismo.
c) Determine e imprima en cuántas personas pertenecen a cada categoría y el porcentaje que representan del total.

7.3 ARREGLOS DE MEMORIA BIDIMENSIONALES.

Los arreglos de memoria bidimensionales pueden compararse con una hoja electrónica por ejemplo Excel, la cual tiene filas y columnas, a esto se le llama arreglo matricial o tabla. De esta forma, la característica es que todos los elementos de la matriz son del mismo tipo de datos, cosa que no sucede en excel.

Observe la siguiente matriz en la cual todos sus elementos son numéricos:

-25	30	89	-45
12	90	-25	50
-70	-36	79	-81
63	42	78	-27

La matriz de posiciones sería la siguiente, considere (Filas, Columnas)

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

En el ejemplo anterior tenemos una matriz con dimensión de 4 por 4, pero como ya sabemos pueden ser de N por N.

Problema 7.10: Para ver un ejemplo de la forma como se manejan los arreglos bidimensionales haremos un programa que sume los elementos de la diagonal principal de la matriz (**prog7-11.c**). Para el caso de la matriz presentada como ejemplo, los elementos que forman parte de la diagonal principal son: -25 + 90 + 79 - 27 = 117, las posiciones que ocupan dichos elementos son (1,1) (2,2) (3,3) y (4,4). Es importante notar que en este programa estamos usando **for()** anidados.

```
/* prog7-10.c */
#include <stdio.h>
/* suma la diagonal principal de una matriz */
/* declaracion de funciones */
void carga_matriz();
void suma_diagonal();
void imprime_resultado();

/* declaracion de variables globales */
int Matriz[10][10], I, J, N, SUMA, fila, columna;

main()
{
   clrscr();
   /* llamada a las funciones */
   carga_matriz();
   suma_diagonal();
   imprime_resultado();
} /* fin de la funcion main() */


/* ---------------------------------------- */
/* definicion de funciones */

void carga_matriz()
{
  clrscr();
  printf("Dame la dimension de la matriz \n");
  scanf("%d",&N);
  for(I=1;I<=N;I++)
  {
     for(J=1;J<=N;J++)
     {
       printf("Dame el elemento[%d,%d] \n",I,J);
       scanf("%d",&Matriz[I][J]);
     } /* fin del for(J=1;J<=N;J++) */
  } /* fin del for(I=1;I<=N;I++) */
  Return;
} /* fin de la funcion carga_matriz */

/* ---------------------------------------- */
void suma_diagonal()
{
   SUMA=0;
   for(I=1;I<=N;I++)
  {
     for(J=1;J<=N;J++)
     {
       if(I==J)
       {
           SUMA=SUMA+Matriz[I][J];
       } /* fin del if(I==J) */
     } /* fin del for(J=1;J<=N;J++) */
  } /* fin del for(I=1;I<=N;I++) */
  Return;
} /* fin de la funcion suma_matriz */
```

```
/* ---------------------------------------- */
void imprime_resultado()
{
   clrscr();
   fila=5;
   for(I=1;I<=N;I++)
   {
     columna=5;
     for(J=1;J<=N;J++)
     {
       gotoxy(columna,fila);printf("%d",Matriz[I][J]);
       columna=columna+10;
     } /* fin del for(J=1;J<=N;J++) */
     printf("\n");
     fila=fila+1;
  } /* fin del for(I=1;I<=N;I++) */
  printf("\n \n La suma de la diagonal principal es %d\n",SUMA);
  printf("Pulse cualquier tecla para continuar \n");
  getche();
  return;
} /* fin de imprime_resultado */
```

Problema 7.11. Realizar el siguiente programa con funciones sin parámetros. En un arreglo bidimensional de 10x4 se tiene almacenadas las calificaciones de treinta alumnos en seis exámenes diferentes. Realice un programa que obtenga lo siguiente.

a) El promedio de calificaciones de los diez alumnos en los cuatro exámenes.
b) El alumno o alumnos que obtuvieron la mayor calificación en el tercer examen, en cualquier caso deberá imprimir cuantos alumnos fueron.
c) El examen en el que el promedio del grupo fue el más alto.
d) Cuántos alumnos están aprobados y cuántos reprobados así como el porcentaje que representa.

	CAL1	CAL2	CAL3	CAL4	PROMEDIO
1	89	90	96	95	92.50
2	67	69	79	100	78.75
3	78	96	93	94	90.25
4	59	79	40	49	56.75
5	97	98	60	97	88.00
6	48	95	70	95	77.00
7	78	100	78	85	85.25
8	94	78	68	60	75.00
9	47	95	97	70	77.25
10	69	86	69	80	76.00
PROM_EX	72.60	88.60	75.00	82.50	

```
/* prog7-11.c */
#include <stdio.h>
/* declaracion de funciones */
void leer_datos();
void inciso_a();
void inciso_b();
void inciso_c();
void inciso_d();

/* declaracion de variables globales */
int I, J, FILA, COLUMNA, EXAMENES, ALUMNOS,MAYORES[15];
int CONTAR,POSICION;
float CAL[15][15], PROMEDIO[15],MAYOR,PROM_EX[15],SUMA;
float APROBADOS,REPROBADOS,PORC_APROBADOS, PORC_REPROBADOS;

main()
{
   clrscr();
   /* llamada a las funciones */
   leer_datos();
   inciso_a();
   inciso_b();
   inciso_c();
   inciso_d();
} /* fin de la funcion main() */


/* ---------------------------------------- */
/* definicion de funciones */

void leer_datos()
{
  clrscr();
  printf("¨Cuantos Alumnos son? \n");
  scanf("%d",&ALUMNOS);
  printf("¨Cuantos Examenes son? \n");
  scanf("%d",&EXAMENES);
  for(I=1;I<=ALUMNOS;I++)
  {
     clrscr();
     printf("Dame las Calificaciones del Alumno # %d\n",I);
     for(J=1;J<=EXAMENES;J++)
     {
        do{
          printf("Dame la califacion del Examen %d\n",J);
           scanf("%f",&CAL[I][J]);
          if(CAL[I][J]<0 || CAL[I][J]>100)
          {
             printf("Calificacion Invalida,");
             printf("favor de insertar otro numero\n");
          } /* fin del if */
         }while(CAL[I][J]<0 || CAL[I][J]>100);
     } /* fin del for(J=1;J<=EXAMENES;J++) */
  } /* fin del for(I=1;I<=ALUMNOS;I++) */
  printf("\n\nPulse cualquier tecla para continuar\n");
  getche();
  return;
} /* fin de la funcion leer_datos */
```

```
/* ---------------------------------------- */
void inciso_a()
{
   clrscr();
   SUMA=0;
   for(I=1;I<=ALUMNOS;I++)
   {
     for(J=1;J<=EXAMENES;J++)
     {
        SUMA=SUMA+CAL[I][J];
     } /* fin del for(J=1;J<=EXAMENES;J++) */
     PROMEDIO[I]=SUMA/EXAMENES;
     SUMA=0;
   } /* fin del for(I=1;I<=ALUMNOS;I++) */

   FILA=4;
   COLUMNA=8;
   printf("Promedios por Alumno\n");
   for(J=1;J<=EXAMENES;J++)
   {
      gotoxy(COLUMNA,FILA); printf("EX%d",J);
      COLUMNA=COLUMNA+10;
   } /* fin del for(J=1;J<=EXAMENES;J++)*/
   gotoxy(COLUMNA,FILA);printf("Promedio");

   FILA=5;
   for(I=1;I<=ALUMNOS;I++)
   {
     COLUMNA=5;
     for(J=1;J<=EXAMENES;J++)
     {
        gotoxy(COLUMNA,FILA);printf("%8.2f",CAL[I][J]);
        COLUMNA=COLUMNA+10;
     } /* fin del for(J=1;J<=EXAMENES;J++) */
     gotoxy(COLUMNA,FILA);printf("Alumno #%d=%8.2f\n",I,PROMEDIO[I]);
     FILA=FILA+1;
  } /* fin del for(I=1;I<=ALUMNOS;I++) */
  printf("\n\nPulse cualquier tecla para continuar\n");
  getche();
  return;
} /* fin de la funcion inciso_a */

/* ---------------------------------------- */
void inciso_b()
{
   /* tercer examen */
   J=3;
   MAYOR=CAL[1][J];
   CONTAR=0;
   for(I=2;I<=ALUMNOS;I++)
   {
       if(CAL[I][J]>MAYOR)
       {
          MAYOR=CAL[I][J];
       } /* fin del if() */
   } /* fin del for(I=1;I<=ALUMNOS;I++) */

   for(I=1;I<=ALUMNOS;I++)
   {
       if(CAL[I][J]==MAYOR)
       {
          CONTAR=CONTAR+1;
          MAYORES[CONTAR]=I;
       } /* fin del if() */
   } /* fin del for(I=1;I<=ALUMNOS;I++) */

   printf("Alumnos con la calificacion mas Alta en el Tercer Examen: %d\n",CONTAR);
   for(I=1;I<=CONTAR;I++)
   {
     printf("Alumno Numero %d\n",MAYORES[I]);
   } /* fin del for(I=1;I<=CONTAR;I++) */
```

```
   printf("\n\nPulse cualquier tecla para continuar\n");
   getche();
   return;
} /* fin de la funcion inciso_b */

/* ---------------------------------------- */
void inciso_c()
{
   SUMA=0;
   for(I=1;I<=ALUMNOS;I++)
   {
     for(J=1;J<=EXAMENES;J++)
     {
        PROM_EX[J]=PROM_EX[J]+CAL[I][J];
     } /* fin del for(J=1;J<=EXAMENES;J++) */
   } /* fin del for(I=1;I<=ALUMNOS;I++) */

   for(J=1;J<=EXAMENES;J++)
   {
      PROM_EX[J]=PROM_EX[J]/ALUMNOS;
      printf("Promedio en el examen #%d = %8.2f\n",J,PROM_EX[J]);
   } /* fin del for for(J=1;J<=EXAMENES;J++) */

   MAYOR=PROM_EX[1];
   POSICION=1;
   for(I=2;I<=EXAMENES;I++)
   {
     if(PROM_EX[I]>MAYOR)
       {
          MAYOR=PROM_EX[I];
          POSICION=I;
       } /* fin del if() */
   } /* fin del for(I=2;I<=ALUMNOS;I++) */
   printf("El Examen del promedio mas alto es %d con %8.2f\n",POSICION,MAYOR);
   printf("\n\nPulse cualquier tecla para continuar\n");
   getche();
   return;
} /* fin de la funcion inciso_c */


/* ---------------------------------------- */

void inciso_d()
{
   clrscr();
   for(I=1;I<=ALUMNOS;I++)
   {
      if(PROMEDIO[I]>=70)
      {
         APROBADOS=APROBADOS+1;
      }
      else
      {  REPROBADOS=REPROBADOS+1;

      } /* fin del if() */
   } /* fin del for(I=1;I<=ALUMNOS;I++) */

   PORC_APROBADOS=APROBADOS/ALUMNOS*100;
   PORC_REPROBADOS=REPROBADOS/ALUMNOS*100;
   printf("Aprobados  = %8.2f  Porcentaje Aprobados=%8.2f\n",APROBADOS,PORC_APROBADOS);
   printf("Reprobados = %8.2f  Porcentaje Reprobados=%8.2f\n",REPROBADOS,PORC_REPROBADOS);
   printf("\n\nPulse cualquier tecla para volver a la pantalla de edicion \n");
   getche();
   return;
} /* fin de la funcion inciso_d */
```

7.4 PROBLEMAS PROPUESTOS: Arreglos de Memoria Bidimensionales

Problema 7.12: Realice un programa que sume los elementos de la diagonal secundaria de una matriz.
Problema 7.13: Realice un programa que sume los elementos centrales de una matriz.
Problema 7.14: Realice un programa que sume los elementos periféricos de una matriz.
Problema 7.15: Realice un programa que sume los elementos de la matriz triangular superior.
Problema 7.16: Realice un programa que sume los elementos de la matriz triangular inferior.
Problema 7.17: Realice un programa que sume dos matrices y guarde el resultado en una tercera matriz.
Problema 7.18: Realice un programa que determine cuántos elementos pares y cuántos impares hay en una matriz y cuántos de ellos son Cero.
Problema 7.19: Realice un programa que suma las filas y las columnas de una matriz. Se deberá imprimir la sumatoria por fila y por columnas.
Problema 7.20: Realice un programa que determine el elemento mayor de una matriz así como su posición
Problema 7.21: Realice un programa que multiplique matrices.
Problema 7.22: Realice un programa que resuelva un problema de ecuaciones simultaneas a través del método: Gauss-Jordan y Montante.
Problema 7.23. En este programa utilice variables enumeradas. En un arreglo bidimensional de 12x3 se tiene los costos de producción de tres departamentos: dulces, bebidas y conservas. Los costos de producción corresponden a cada mes de año anterior.

Escriba un programa que pueda proporcionar la siguiente información.

e) ¿En qué mes se registro el mayor costo de producción de dulces?.
f) ¿Promedio anual de los costos de producción de bebidas.
g) ¿En qué mes se registró el mayor costo de producción en bebidas y en que mes el de menor costo.
h) ¿Cuál fue el rubro que tuvo el menor costo de producción en diciembre.