MODULO : ALGORITMICA PARA PROGRAMACION

POR : YAMIL ARMANDO CERQUERA ROJAS
DOCENTE UNIVERSIDAD SURCOLOMBIANA
Neiva – Huila - Colombia
2001

Nota: Este módulo se encuentra a disposición de todos aquellos que estén interesados en aprender a resolver problemas básicos haciendo uso de las herramientas que nos da los diferentes lenguajes de programación para ello. Conocidas como ***Estructuras de programación*** (Asignación, Decisión, Cíclicas y de Selección Múltiple).

Cualquier sugerencia o critica la podrán dirigir a **yacerque@usco.edu.co.**

Agradecimientos especiales al personal de monografías.com por el LINK.

CONTENIDO

INTRODUCCIÓN

EL lenguaje C es el resultado de un proceso de desarrollo que inició con un lenguaje denominado BCPL. Este influenció a otro llamado B (inventado por Ken Thompson). En los años 70; éste lenguaje llevó a la aparición del C.

Con la popularidad de las microcomputadoras muchas compañías comenzaron a implementar su propio C por lo cual surgieron discrepancias entre sí.

Por esta razón ANSI (American National Standars Institute, por sus siglas en inglés), estableció un comité en 1983 para crear una definición no ambigua del lenguaje C e independiente de la máquina que pudiera utilizarse en todos los tipos de C.

Algunos de las C existentes son:

- Quick C
- C++
- Turbo C
- Turbo C ++
- Borland C
- Microsoft C
- Visual C
- C Builder

Los códigos presentados en este libro están basados en el C y Turbo pascal estándar los cuales pueden utilizarse en todos los tipos de C y Pascal existentes en el mercado.

C es un lenguaje de programación de nivel medio ya que combina los elementos del lenguaje de alto nivel con la funcionalidad del ensamblador.

Su característica principal es ser portable, es decir, es posible adaptar los programas escritos para un tipo de computadora en otra.

Otra de sus características principales es el ser estructurado, es decir, el programa se divide en módulos (funciones) independientes entre sí.

El lenguaje C inicialmente fue creado para la programación de:

- Sistemas operativos

- Intérpretes
- Editores
- Ensambladores
- Compiladores
- Administradores de bases de datos.

Actualmente, debido a sus características, puede ser utilizado para todo tipo de programas.

Como principal objetivo, este libro trata de vulgarizar el lenguaje C colocándolo al alcance de todos los estudiosos de cualquier nivel o profesionales de la programación con conocimientos mínimos informáticos.

Para ello, a lo largo de los diferentes capítulos, se trata al lenguaje no como algo aislado en si mismo, sino relacionado específicamente con la Metodología de la Programación y la Estructura de Datos, fuentes indispensables del conocimiento profundo de cualquier lenguaje de programación.

El término vulgarizar, que aquí se utiliza, no significa en modo alguno eliminar la teoría del lenguaje y descender al terreno práctico olvidando la misma, si no completamente teoría y practica mediante la inclusión de todos los ejemplos oportunos y las aclaraciones necesarias aunque estas puedan parecer, a veces, superfluas e incluso dadas por sabidas.

OBJETIVOS Y CONTENIDO

Este libro tiene un objetivo fundamental: Introducir y explicar los conceptos fundamentales de las estructuras de programación utilizadas en los diferentes de lenguajes de programación de microcomputadoras. Esta diseñado para ser utilizado de diversas formas: Como libro de texto, libro complementario, libro de problemas de programación, libro de análisis numérico o bien como libro de introducción a la programación de computadoras. El seguimiento de este libro no requiere ningún conocimiento previo de programación de computadoras, aunque, si se utiliza como libro de problemas puede complementarse con algún libro de mayor contenido teórico relativo a algoritmos, estructuras de datos o bien lenguajes de programación.

El libro esta dividido en varios capítulos que abarcan los conceptos esenciales de las estructuras de programación en los diferentes lenguajes comúnmente utilizados en la programación de computadoras.

En la primera parte se trata lo relacionado con los símbolos utilizados en la diagramación o solución de problemas a nivel simbólico (Diagramas de flujo).

En la segunda parte se desarrollarán algunos ejercicios donde se muestra de manera práctica la utilización de dichos símbolos

Una tercera parte donde se analizan las estructuras de asignación

Una cuarta parte donde se analizan las estructuras de decisión

Una quinta parte donde se analizan las estructuras repetitivas o cíclicas

Una sexta parte donde se analizan las estructuras de selección múltiple.

Cada uno de los capítulos esta diseñado de tal forma que el lector entienda de manera clara y sencilla el uso de cada uno de las estructuras mencionadas anteriormente. De igual forma cada uno de los capítulos estará soportado con una buena cantidad de ejercicios resueltos que aclaran la forma correcta de utilización de dichas estructuras. A cada uno de los ejercicios que se plantean en el presente libro, se le hará un análisis de su solución de manera mecánica, se llevará a un análisis simbólico y luego se desarrollará el algoritmo al nivel de diagramas de flujo, Seudo lenguajes, Seudo códigos y Códigos.

Para esta última parte se codificará cada ejercicio en Turbo Pascal y Lenguaje C.

PARA QUE SIRVE LA PROGRAMACIÓN

Una persona piensa y se comporta obedeciendo a un secuencial lógico. Un computador realiza tareas y maneja datos en memoria obedeciendo a una secuencia de pasos lógicos para lo cual ha sido programado.

Programación de computadoras es la ciencia que permite a una persona programar una computadora para que resuelva tareas de manera rápida. Un Programa de computadora se puede definir como una secuencia de instrucciones que indica las acciones o tareas que han de ejecutarse para dar solución a un problema determinado.

Programar computadoras es indispensable en cualquier área de la ingeniería, ya que diferentes problemas que se puedan presentar tardan tiempo resolverlos de manera manual. La computadora resuelve problemas de acuerdo como se le haya programado de manera rápida.

QUE ES LA PROGRAMACIÓN

La definición anterior deja muchas cosas que decir. Para llegar a tener una secuencia de instrucciones que den solución a un problema es necesario ejecutar varias etapas.

Etapa de análisis: En esta etapa el programador debe entender claramente el problema. Saber que es lo que se quiere resolver. (analizar)

Etapa de Solución general: Escribir la serie de pasos que sean necesarios para dar solución al problema. Estos pasos se pueden desarrollar a través de un Diagrama de flujo (Utilizando símbolos) ó a través de un seudo lenguaje (Utilizando Lenguaje común). A lo anterior es lo que se conoce con el nombre de Algoritmo.

Etapa de prueba: Consiste en chequear el algoritmo paso a paso para estar seguro si la solución da solución verdaderamente el problema. (Prueba de escritorio).

Etapa de implementación específica: Consiste en traducir el algoritmo a un lenguaje de programación. (Codificar).

Etapa de prueba: Consiste en ejecutar el programa en un computador y revisar los datos arrojados para ver si son correctos y hacer los ajustes necesarios. (Implementar).

Etapa de uso: Consiste en instalar el programa de manera definitiva para el uso por parte del usuario.

QUE ES UN LENGUAJE DE PROGRAMACIÓN

Se puede definir un lenguaje de programación como un conjunto de reglas ó normas, símbolos y palabras especiales utilizadas para construir un programa y con él, darle solución a un problema determinado.

El lenguaje de programación es el encargado de que la computadora realice paso a paso las tareas que el programador a diseñado en el algoritmo.

Se puede decir que un lenguaje de programación es el intermediario entre la máquina y el usuario para que este último pueda resolver problemas a través de la computadora haciendo uso de palabras (funciones) que le traducen dicho programa a la computadora para la realización de dicho trabajo.

NIVELES DE LOS LENGUAJES

Desde que se desarrollaron las máquinas programables se han desarrollado lenguajes con los cuales las personas puedan dar órdenes a éstas. En su orden los lenguajes de programación se pueden clasificar así:

Lenguaje de máquina: Las primeras computadoras se programaban en código de máquina. Se puede decir que los programas eran diseñados en código binario. Eran difíciles de leer, difíciles de entender y por su puesto difíciles de corregir. Los programas se caracterizaban por ser pequeños.

Lenguajes de Bajo Nivel: Para dar solución a lo difícil que era programar en código máquina, se desarrolló un lenguaje conocido como lenguaje ensamblador. Este lenguaje era encargado de tomar algunas palabras comunes a una persona y traducirlas al código máquina. Lo anterior facilitaría un poco la escritura de programas.

Lenguajes de alto nivel: Como las personas resuelven problemas y se comunican en lenguajes naturales (español, ingles, francés, etc.), se desarrollaron lenguajes de programación que estuvieran mas cerca de ésta manera de resolver problemas. De los lenguajes de alto nivel se puede citar el Basic, Cobol, Fortran, Pascal, Turbo Pascal, C, Modula, Ada. Como se hace necesario traducir el programa a lenguaje de máquina, en los lenguajes de alto nivel esa operación la realiza algo que se conoce con el nombre de Compilador.

TIPOS DE PROGRAMACIÓN

Dependiendo del lenguaje de programación que se elija, se puede hablar del tipo de programación que se va a realizar.

Secuencial : Se considera programación secuencial a los programas que se diseñan con instrucciones que van unas detrás de otras. Las líneas se ejecutan una a una en secuencia. Ejemplos tales como Basic, Cobol.

Estructurada: Se considera programación estructurada a la programación que se hace por módulos. Cada módulo realiza alguna tarea específica y cuando se necesite esa tarea simplemente se hace el llamado a ese módulo independiente de que se tengan que ejecutar los demás. Ejemplos tales como: Turbo PASCAL, C, Modula, Ada.

Orientada a Objetos: Se considera programación orientada a objetos aquellos lenguajes que permiten la utilización de objetos dentro del diseño del programa y el usuario puede pegar a cada objeto código de programa. Ejemplos de estos lenguajes se pueden mencionar el Visual Basic de la Microsoft, C Builder de la Borland Internacional, Java, Xml entre otros.

Lógica o de lenguaje natural: son aquellos programas que se diseñan con interfaces tal que la persona o usuario puede ordenar a la máquina tareas en un lenguaje natural. Pueden interactuar como una persona pero nunca llegan a producir conocimiento. Ejemplo como Prolog (Programming Logic). Estos lenguajes se desarrollaron con base en las estructuras de sus antecesores. Recorren o navegan las bases de datos obedeciendo a reglas.

Inteligencia Artificial: Los programas de inteligencia artificial Son programas que se acercan a la inteligencia humana. Estos programas son capaces de desarrollar conocimiento. Este tipo de lenguajes trabajan similar a la mente humana.

EL SECRETO DE PROGRAMAR ESTÁ EN LA ESTRUCTURACIÓN

Turbo Pascal y C por su diseño son lenguajes estructurados. C y Turbo PASCAL no permiten al programador enlazar sentencias de cualquier manera. Existe una estructura básica que cada programa debe seguir (Estructura de un programa) y el compilador es estricto a la hora de hacer cumplir estas reglas. Un programa ha de ser codificado en varias partes y cada una de ellas debe ir en el lugar que le corresponde.

La idea fundamental del lenguaje C y Turbo Pascal es crear programas que sean comprendidos sin necesidad de emplear docenas de páginas de diagramas de flujo y miles de explicaciones. Esta manera de realizar los programas es a lo que denomina como "PROGRAMACIÓN ESTRUCTURADA".

Aunque se pueda llegar a realizar programas que aparentan tener una estructura (ser estructurados), Turbo Pascal y C son lenguajes que exige su utilización.

La programación estructurada le permite realizar pequeñas rutinas específicas para cada tarea que se quiera realizar, y a cada una de esas rutinas se les da un nombre (Identificador) para cuando el programador la requiera sólo la llame con su nombre y automáticamente se ejecutará.

ELEMENTOS BÁSICOS DE UN PROGRAMA

En el ámbito general, un programa codificado o escrito bajo cualquier lenguaje de programación estructurado consta básicamente de dos secciones:

- Sección encabezado
- Sección cuerpo de programa

La sección de encabezado es usada para declarar, mencionar o identificar las variables con sus respectivos tipos y/o las constantes que se vayan a utilizar en el desarrollo del programa, así como también el nombre de las funciones y/o los procedimientos que ejecutarán las instrucciones de los diferentes algoritmos que va a tener dicho programa. Además en esta sección se declaran los archivos de inclusión (Archivos con extensión ".h") que permiten el uso de algunas funciones que son necesarias para el desarrollo en si del programa. Igualmente se especifican las estructuras de datos complejas que se vayan a manejar.

En la sección cuerpo de programa realmente se describen todos los procedimientos y/o funciones que se van a ejecutar dentro del programa así como también el código del programa principal. Como cuerpo de programa es indispensable que haya parte principal mientras que los procedimientos y/o funciones son opcionales.

ELEMENTOS BÁSICOS DE UN PROGRAMA EN C

El siguiente ejemplo ilustra algunos de los componentes que con mayor frecuencia se usan cuando se trata de codificar programas utilizando el lenguaje de programación C. Cada una de las líneas es numerada para efectos de explicación posterior. En el momento de probarlo en el lenguaje de programación no es necesario la numeración.

```
1. /* Programa para calcular el producto de dos números previamente ingresados por teclado */
2. #include “stdio.h”
3. int a,b,c;
4. int producto(int x, int y);
5. main()
6. {
7. /* pide el primer numero */
8. printf(“digite un numero entre 1 y 100“);
9. scanf(“%d”,&a);
10.
11. /* pide el segundo numero */
12. printf(“digite un numero entre 1 y 100”);
13. scanf(“%d”,&b);
14.
15. /* calcula y despliega el producto */
16. c = producto(a,b);
17. printf(“\n%d por %d = %d”, a, b, c);
18. }
19.
20. /* función que calcula y regresa el producto de sus dos argumentos */
21. int producto(int x, int y)
22. {
23.  return(x*y);
```

24. }

Aunque cada uno de los programas son distintos, todos tienen características comunes. Los elementos básicos de un programa codificado en lenguaje C son los siguientes tomando el ejemplo anteriormente descrito:

- La función main() (línea 5 y desarrollada de la 6 – 18)
- La directiva #include (línea 2)
- Definición de variables (línea 3)
- Prototipo de función (línea 4)
- Enunciados del programa (línea 8,9,12,13,16,17,23)
- Definición de función (línea 21 – 24)
- Comentarios (Línea 1,7,11,15,20)

Veamos en que consiste cada uno:

main()

En C, todo el código está basado en funciones. El programa principal no es la excepción. main() indica el comienzo de la función principal del programa la cual se delimita con llaves. . Este componente es obligatorio en cualquier programa que se compile con lenguaje C. En su forma simple la función main consiste en la palabra main seguida de paréntesis () y las instrucciones que ejecutan se encuentran demarcadas por las dos llaves {}. Sin embargo no todos los programas tienen esta tradicional función; los programas de Windows escritos en C y C++ disponen de una función de inicio llamada winmain() en lugar de la tradicional main():

Nota: La sección main o función main es la primera sección de código que se ejecuta cuando se le dé la orden de ejecutar el código. Desde aquí se hará el respectivo enlace con las demás funciones que posea el programa.

La directiva #Include:

La directiva #include da instrucciones al compilador C para que añada el contenido de un archivo de inclusión al programa durante la compilación. Un

archivo de inclusión es un archivo de disco separado que contiene información necesaria para el compilador. Varios de estos archivos (Algunas veces llamados archivos de encabezado) se proporcionan con el compilador. Nunca se necesita modificar la información de estos archivos y esta es la razón por la cual se mantienen separados del código fuente. Todos los archivos de inclusión deben tener la extensión (.h) (por ejemplo stdio.h).

Se usa la directiva #include para darle instrucciones al compilador que añada un archivo específico al programa durante la compilación. La directiva #include, en este programa de ejemplo, significa "añada el contenido del archivo stdio.h ".

Nota: La mayoría de los programas codificados en lenguaje C requieren uno o más archivos de inclusión. Depende del tipo de funciones que se estén utilizando: Ejemplo de ellas printf(), scanf(), clrscr(); gotoxy().....

Definición de variables:

Una variable es un nombre asignado a una posición de almacenamiento de datos. El programa utiliza variables para guardar varios tipos de datos durante la ejecución del programa. En C, una variable debe ser definida antes de que pueda ser usada. Una definición de variable le informa al compilador el nombre de la variable y el tipo de datos que va a guardar. En el programa de ejemplo la definición de la línea 3, "int a,b,c" define tres variables, llamadas a, b, y c que guardará cada una un valor de tipo entero.

Las variables se pueden declarar en la zona de encabezado de un programa o al inicio de una función o un procedimiento.

Variables globales:

Las variables globales son aquellas variables que se definen o declaran en la zona de encabezado de cualquier programa en C. Estas variables pueden ser utilizadas en cualquier parte del programa, igualmente puede ser modificado su valor desde cualquier instrucción.

Variables Locales:

Las variables son consideradas como locales cuando su declaración se hace al inicio de una función o un procedimiento. Las variables que hayan sido declaradas como locales solo podrán ser reconocidas por el procedimiento o función donde se haya declarado. En ninguna otra parte del programa se puede hacer uso de ellas.

Un reciente cambio de regla de aproximación del comité de ANSI de C++ afecta la visibilidad de una variable que se declara en una instrucción como for. El siguiente código generará un error de compilador.

```
Int index;
For (int i=1;I<10;I++)
{ cout <<I;
}
index=I;
```

Prototipo de función:

Un prototipo de función proporciona al compilador C el nombre y los argumentos de una función contenida en el programa, y debe aparecer antes de que la función sea usada.

Un prototipo de función es diferente de una definición de función que contiene las instrucciones actuales que hacen a la función. El prototipo de función debe llevar (;).

Nota: Un prototipo de función debe llevar (;) en la sección de encabezado. Cuando se desarrolla en el cuerpo del programa no debe llevar (;)

Enunciados del programa:

El trabajo real de un programa C es hecho por sus enunciados. Los enunciados de C despliegan información en la pantalla, leen entradas desde el teclado, ejecutan operaciones matemáticas, llaman funciones, leen archivos de disco y hacen todas las otras operaciones que un programa necesita ejecutar.

printf():

El enunciado printf() es una función de biblioteca que despliega información en la pantalla. El enunciado printf puede desplegar un simple mensaje de texto o un mensaje y el valor de una o más variables del programa.

Ej: printf(“este es un simple comentario”);

printf(“El cuadrado de %d es igual a %d”,a,a*a);

scanf():

El enunciado scanf() es otra función de biblioteca. Ella lee datos desde el teclado y asigna los datos a una o más variables del programa.

Ej: scanf("%d",&a);

scanf("%d %d",&a,&b);

En el primer ejemplo se lee un valor entero y este es asignado a la dirección de la variable a. En el caso del segundo ejemplo se leen dos datos enteros y cada valor es almacenado en las variable ***a*** y ***b*** respectivamente.

Definición de función:

Una función es una sección de código independiente y auto contenida que es escrita para ejecutar determinada tarea. Cada función tiene un nombre y el código de cada función es ejecutado incluyendo el nombre de la función, en una instrucción de programa. A esto se le llama llamado de la función.

La función denominada producto, es una función definida por el usuario. Tal como lo indica su nombre las funciones definidas por el usuario son escritas por el programador durante el desarrollo del programa. Esta función es simple, ya que todo lo que hace es multiplicar dos valores y regresar el resultado de dicha multiplicación al programa que la llamó.

El C también incluye funciones de biblioteca que son parte del paquete del compilador C. Las funciones de biblioteca ejecutan la mayoría de las tareas comunes (como la entrada /salida de la pantalla el teclado y disco) que necesita el programa. En el programa de ejemplo printf y scanf son funciones de biblioteca.

Comentarios:

Se identifican porque van entre diagonales y asterisco. Nos sirve para escribir información que nos referencie al programa pero que no forme parte de él. Por ejemplo especificar que hace el programa, quien lo elaboró, en que fecha, que versión es, etc. El compilador ignora lo que haya como comentario en el momento de hacer la compilación del programa.

Ej: /* Este es un comentario */

En el caso de otra interfaces como MatLab los comentarios simplemente se declaran iniciando la línea con un %

Flujo de sentencias:

Es la declaración de todas las instrucciones que conforman un programa. Todas las sentencias van separadas por (;), en renglones separados o de manera seguida.

Definición de funciones creadas por el programador utilizadas en main(): Finalmente, se procede a definir el contenido de las funciones utilizadas dentro de main(). Estas contienen los mismos elementos que la función principal.

Importante:

Después de cada asignación o función es imprescindible colocar un punto y coma (;) ya que éste es un terminador de proposiciones. En caso de que no se escriba, se marcará un error a la hora de compilar el programa.

En C, los comandos, palabras reservadas o funciones deben ser escritos con letras minúsculas, tal como se hizo en el programa de ejemplo. En el lenguaje Turbo Pascal no interesa de que modo se escriban. En el caso de las variables o las funciones definidas por el usuario la situación es similar para el caso del lenguaje C, no es lo mismo: Apellido - apellido ó APELLIDO

Aunque para nosotros es lo mismo, el compilador de C, los toma como tres nombres distintos. Por tanto, asegúrese de mandar llamar las variables o funciones exactamente de la misma forma en que las declaró. En el caso del los lenguajes Turbo Pascal y Basic no diferencian entre mayúsculas y minúsculas o sea que para dichos lenguajes lo mismo es "A" que "a".

Ejemplos:

a) En el siguiente ejemplo se despliega o muestra un mensaje de bienvenida en la posición 20 sobre el eje X (Horizontal) y 10 sobre el eje Y (Vertical) de la pantalla y se espera 2.5 segundos aproximadamente mientras el usuario observa dicho mensaje.

```
#include<stdio.h>
#include<conio.h>
main()
```

```
{       clrscr(); gotoxy(20,10);
        printf("BIENVENIDO AL CURSO DE C ESTÁNDAR"); delay(2500); }
```

La función delay en el ejemplo anterior permite indicarle a la maquina cuando ejecute dicha función que se espere 2.5 segundos aproximadamente mientras el usuario lee lo que esta presentado en la pantalla. Pasado los dos segundos el compilador continuara la ejecución de la siguiente línea.

b) El siguiente programa le pregunta por su nombre y los años que tienes. Al final da como respuesta el número de días vividos y le coloca un mensaje para que termine el programa. Por la forma que se detiene el programa (getch()) hasta que no se pulse una tecla la computadora no terminara la ejecución del programa.

```
#include<stdio.h>
#include<conio.h>
main()
{ char nombre[50];  int edad;
        clrscr();
        gotoxy(10,5);   printf("¿Cómo te llamas?\n ");
        scanf("%s",&nombre);
        gotoxy(10,6);   printf("¿Cuántos años tienes?\n");
        scanf("%i",&edad);
        edad = edad * 365;
        gotoxy(10,8);   printf("%s, has vivido %d días",nombre,edad);
        gotoxy(40,22); printf("Pulsa cualquier tecla para terminar...");
        getch();   }
```

Los parámetros %s y %d corresponden a los tipos de datos que se leen o escriben. En la sección de tipos de datos podrá encontrar los diferentes tipos de datos así como sus parámetros.

c) El siguiente ejemplo permite la captura por teclado de 3 números cada uno de ellos almacenados en un nombre de variable diferente y al final se entrega el promedio de dichos valores.

```
#include<stdio.h>
#include<conio.h>
main()
{ float numero;   float promedio=0;
        clrscr();
        gotoxy(10,5);    printf("Dame el primer número: ");
        scanf("%f",&numero);
        promedio+=numero;
        gotoxy(10,6);    printf("Dame el segundo número: ");
        scanf("%f",&numero);
        promedio+=numero;
        gotoxy(10,7);    printf("Dame el tercer número: ");
        scanf("%f",&numero);
        promedio += numero;
        promedio = promedio/3;
        gotoxy(10,8);    printf("El promedio es %f",promedio);
        gotoxy(40,22);  printf("Presione cualquier tecla para terminar...");
        getch();
}
```

Si el ejemplo anterior lo desea correr bajo MatLab tiene que guardarlo en un archivo tipo m en el directorio Work de Matlab o en el directorio donde haya configurado en Set Path de la Opción de menú File. Una vez guardado bajo cualquier nombre, entonces simplemente en el prompt del matlab lo ejecuta llamando el archivo por el nombre guardado.

```
promedio=0;
n=input('Dame el 1 número:');
promedio=promedio+n;
n=input('Dame el 2 número: ');
promedio=promedio+n;
n=input('Dame el 3 número: ');
promedio = promedio+ n;
```

```
promedio = promedio/3;
fprintf('El promedio es %f \n',promedio);
fprintf('Presione tecla para terminar...\n');  pause
```

d) El siguiente ejemplo convierte un número capturado por teclado en sistema numérico decimal al sistema octal utilizando parámetros que ofrece el lenguaje C para dicha conversión.

```
#include<stdio.h>
#include<conio.h>
main()
{  int numero;
        clrscr();
        gotoxy(10,5);    printf("Digite un número entero en decimal: ");
        scanf("%i", &numero);
        gotoxy(10,6);    printf("\n\n Su representación en octal es %o");
        gotoxy(40,22);   printf("Presione cualquier tecla para terminar...");
        getch();   }
```

e) El siguiente ejemplo escribe un mensaje de advertencia sobre algo al usuario y lo mezcla con algunos sonidos que son familiares.

```
#include <dos.h>
#include<conio.h>
int main(void)
{       clrscr();
        gotoxy(28,11);   printf("¡ P E L I G R O ! ");
        sound(250); delay(600);
        sound(80);   delay(600);
        delay(600);   nosound();
        return 0;       }
```

ESTRUCTURAS ESTÁTICAS DE DATOS

Se denominan estáticas las estructuras de datos simples, o complejas, que una vez definidas dentro de un programa, permanecen inalteradas durante la ejecución del mismo, sin poder variar, por tanto, su posición en memoria, ni su longitud en bytes, declarada al especificar el tipo de la misma.

En este sentido, y de menor a mayor complejidad, son estructuras estáticas tradicionales de datos:

- Las variables de carácter
- Las variables numéricas enteras
- Las variables numéricas de punto flotante o reales
- Las variables de tipo cadena (string)
- Las matrices (arrays), formadas por elementos contiguos en memoria de los tipos citados anteriormente
- Los registros y archivos

Tradicionalmente se han definido como dinámicas las estructuras de cola, pila y árbol por permitir la variación del número de sus elementos, dentro de ciertos límites, durante la ejecución del programa.

Por lo complejo que resulta el manejo de todo tipo de estructuras, en este libro se menciona tan solo los tipos de estructuras simples que son los mas manejados para la solución de problemas comunes a las Ingenierías.

TIPOS DE DATOS

En el presente capítulo se hará un recuento del tipo de datos usados en los lenguajes de programación con el fin de que el usuario pueda en un momento dado entender el tipo de datos utilizado en la mayor parte de ejercicios solucionados en el presente libro. Se visualizara para cada lenguaje en particular el tipo de dato que maneja al igual que sus rangos. Se dará cuenta que la mayoría maneja los datos con la misma filosofía y la variación de unos a otros esta dado solo por la sintaxis que se use para hacer referencia a ellos.

La gran variedad de datos que trae consigo un lenguaje de programación hace que el programador escoja el tipo mas adecuado para la aplicación que esté desarrollando. El rango de datos que maneja cada tipo, indica al usuario que escoger. Los datos se pueden agrupar en los siguientes tipos.

PREDEFINIDOS POR LOS LENGUAJES:

- ENTEROS
- DECIMALES
- DE CADENA
- BOLÉANOS

DEFINIDOS POR EL USUARIO

- SUBRANGO
- ENUMERADOS

PREDEFINIDOS POR LOS LENGUAJES

TIPOS DE DATOS ENTEROS

Los tipos enteros largos son más precisos, pero los enteros cortos son más simples, rápidos y ocupan menos memoria.

El valor extremo del rango de cada uno de los enteros está dado por el espacio que ocupa dicho entero en memoria, veamos.

Un byte de memoria esta formado por 8 bits. Cada uno de estos bit pueden ser un bit 0 o un bit 1. las combinaciones que podemos tener en un byte (8 bit) podrían ir desde:

0 0 0 0 0 0 0 0 a 1 1 1 1 1 1 1 1
(8 bits ceros) (8 bits unos)

Si se pasa el conjunto de bits (Sistema Binario) a Sistema decimal tenemos que el rango que se maneja ahora irá desde:

0	a	255
(decimal correspondiente a los 8 bits ceros)		(decimal correspondiente a los 8 bits unos)

Como el rango anterior está utilizando todos los ocho bits del byte para almacenamiento de datos, sólo toma datos enteros positivos, no hay forma de reservar ningún campo bit para el signo.

En el caso del entero corto, por utilizar siete bits para almacenamiento de datos, el octavo es usado para signar datos (signo mas o menos). Por tanto solo dispone en memoria de 7 bit para almacenar (bits unos) o (bits ceros). El rango manejado en sistema binario con siete (7) bit irá desde 0000000 hasta el 1111111, que en sistema decimal corresponde al rango de 0 al 127, o sea 128 datos enteros positivos. Como se deja un bit reservado para manejo de signo, entonces los datos negativos serán 128 también que irán en rango desde el -1 al -128.

El rango definitivo de enteros positivos y negativos para el tipo de dato entero corto será desde -128 al 127.

TIPOS DE DATOS REALES

Los números reales son aquellos datos que tienen parte decimal. Un real se caracteriza porque siempre tiene dígitos antes y después del punto. Cada uno de los lenguajes de programación tiene un grupo para este tipo de datos. Al igual que los enteros, los reales se diferencian unos de otros por el rango que cada uno de ellos maneja. Vea tabla de tipos de datos.

TIPOS DE DATOS CHAR

Se define como Char a cualquier caracter o símbolo que conforman la tabla denominada ASCII. Una variable que se defina como tipo Char puede almacenar un solo caracter especificado entre apóstrofes. {'A'}.

Cada caracter se almacena en memoria RAM como un byte o dicho de otra manera cada caracter se almacena en una posición de memoria. Por ser un solo byte, el número de caracteres diferentes que se pueden almacenar es de 256.

correspondiendo a los valores que van de 0 a 255 (en decimal) o 00000000 a 11111111 (en binario). El conjunto de caracteres de la tabla ASCII se agrupan en dos bloques así:

El primer bloque conformado por los caracteres de control. Corresponden a éste bloque los caracteres del código ASCII que van de 1 a 31.

El segundo bloque conformado por los caracteres básicos o caracteres imprimibles. Van del código ASCII 32 al 126, se introducen directamente por el teclado
El tercer bloque conformado por caracteres del código ASCII ampliado. Para hacer referencia a ellos en turbo PASCAL se preceden del signo #.

Para asignar cualquier caracter ASCII a una variable de tipo Char existen varias formas así:

```
A := 'g';      {Se asigna el caracter g a la variable A}
B := A;        {Se asigna el valor de A, a la variable B}
C := #65;   {Se asigna el caracter numero 65 de la tabla ASCII a la variable C}
```

TIPOS DE DATOS STRING

Los tipos de datos cadena {String} son secuencia de caracteres que puede llegar a tener una longitud máxima de 255. cada caracter de esta cadena ocupa un byte en memoria, y la cadena ocupa tantos Bytes como caracteres le hayan asignado.

Las cadenas se pueden definir del tamaño que se necesite. El comportamiento de los String es similar al de los arreglos.

```
Nombre := 'Pedro Pérez';
```

La anterior cadena tiene 11 caracteres, por tanto va a ser una variable que ocupa 11 Bytes en memoria. Una variable de tipo cadena al ser definida en la sección de variables se le está asignando un tamaño fijo en cuanto al número máximo de caracteres que puede almacenar en un momento determinado.

Ejemplo para lenguaje pascal:

```
VAR
        Nom : String[20];
          Dir    : String;
```

Ejemplo para lenguaje C

Char nom[10];

Cuando se define de tipo String y se restringe el número de caracteres, en el caso de nom tan solo a 20, se esta indicando al compilador que dentro del programa, máximo se le debe asignar una cadena de 20 caracteres o menos. Si un usuario asigna mas de lo que se estableció, simplemente se desechan los caracteres que sobren a la derecha. En el caso de la segunda definición (Dir), no se está restringiendo su tamaño por tanto es una variable que puede almacenar el máximo de caracteres para variables de este tipo (255); Vale anotar que durante la ejecución de un programa se puede modificar la longitud de la variable tipo String.

Algunas funciones de Turbo Pascal aplicadas a cadenas son las siguientes:

Fillchar: Función que llena el buffers de la variable de una caracter determinado. Si en un momento determinado se quisiera llenar la variable Nom con letras 'A' en lugar de hacer la asignación Nom:='AAAAAAAAAAAAAAAAAAAA', se puede realizar así:

Fillchar(Nom,20,'A'); Se le está diciendo que llene la variable Nom con 20 caracteres 'A'. si no se conoce el tamaño de la variable entonces se puede utilizar así:

Fillchar(Nom,sizeof(Nom),'A'); Se utiliza la función Sizeof para indicar que llene totalmente la variable Nom con el caracter 'A'.

TIPOS DE DATOS BOOLEAN

Algunos lenguajes de programación poseen este tipo de dato. El tipo boolean sólo tiene dos valores posibles: False (00) ó True(01). Una variable que se declare de tipo boolean tendrá asignado siempre uno de los dos valores, o verdadero o falso, y ocupa un solo byte de memoria RAM.

```
Program lógica;
Var     op : boolean;
Begin
        op := 2<3;              {toma el valor de True}
        If op then
                Write('verdadero! 2 es menor que tres')
```

```
        else
                write('El computador está loco!');
End;
```

En el ejemplo anterior se le asigna a Op un valor lógico que depende de la evaluación que se haga de ver si 2 es menor que 3. Como efectivamente es menor entonces se le asigna el valor de TRUE a op, y al evaluarlo en el si condicional (If) ejecuta la sentencia que está del lado del entonces (then en algunos lenguajes), de lo contrario algo raro andará pasando con su computador.

TIPOS DEFINIDOS POR EL USUARIO:

DATOS DE TIPO SUBRANGO:

La estructura más sencilla de definir es un subconjunto de datos tipo ordinal, esta estructura recibe el nombre de subrango. Si se escogen dos valores válidos, estos dos valores más todos los que queden entre los dos, forman el subrango de datos. Los datos que queden entre los datos definidos como inicio y final, corresponden a los datos que están entre esos dos valores en la tabla ascii. Por ejemplo los siguientes son datos tipo subrango.

```
Type
        Mayúsculas = 'A' .. 'Z';
        Minúsculas  = 'a' .. 'z';
        NumLiteral =  '0' .. '9'
        NumDigito  =  0 .. 9
```

En el caso anterior, se está definiendo una estructura Mayúsculas que tiene definidos caracteres que son A, B, C, D, E....Z; otra estructura Minúsculas que tiene definidos caracteres que son a, b, c, d, e, f, g, h.....z y así sucesivamente.

Cuando se declaran subrangos con apóstrofes, se está indicando que se trata de caracteres (Char), si se quitan los apóstrofes se tratará de números, caso del tipo NumDigito.

DATOS DE TIPO ENUMERADO

Un tipo enumerado se puede considerar como un tipo ordinal definido por el programador. Consiste en una lista de datos cuyos nombres no se pueden repetir. En el caso anterior es fácil declarar rangos porque ya están predefinidos en un código llamado ASCII, pero cuando se trate de otro tipo de datos, se hace necesaria su enumeración. Ej.: resulta difícil declarar una variable para que tome

como valores las frutas que están entre Mora .. Tomate; realmente no se podría. Para éste caso se hace necesario enumerar todos los valores que puede tomar esa variable.

```
Type
        fruta = (anon, mango, melón, mora, fresa, limón, naranja);
        color= (rojo, naranja, amarillo, verde, violeta, azul, negro);
```

La lista que forma un tipo enumerado se da siempre entre paréntesis

LENGUAJE C++

Posibles combinaciones de los tipos básicos de datos en lenguaje C y los modificadores con los tamaños y rango de bits comunes se muestran en la siguiente tabla. Los tipos básicos de datos en lenguaje son: Char, int, float, double bool y void. Algunos de ellos pueden ser modificados por los Modificadores Short, signed, unsigned, long. Si se utiliza un modificador Short se antepone al modificar %h, Si se utiliza el modificador long %l, Para el unsigned el modificador %u.

Tipo	Tam en Bits	Rango	Modificador	
			Lect	Escr
Char	8	-128..127	%c	%c
Unsigned char	8	0..255		
Signed char	8	-128..127		
Int	16	-32768..32767	%d %i	%d %i
Unsigned int	16	0..65535		
Short int	16	-32768..32767	%hd	%hd
Unsigned short int	16	0..65535		
Long int	32	-2,147,483,648..2,147,483,647		
Unsigned long int	32	0..4,294,967,295	%lu	%lu
Float	32	3.4E-38..3.4E+38	%f	%f
Double	64	1.7E-308..1.7E+308	%lf	
Long double	80	3.4E-4932..1.1E+4932	%Lf	%Lf
bool	1	True o false		

Solo los tipos de datos double y float aceptan números con punto flotante (números con fracciones decimales), en tanto que los otros tipos manejan únicamente enteros. Aunque es valido asignar un valor que contenga una fracción

decimal a un dato de tipo entero, dicha fracción se elimina y solo se asigna la parte entera.

Ejemplos de declaración de variables son:

```
int x,y;
unsigned short int y,k,a;
long double x,y;
int w=10;
```

Se puede al momento de declarar una variable asignarle un valor cualquiera.
Las variables en lenguaje C se declaran como locales a procedimientos o funciones y globales en el encabezado del programa.

LENGUAJE TURBO PASCAL

DATOS ENTEROS

NOMBRE	RANGO (desde - hasta)	TAM (Byte)	FORMATO
Integer	-32.768..32.767	2	Entero con signo
Word	0..65.535	2	Entero sin signo
Shortint	128..127	1	Entero corto con signo
Byte	0..255	1	Entero corto sin signo
Longint	-2.147.483.648..2.147.483.647	4	Entero largo con signo

DATOS REALES

NOMBRE	RANGO (desde - hasta)	TAM Bytes	CIFRAS SIGNIFICATIVAS
Real	2.9 e –39 1.7 e 38	6	11-12
Single	1.5 e –45 3.4 e 38	4	7-8
*Double	5.0 e 324 1.7 e 308	8	15-16
*Extended	1.9 e –4851 1.1 e 4932	10	19-20
*Comp	-9.2 e 18 9.2 e 18	8	18-19

*** Solo disponible con un coprocesador matemático 80x87 o en nivel de coprocesador simulado.**

Los tipos de datos reales diferentes al real, solo se pueden usar cuando existe u coprocesador matemático o una emulación del mismo mediante el uso de las directivas $N+ y $E-.

Lo anterior gracias a que Turbo Pascal a partir de la versión 5.0 emula en software el chip coprocesador matemático permitiendo utilizar los diferentes tipos de datos reales para ejecutar aplicaciones en máquinas que no posean este chip coprocesador.

Ejemplos de la forma como se declaran los datos en Turbo pascal.

```
Var
X       :char;
Y       :string(10);
A       :byte;
B,c     :integer;
D       :double;
E       :extended;
```

VISUAL BASIC: Posibles combinaciones de los tipos básicos de datos en basic y todos sus sucesores hasta visual basic y los modificadores con los tamaños y rango de bits comunes. Los tipos básicos de datos en lenguaje son: Integer, long, single, double, currency, string, byte, boolean, date, object y variant

Tipo	Tam byte	Rango	Tipo Carac
Byte	1	0..255	
Integer	2	-32768..32767	%
Long	4	-2147483648..2147483647	&
Single	4	-3.4E+38..3.4E+38	
Double	8	-1.79E-308..1.79E+308	
Currency	1	-22337203685477.58..922337203685477.58	
String	2	0..65535	$
Boolean	2	True o False	
Date	8	1/enero/100 .. 31/diciembre/9999	
Object	4	cualquier referencia a objeto	
Variant		16 con números, 22 con caracteres	

Ejemplos de declaración de variables son:

Dim X as integer
Dim X as byte
Dim X as double, Y as string

La declaración de variables en las versiones de Visual Basic se hacen locales a Objetos, generales a Formas o Globales a Proyectos. Las variables globales comparten variables entre formas diferentes que contenga el proyecto. En las demás versiones de Basic anteriores se declaran generales a todo el programa que se desarrolle.

En Matlab se manejan casi los mismo datos que manejan en el lenguaje c. Para saber que tipo de formato vasta simplemente con digitar en el prompt del MatLab la instrucción help format y enter (↵). Se mostrará una pantalla con información sobre formatos utilizados en MatLab.
» help format↵

FORMAT Set output format.

All computations in MATLAB are done in double precision.
FORMAT puede ser usado para escoger entre diferentes formatos de salida como sigue;

FORMAT Por defecto, es lo mismo que short.
FORMAT SHORT formato de punto fijo con 5 dígitos.
» pi↵ ans = 3.1416
FORMAT LONG formato de punto fijo con 15 digitos.
» pi↵ ans = 3.14159265358979
FORMAT SHORT E formato de punto flotante con 5 dígitos.
» pi↵ ans = 3.1416e+000
FORMAT LONG E formato de punto flotante con 15 dígitos
» pi↵ ans = 3.141592653589793e+000
FORMAT SHORT G Best of fixed or floating point format con 5 digitos.
» pi↵ ans = 3.1416
FORMAT LONG G Best of fixed or floating point format con 15 digitos.
» pi↵ ans = 3.14159265358979
FORMAT HEX formato Hexadecimal
» pi↵ ans = 400921fb54442d18
FORMAT + los símbolos +, - y blank son impresos para elementos positivos, negativos y zero. Partes imaginarias son ignoradas.
FORMAT BANK formato fijo para dolares y centavos.

» pi↵ ans = 3.14
FORMAT RAT Approximation by ratio of small integers.
» pi↵ ans = 355/113

El format RAT da el fraccionario mas aproximado al valor expuesto. En este caso el valor de ***PI***. No olvide que a pesar de que acá se escribe en mayúsculas cuando use el MatLab deberá escribir las funciones en minúscula.

Spacing:

FORMAT COMPACT Suppress extra line-feeds.
FORMAT LOOSE Puts the extra line-feeds back in.

ALGORITMOS Y PROGRAMAS

La principal razón para que las personas aprendan lenguajes de programación es utilizar la computadora como una herramienta para la resolución de problemas. Dos fases pueden ser identificadas en este proceso.

1. Fase de resolución del problema
2. Fase de implementación en una microcomputadora

El resultado de la primera fase es el diseño de un algoritmo para resolver el problema. Un algoritmo se puede considerar como el conjunto de instrucciones que conducen a la solución de un problema determinado. Dichas instrucciones deben tener una secuencia lógica para poder llegar a la solución real. El algoritmo se puede expresar de diversas maneras. Mediante símbolos, utilizando un lenguaje determinado para hablar la solución, describir sintéticamente dicha solución o simplemente escribiéndolo en cualquier código valido para algún lenguaje de programación. La última forma de describir el algoritmo es a lo que se le denomina PROGRAMA.

DISEÑO DE PROGRAMAS:

El diseño de programas no es tarea difícil. Un programa se construye teniendo en cuenta dos cosas que me facilitan cualquier lenguaje de programación: Las Estructuras de Programación y las Estructuras de Datos.

La utilización eficiente de cada una de las estructuras antes mencionadas permiten dar solución a cualquier problema que se desee resolver. En este libro se tratará con profundidad las estructuras de programación y se utilizara algunas estructuras de datos que sean necesarias para la solución de los ejemplos que se resuelvan. (Básicamente tienen que ver con variables simples y arreglos)

En la fase de resolución de cada uno de los ejemplos se incluyen etapas así:

1. Análisis del problema
2. Diseño del algoritmo
   Diagrama de flujo
   Seudo lenguaje
   Código
   Pascal
   Lenguaje C++
3. Verificación o prueba de escritorio

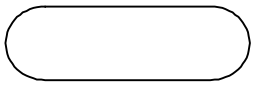
DIAGRAMAS DE FLUJO

Un diagrama de flujo es un dibujo que utiliza símbolos estándar de diagramación de algoritmos para computador, en el que cada paso del algoritmo se visualiza dentro del símbolo adecuado y el orden en que estos pasos se ejecutan. Se indica su secuencia conectándolos con flechas llamadas líneas de flujo porque indican el flujo lógico del algoritmo.

En esencia el diagrama de flujo es un medio de Presentación visual y gráfica de flujo de datos, a través de un algoritmo, las operaciones ejecutadas dentro del sistema y la secuencia en que se ejecutan.

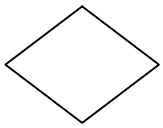
Los símbolos utilizados en los diagramas han sido normalizados por las organizaciones ANSI (American National Institute) y por ISO (International Standard Organization) aunque el uso de esos Estándar es voluntario.

SÍMBOLOS UTILIZADOS EN LA CONSTRUCCIÓN DE DIAGRAMAS DE FLUJO



Para iniciar y Terminar un algoritmo

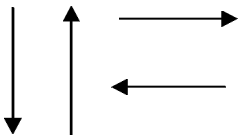
Para representar un proceso



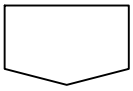
Para toma de decisiones: Símbolo utilizado tanto en decisiones como en estructuras cíclicas

Para representar entrada de datos

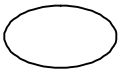
Para representar salidas de datos



Indican la dirección de flujo en un diagrama, además conecta todos los símbolos del diagrama



Conector entre diagramas



Conector dentro de un diagrama

REGLAS DE PROGRAMACIÓN:

Desde que las ideas de Knuth, Dijkstra y Wirth fueron consolidadas en el campo informático, las reglas para la construcción de algoritmos han ido variando constantemente y de igual forma los lenguajes de programación, en general se han ido adaptando a estos reglas o técnicas de programación.

Las reglas que se deben considerar en una buena programación son:

1. Diseñar algoritmos en etapas yendo de lo general a lo particular (método descendente)
2. Dividir el algoritmo en partes independientes -módulos- y tratar cada módulo independientemente.
3. Establecer y utilizar la solución de problemas técnicas de programación estructuradas
4. Dar especial importancia a las estructuras de datos
5. Describir completamente cada algoritmo
6. Verificar o realizar la prueba de escritorio a cada algoritmo desarrollado.

Un programa puede ser considerado como el conjunto de **Estructuras de datos** mas el conjunto de **Estructuras de programación** más el **Encabezado** que exige cada lenguaje en particular.

PROGRAMACIÓN ESTRUCTURADA

La programación estructurada es el conjunto de técnicas para desarrollar programas fáciles de escribir, verificar, leer y mantener

Se puede concretar mas la definición diciendo que la programación estructurada es el conjunto de técnicas que incluye:

- Un número limitado de estructuras de programación
- Diseño descendente
- Descomposición modular con independencia de los módulos

El teorema de Bohm y Jacopini establece que un programa propio puede ser escrito utilizando solo tres tipos de estructuras de control:

Secuencial
Selectiva
Repetitiva

Hoy me atrevería a decir que un programa propio puede ser escrito utilizando lo siguiente:

Declaraciones:

- Librerías de inclusión
- Declaración de funciones y/o procedimientos
- Definición de constantes y/o variables

Estructuras de programación:

- Asignación
- Decisión
- Cíclicas
- De selección múltiple

Estructuras de datos:

- Estáticas simples
- Dinámicas
- Registros
- Arreglos
- Archivos

Funciones:

- Predefinidas por el lenguaje
- Definidas por el usuario.

Generalmente cuando el algoritmo o solución de un problema determinado se deja en términos de diagrama de flujo, Seudo lenguaje e incluso en Seudo código se puede trabajar únicamente con estructuras de programación.

ESTRUCTURAS DE PROGRAMACIÓN

También llamadas estructuras de control por algunos autores. Son aquellas que le permiten a un usuario ejecutar tareas que a la final le permiten dar solución a problemas que se quieran resolver usando microcomputadoras.

En general se puede decir que las estructuras de programación son herramientas que el lenguaje le provee al usuario para solucionar problemas haciendo uso de computadoras.

Las estructuras de programación que tienen la mayoría de los lenguajes son cuatro así:

Estructuras de Asignación
Estructuras de Decisión
Estructuras Cíclicas
Estructuras de Selección múltiple.

ESTRUCTURA DE ASIGNACIÓN

Esta estructura se conoce también como sentencia en algunos lenguajes estructurados. Las estructuras de asignación, son utilizadas en el cuerpo de programa, procedimientos esclavos o funciones.

Una estructura de este tipo consiste en la asignación de una expresión a un identificador (comúnmente llamado variable) válido en un lenguaje de programación. La parte de esta estructura solamente puede estar ocupada por una variable.

Toda variable, literal o constante aparecerá formando la expresión al lado derecho.

Variable = Expresión

Expresiones simples

La expresión más simple consiste en un solo concepto: una simple variable, constante literal, constante numérica o constante simbólica.

PI	Constante simbólica definida por algunos lenguajes.
'20'	Constante literal definida por el usuario.
20	Constante numérica definida por el usuario.
I	Variable definida de algún tipo ofrecido por los lenguajes.

Expresiones complejas

Las complejas consisten en expresiones simples conectadas o relacionadas con operadores bien sean matemáticos o relacionales.

Ejemplos:

A + B
(A + B) * (C + B)

Operadores

Un operador es un símbolo que le da instrucciones al lenguaje de programación para que ejecute alguna operación, o acción, en uno o más operandos.

Un operando es algo sobre lo cual actúa un operador (podría considerarse como una expresión)

El operador de asignación

Permite evaluar una expresión y calculado su valor guardarlo en una posición de memoria asignando dicho valor al nombre de una variable. Dicho de otra manera, el valor calculado de la expresión queda referenciado a la variable a la cual se le asigna.

La forma general de uso es:

Lenguaje C	Lenguaje Pascal
Variable = Expresión lenguaje C	Variable:= Expresión versiones de Pascal.
X = Y	X := Y

En un programa codificado en algún lenguaje, la estructura de asignación del ejemplo anterior no significa que X es igual a Y. En cambio significa "asigne el valor de Y a X".

En cualquier lenguaje de programación el lado derecho representa cualquier expresión y el lado izquierdo debe ser un nombre de variable declarado previamente y valido en el lenguaje de programación.

Operadores matemáticos

La mayoría de los lenguajes utilizan cuatro operadores matemáticos a saber:

Operador	Símbolo	Acción	Ejemplo
Suma	+	Suma dos operandos	X + Y
Resta	-	Resta el segundo operando del primero	X - Y
Multiplicación	*	Multiplica sus dos operandos	X * Y
División	/	Divide el primer operando entre el segundo	X / Y

Cada lenguaje en particular involucra otra serie de operadores tales como:

Basic:

^ : Exponenciación

Lenguaje C:

Operadores Unarios:

++ : Incremento en uno
- - : Decremento en uno

Operador matemático

% : Módulo: Toma la parte residuo de una división: Ej: Si se divide 11 entre 4 da de resultado 2 y sobra de residuo 3. Si se hace X=11%3 entonces X toma el valor residuo de 3.

Lenguaje pascal

Mod : Para tomar la parte residual de una división. Similar al % del C.
Div : Para tomar la parte entera de una división.

El operador sizeof

Además de los operadores citados, comunes a otros lenguajes de programación, C utiliza también, el operador sizeof(), que le permitirá conocer la longitud, en bytes, que un tipo de dato ocupa en memoria.

Así, por ejemplo; sizeof(variable)

Devolverá 2 si la variable ha sido definida de tipo entera (int), 4 si la variable ha sido declarada de tipo real (float) y así sucesivamente para todos los tipos manejados por C.

El valor devuelto por sizeof() depende del compilador utilizado; no obstante, este operador que se puede utilizar con variables predefinidas por el sistema o definidas por el usuario, asegura la portabilidad del programa entre sistemas.

Análogamente: sizeof(tipodedato)

Retornará un valor que corresponderá al espacio reservado por el compilador para el tipo de dato especificado en el momento de compilar el programa.

Asi: sizeof(float)

Devolverá 4 que, como se puede observar en la tabla de tipo de datos, el valor corresponde al número de bytes que se reservan en memoria para cada variable que se declare del tipo de dato float.

Operadores al nivel de bits

El lenguaje C se diseño para sustituir en muchos trabajos al lenguaje ensamblador. Por ello, soporta un juego completo de operadores capaz de manejar los datos al nivel de bits.

Las operaciones al nivel de bits soportan solamente los datos de tipo caracter (char) y entero (int) con sus variantes. Son los siguientes:

& AND binario
| OR binario (ascii 124)

^ XOR binario (ascii 94)
~ NOT binario (tilde, ascii 126)
>> desplazamiento de bits a derecha
<< desplazamiento de bits a izquierdas
Los operadores AND OR y NOT se comportan igual que los anteriormente definidos pero al nivel de bits.

La tabla de verdad de estos operadores es la siguiente.

X	Y	X&Y	X/Y	X^Y	~X
0	0	0	0	0	1
0	1	0	1	1	1
1	1	1	1	0	0
1	0	0	1	1	0

Los operadores de desplazamiento de bits mueven todos los bits de una variable a la izquierda o a la derecha un número fijo de posiciones según se especifique.

Los formatos de estas instrucciones son los siguientes:

NombreVariable>> NumeroPosicionesDerecha

Precedencia de operadores

En una expresión que contiene mas de un operador aritmético diferente el computador evalúa dicha expresión de acuerdo a la siguiente jerarquía de operadores.

=	Jerarquía más baja
+ -	
/ * % mod div	
Negación(-), ++, - -	
Exponenciación	Jerarquía más alta

La tabla anterior indica que la primera operación aritmética que realiza, sería la exponenciación, luego la operación de negación y las operaciones unarias del lenguaje C, luego la multiplicación y/o la división, luego las sumas y las restas y por último la asignación quien tiene la jerarquía mas baja, es decir que seria la ultima operación que realizaría la computadora.

Cuando en una expresión existe operadores de igual jerarquía la computadora los resuelve de izquierda a derecha. Si la expresión contiene agrupadores (o sea

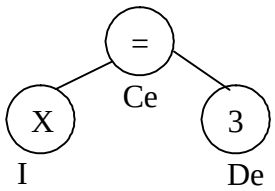
paréntesis) el destruye primero los paréntesis resolviendo lo que esté dentro de el y luego seguirá resolviendo las operaciones de acuerdo a lo descrito anteriormente.

Ejemplo: X = 3

La expresión anterior es sencilla. Consta tan solo de una variable X definida de un tipo entero y una expresión constante igual a tres.

En el ejemplo anterior se tiene un solo operador, por tanto el microprocesador asigna el valor de 3 a la variable X.

Visto simbólicamente se puede representar en un árbol de la siguiente manera



Para tener una mejor idea del orden en que efectúa cada una de las operaciones en una estructura de asignación es necesario identificar el tipo de notación con la cual trabaja el microprocesador.

Notación infija:

Es la notación con la cual escribimos todas las expresiones. En esta notación los operadores están entre los operandos

Si se realiza el recorrido en el árbol del ejemplo anterior se puede decir que la expresión estaría escrita en notación infija haciendo el siguiente recorrido:

Iz - Ce - De o sea X = 3

Notación postfija:

Es la notación que utiliza el microprocesador para resolver todas las expresiones que se asignen a una variable en una estructura de asignación o simplemente al solucionar una expresión. En esta notación los operadores estas después de los operandos.

Si se realiza el recorrido en el árbol del ejemplo anterior se puede decir que la expresión estaría escrita en notación postfija haciendo el siguiente recorrido:

Iz - De- Ce o sea X 3 = (Expresión en notación Postfija)

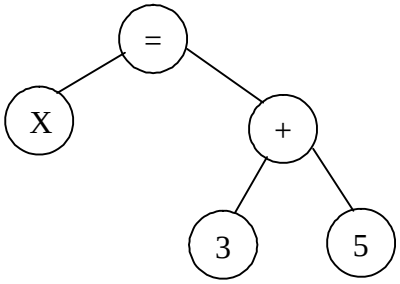
Ejemplo:

X = 3 + 5

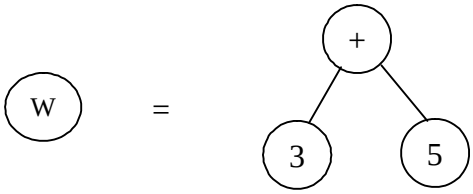
La expresión anterior consta de dos partes. Una variable X a la cual se le asignará (=) la expresión de la derecha. Antes de asignar un valor a la variable X el computador tendrá que resolver dicha expresión (3 + 5).

En el ejemplo anterior el microprocesador ejecuta primero la suma y el resultado lo asigna por ultimo a la variable X.

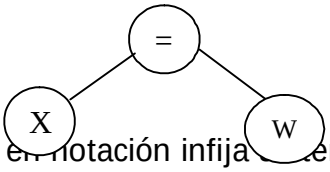
Visto simbólicamente, la estructura de asignación se puede representar en un árbol de la siguiente manera



Si la expresión de la derecha, o sea 3+5 se hace igual a W.



Se puede representar el árbol del ejemplo de la siguiente manera:



Si se expresa el anterior árbol en notación infija se tendrá: X= W

Si se representa en notación postfija se tendrá: X W =

Quiere decir lo anterior que primero se resolverá la expresión W y luego se asignará el valor encontrado en dicha evaluación a la variable X.

Ahora si se resuelve el árbol de la expresión se tendrá:

Notación infija	Notación postfija
W = 3 + 5	W = 3 5 +

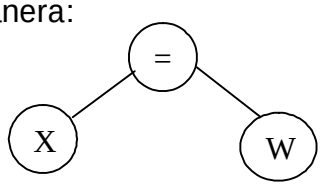
Tomando toda la expresión se tendrá entonces:

Notación infija	Notación postfija
X = 3 + 5	X 3 5 + =

Observando la última expresión, se notará que el microprocesador evaluará en notación postfija la estructura de asignación y los operadores los ejecuta de izquierda a derecha operando los dos operandos inmediatamente anteriores, o sea que sumará 3 y 5 y dicho valor (8), lo asignará a X.

Ejemplo:

Si se tiene la expresión X = A + B / C * E + F / G * H y se asigna al identificador W la expresión A + B / C * E + F / G * H, la estructura de asignación se puede representar de la siguiente manera:



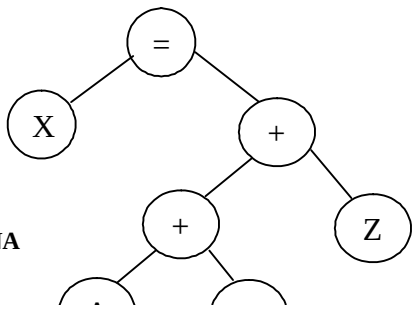
En la expresión W = A **+** B / C * E **+** F / G * H, se tienen varios operadores repetidos, caso particular la suma (+) que está presente dos veces.

Si al identificador Y se le asigna la expresión B / C * E ***"(Y=B/C*E)"*** y al identificador Z se le asigna la expresión F / G * H ***"Z=F/G*H"***, la expresión inicial puede escribirse entonces de la siguiente manera.

X = A + Y + Z

Tenga en cuenta que si hay operadores de igual jerarquía el microprocesador los resuelve de izquierda a derecha.

Al representar esta expresión en un árbol se tendrá lo siguiente:



Si se hace el recorrido en notación postfija al árbol que representa la expresión anterior quedará de la siguiente manera:

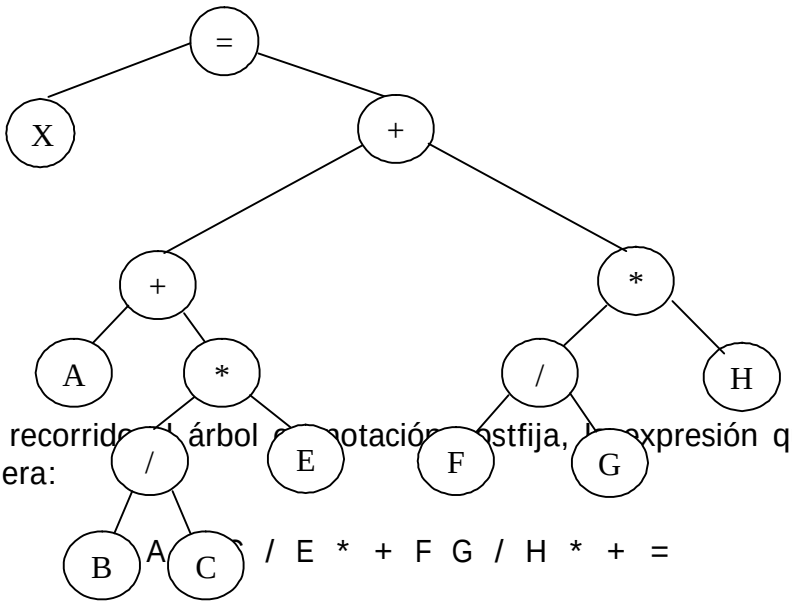
X A Y + Z + =

En el árbol anterior se puede observar que las operaciones se realizan de abajo hacia arriba o sea que primero se realiza la suma de A con Y. A este resultado se le suma el valor de Z y por último se asigna dicho valor a la variable X.

Donde Y y Z representan las expresiones:

Y = B / C * E Z = F / G * H

Si se involucran las expresiones Y y Z definidas anteriormente, el árbol quedará de la siguiente manera:



Si se hace el recorrido al árbol en notación postfija, la expresión quedará de la siguiente manera:

A B C / E * + F G / H * + =

Para pasar de notación infija a notación postfija, la máquina utiliza la teoría de colas para dicho trabajo. Se utiliza para tal efecto las listas de pila y cola.

Las pilas es la teoría de manejo de datos en donde el último dato que ingresa a la lista, es el primer dato que sale de ella (Last In First Out - LIFO) y las colas es la teoría de manejo de información en donde el primer dato en entrar, es el primer dato en salir. (First In First Out - FIFO).

Teniendo en cuenta lo anterior se trabajará de la siguiente manera para realizar el traslado de una expresión en notación infija a notación postfija. Cada uno de los operandos y operadores se van almacenando en un a lista de cola de entrada (LIFO). Una vez los datos ahí entonces se empiezan a pasar cada uno de los términos así: Cada operando pasará directamente a la cola de salida y cada operador pasa a una pila de espera de operadores.

Hay que tener en cuenta que cuando un operador llega a la pila de operadores, dicho operador desplaza a la cola de salida todos aquellos operadores que sean de igual o mayor jerarquía. "Ver jerarquía de operadores".

Todo operador debe esperar en la pila de operadores, es decir el operador que llega de la lista de cola de entrada, necesariamente debe esperar en la lista de pila de operadores.

Retomando el ejemplo anterior se tendrá lo siguiente:

Lista de cola de ENTRADA (FIFO) :

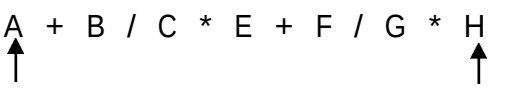
```
X = A + B / C * E + F / G * H
↑                           ↑
Cab                        Cola
```

Los identificadores **"Cab"** y **"Cola"** indican los elementos que se encuentran en la Cabeza "primer elemento en entrar" y Cola "último elemento en entrar" respectivamente de la Lista de cola en referencia. En el caso de la Lista de Pila el elemento que se encuentra listo a salir "último en entrar" será apuntado por el identificador Tope.

En teoría de Colas el elemento que está en la cabeza, fue el primer elemento en entrar a la lista y el elemento que está en la cola, fue el último elemento que entró en dicha lista.

Si se pasan los dos primeros elementos de la lista de entrada, es decir el operando **"X"** a la lista de cola de salida y el operador **"="** a la lista de pila de espera de operadores, las listas quedarán así:

Lista de cola de ENTRADA (FIFO) :



Pila de espera (LIFO) :

Lista de cola de SALIDA (FIFO) :

Para este primer paso, el operando "X" pasó directamente a la cola de salida. En estos momentos por ser el único elemento que existe en la lista de cola de salida, es considerado como cabeza y cola de la misma. Y el operador "=" paso a la pila de espera (es de anotar nuevamente que todo operador debe hacer espera en la pila de operadores).

Un segundo par de elementos de la cola de entrada serán enviados a la cola y pila de salida así: " A +"

Lista de cola de ENTRADA (FIFO) :

Pila de espera :

El operador (+) que llegó a la pila de espera de operadores, no desplazó el operador "=" que estaba en el tope de la pila, porque dicho operador "=" es de menor jerarquía que el operador "+".

Lista de cola de SALIDA (LIFO) : X A

Cab Cola

Observando lo anterior, en la cola de salida hay dos operandos "X A" y en la pila de espera dos operadores "+ =".

Un tercer par de elementos de la cola de entrada serán enviados a la cola y pila de salida así: "B /"

Lista de cola de ENTRADA (FIFO) : C * E + F / G * H

Cab Cola

Pila de espera : / + =

TOPE

El operador que llegó a la pila de espera no desplaza ningún operador de ella ya que los que se encuentran a la espera son de menor jerarquía.

Lista de cola de SALIDA (LIFO) : X A B

Cab Cola

Un cuarto par de elementos de la cola de entrada serán enviados a la cola y pila de salida así:

Lista de cola de ENTRADA (FIFO) : E + F / G * H

Cab Cola

Pila de espera : * + =

TOPE

Lista de cola de SALIDA (LIFO) : X A B C /

Cab Cola

Al llegar a la pila de espera el operador (*) desplaza a la cola de salida al operador (/) que se encontraba en el tope de la pila porque es de igual jerarquía.

Un quinto par de elementos de la cola de entrada serán enviados a la cola y pila de salida así:

Lista de cola de ENTRADA (FIFO) : F / G * H

Cab Cola

Pila de espera : + =

TOPE

Lista de cola de SALIDA (LIFO) : X A B C / E * +

Cab Cola

Al pasar de la cola de entrada el operador "+" a la pila de espera, este desplaza a la cola de salida los operadores "*" y "+" que se encuentran en el tope de la pila ya que son de mayor e igual jerarquía respectivamente. Primero desplazará el operador "*" y por último desplaza al operador "+".

Si se continúa pasando par de elementos hacia las respectivas listas se tendrá lo siguiente:

Lista de cola de ENTRADA (FIFO) : G * H

Cab Cola

Pila de espera : / + =

TOPE

Lista de cola de SALIDA (LIFO) : X A B C / E * + F

Cab Cola

El operador (/) pasó a la pila de espera. El no desplaza ningún operador porque el que está en el tope de la pila es de menor jerarquía.

Lista de cola de ENTRADA (FIFO) : H

Cab
Cola

Pila de espera : * + =

TOPE

Lista de cola de SALIDA (LIFO) : X A B C / E * + F G /

Cab Cola

Ahora nos queda tan solo un operando en la lista de cola de entrada. Al pasar a la cola de salida y quedar vacía la lista de cola de entrada, se procede a vaciar la lista de pila de operadores en su orden. primero saldrán los últimos que entraron así:

Lista de cola de SALIDA (LIFO) : X A B C / E * + F G / H * + =

Cab Cola

PLANTEAMIENTOS Y EJECUCIÓN DE PROGRAMAS

Cuando se inicia en el fascinante mundo de la programación de computadoras, es necesario a la hora de construir algoritmos primero que todo intentar plantear al nivel de símbolos dicha solución. A la construcción con los símbolos indicados es a lo que se conoce con el nombre de Diagrama de flujo. Recuerde que un algoritmo es la secuencia lógica de pasos que se sigue en la solución de un problema determinado.

En los siguientes ejercicios se verá la forma de aplicar la simbología de los diagramas de flujo en la solución de problemas de tipo matemático sencillos.

Aunque parezcan sencillos es una manera de adentrar un poco en el conocimiento de la diagramación estructurada. En dichos ejercicios se utilizarán funciones que permitan la entrada de datos por el teclado como medio estándar de entrada de datos de la máquina y funciones que me permitan manejar la pantalla como medio de salida mas utilizado en las computadoras para presentar información. Adicional a las funciones de entrada y salidas de datos se utilizará la Estructura de asignación en la solución de pequeñas expresiones de tipo matemático.

En los siguientes ejercicios se realiza el diseño de algoritmos al nivel de diagrama de flujo, Seudo lenguaje, Seudo código y Código en el lenguaje Pascal y C, que dan solución a cada una de las preguntas planteadas.

1. Desarrolle un algoritmo que le permita leer dos valores y escribir la suma de los dos.

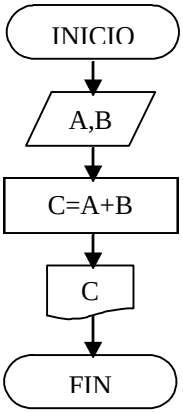
Análisis:

Para dar solución a este ejercicio es necesario realizar tres tareas: Leer los valores que para el caso concreto del ejemplo son dos (2), Calcular la suma de dichos valores y por último escribir el resultado obtenido de dicha suma. Cada una de las tareas planteadas se enmarcan dentro de un símbolo utilizado en diagramación así:

La lectura de cada dato desde el teclado se almacenará en variables, una guardará el primer valor que para el desarrollo se ha identificado con el nombre A y la otra el guardará el segundo valor que se ha denominado o identificado con el nombre B (estas operaciones se representarán en el símbolo de lectura. También se puede representar cada operación en símbolos aparte).

El cálculo de la suma se realizará y su valor será almacenado en la variable identificada como C (esta operación se representará en el símbolo de proceso) y por último el valor de respuesta almacenado en la variable C, se escribirá en la pantalla (esta operación se representa en el símbolo de escritura). Vea el diagrama siguiente.

Diagrama de Flujo



Seudo lenguaje

Inicio

Leer un valor y guardarlo en A y leer un segundo valor y guardarlo en B

Realizar la suma de A con B y guardar su valor en la variable C

Escribir el valor de C

Fin

Seudo código

```
Inicio
Leer(A,B)
C=A+B
Escribir(C)
Fin
```

Código Turbo Pascal:

```
Program cuadrado;
Uses crt;
N,R:real;
Begin
    Clrscr;
    Gotoxy(10,10);
    Write("Digite un número entero A ");
    Readln(A);
    Gotoxy(10,11);
    Write("Digite un número entero B ");
    Readln(B);
    C:= A+B;
    Gotoxy(10,12);
    Write("La suma  es: ",C);
    Repeat until keypressed;

End
```

Código Lenguaje C++:

```
#include "conio.h"
#include "stdio.h"
int A,B,C;
main()
{
  clrscr();
  gotoxy(10,10);
  printf("Digite un número entero ");
  scanf("%d",&A);
  gotoxy(10,11);
  printf("Digite un número entero ");
  scanf("%d",&B);
  C= A+B;
  gotoxy(10,12);
  printf("La suma es: %d ",C);
  getch()
    }
```

En el ejercicio anterior se hubiera podido escribir directamente el valor de la suma sin necesidad de almacenarlo previamente en la variable C utilizando para ello la siguiente sentencia.

```
Write("La suma  es: ",A+B);
Repeat until keypressed;
```

```
gotoxy(10,12);
printf("La suma es: %d ",A+B);
```

Tenga en cuenta que si lo hace de esta manera no estará utilizando estructura de asignación alguna.

2. Desarrolle un algoritmo que le permita leer un valor entero, calcular su cuadrado y escribir dicho resultado.

Análisis:

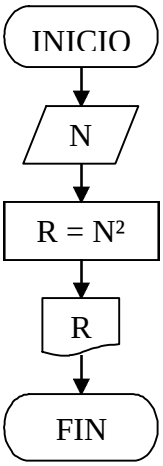
En el ejercicio planteado, se puede identificar como tareas a realizar, cada una de las tres acciones solicitadas. Leer, Calcular y Escribir.

Cada uno de los procesos mencionados se describen en símbolo respectivo quedando el diagrama de flujo de la siguiente manera.

Es necesario aclarar que cada proceso corresponde a una estructura de Asignación. En el caso del ejercicio el calcular el cuadrado.

Algoritmo:

Diagrama de flujo:



Seudo lenguaje:

Inicio

Leer el valor número entero y almacenarlo en la variable N.

Calcula el cuadrado del numero leído y que está almacenado en la variable N, y su resultado almacenarlo en la variable R.

Escriba el valor encontrado del cuadrado que esta almacenado en la variable R.

Fin

Seudo código:

Inicio
Leer(N); R = N²; Escribir (R)
Fin

Código Turbo Pascal:

```
Program cuadrado;
Uses crt;
N,R:real;
Begin
    Clrscr;
    Gotoxy(10,10);
    Write("Digite un número entero N");
    Readln(N);
    R=exp(2*ln(N));
    Gotoxy(10,12);
    Write("El cuadrado del numero es: ",R);
    Repeat until keypressed;

End
```

Código Lenguaje C++:

```
#include "math.h"
#include "conio.h"
#include "stdio.h"
int N,R;
main()
{
  clrscr();
  gotoxy(10,10);
  printf("Digite un número entero N");
  scanf("%d",&N);
  R=pow(N,2);
  gotoxy(10,12);
  printf("El cuadrado es: %d ",R);
  getch()
  }
```

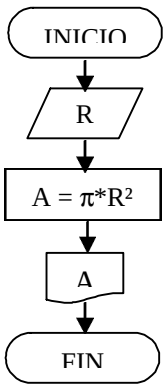
3. Desarrolle un algoritmo que le permita leer un valor para radio (R), calcular el área (A) de un círculo $A = \pi * R^2$ y escribir su valor.

Análisis: En el ejercicio se identifican como tareas las tres acciones solicitadas. Leer, Calcular y Escribir. Cada uno de los procesos mencionados se describen en el símbolo respectivo quedando el diagrama de flujo de la siguiente manera.

Algunos lenguajes de programación traen incorporada una constante con el valor del número pi "3.1415...". Dicha constante es PI. Para utilizarla simplemente se hace referencia a ella.

Algoritmo:

Diagrama de flujo: Seudo lenguaje:



Inicio

Leer el valor del radio y guardarlo en la variable R

Calcular el valor del área aplicando la formula $\pi*R^2$ y almacenar dicho valor en la variable A

Escribir el valor encontrado para el área guardado en la variable A.

Fin

Seudo código:

Inicio
Leer(R)
$A = \pi * R^2$
Escribir (A)
Finalizar

Código Turbo Pascal:

```
Program área;
Uses crt;
Const pi=3.1415926;
R,A:real;
Begin
    Clrscr;
    Gotoxy(10,10);
    Write("Digite el valor de Radio R");
    Readln(R);
    A = pi*exp(2*ln(R));
    Gotoxy(10,12);
    Write("El valor del área es: ",A);
    Repeat until keypressed;

End.
```

Código Lenguaje C++:

```
#include "math.h"
#include "conio.h"

#include "stdio.h"
float R, A;
main()
{
     clrscr();
     gotoxy(10,10);
     printf("Digite el Radio R");
     scanf("%f",&R);
     A = pi*pow(R,2);
     gotoxy(10,12);
     printf("El área es: %f ", A);
     getch();
        }
```

4. Determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de sus dos catetos. Desarrolle los correspondientes algoritmos.

Análisis: En el ejercicio se puede definir como tareas las tres acciones solicitadas. "Leer, Calcular y Escribir", Leer cada uno de los valores de los dos catetos y almacenarlos en cada uno de los identificadores definidos para el caso, calcular la hipotenusa aplicando la fórmula correspondiente almacenando su valor en el identificador del caso y escribir el valor encontrado para la hipotenusa como respuesta. Cada una de las acciones se describen dentro del símbolo respectivo, quedando el diagrama de flujo de la siguiente manera.

Algoritmo:

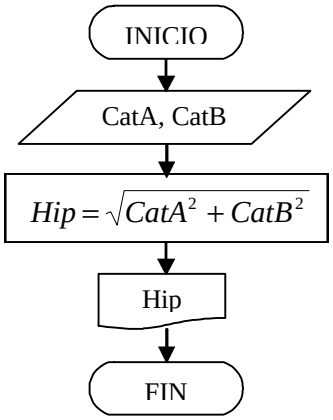
Cuando se requiere desarrollar operaciones matemáticas tanto en el diagrama de flujo como en el Seudo lenguaje se dejan planteadas tal y cual como se escriben normalmente de manera simbólica. En el momento de codificar dicho algoritmo se traduce el símbolo a la función correspondiente en el lenguaje.

Caso específico del ejemplo anterior la raíz cuadrada en código es sqrt y el símbolo es el que conocemos ".||.

Para el desarrollo del siguiente ejercicio se trabajará con las siguientes variables:

CatA : Cateto del triangulo
CatB : Cateto del triangulo
Hip : Hipotenusa del triangulo

Diagrama de flujo



Seudo lenguaje

Inicio

Leer el valor de cada cateto y almacenarlo en la variable CatA y CatB

Calcular el valor de Hip con la formula indicada

Escribir el valor de la Hip

Fin

Seudo código

En el seudo código se dejan planteadas las operaciones tal y cual se hizo en el diagrama de flujo o en el seudo lenguaje.

Inicio
Leer(CatA,CatB)
$HIP = \sqrt{CatA^2 + CatB^2}$
Escribir(Hip)
Fin

Código Turbo Pascal:
"programa que calcula el área de un triángulo rectángulo conociendo el valor de sus catetos."

```
Program área;
Uses crt;
CatA,CatB,HIP:real;
Begin
   clrscr;                gotoxy(10,10);
   Write("Digite el valor CatetoA");
   Readln(CatA);          gotoxy(10,12)
   Write("Digite el valor CatetoB");    Readln(CatB);
   Hip= sqrt(exp(2*ln(CatA))+exp(2*ln(CatB)))
   Gotoxy(10,14);
   Write("El valor HIPOTENUSA= ",HIP);
   Repeat until keypressed;
End.
```

Código Lenguaje C++:

```
/* Programa que calcula el valor del área de un triángulo
rectángulo conociendo sus catetos */
#include "math.h"
#include "conio.h"
#include "stdio.h"
float CatA,CatB,HIP;
main()
{  clrscr();              gotoxy(10,10);
   printf("Digite el valor CatetoA");   scanf(&CatA);
   gotoxy(10,12);       printf("Digite el valor CatetoB");
   scanf(&CatB);
   HIP = sqrt( pow(CatA,2) + pow(CatB,2 ))
   gotoxy(10,14);
   printf("El valor de HIPOTENUSA= %f",HIP);
   getch();      }
```

Para digitarlo en MatLab es necesario guardar el siguiente código en el directorio Work de MatLab y ejecutarlo desde el prompt del MatLab digitando solo el nombre con el cual fue guardado.

```
cata=input('Digite el valor Cateto A : ');
catb=input('Digite el valor Cateto B : ');
hip=sqrt(cata^2+catb^2);
fprintf('El valor de HIPOTENUSA = %f \n',hip);
```

Nota: Para el caso de MatLab no es necesario declarar la librería matemática porque se supone que el ya la tiene.

Si necesita ayuda en el prompt digite help sqrt o help sym/sqrt.m

5. Desarrolle un algoritmo que le permita leer un valor que represente una temperatura expresada en grados Celcius y convierta dicho valor en un valor expresado en grados Fahrenheit.

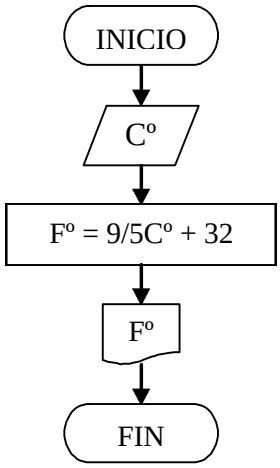
Análisis: En el ejercicio planteado, las tareas a realizar son: Leer, convertir y Escribir.

Tareas:

Leer la temperatura en grados Celcius
Calcular la conversión a grados Fahrenheit
Escribir el valor encontrado

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer los grados centígrados y almacenarlos en la variable c

Asignar a la variable f la conversión de grados centígrados a Fahrenheit dado por la fórmula 9/5c+32

Escribir el valor de la variable f

Fin

Seudo código

Inicio
Leer(C)
F = 9/5*C + 32
Escribir(F)
Fin

Al momento de definir el tipo de variables a utilizar, hay que tener en cuenta que en lenguaje C cuando se divide dos variables de tipo entero el resultado igualmente será del tipo entero.

Código Turbo Pascal:

```
Program convertir;
Uses crt;
C,F:real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite el valor expresado en Grados °C");
  readln(C);
    F = C*9/5+32;
    gotoxy(10,12);
write("El valor en grados  farenheit es : ",F);
    repeat until keypressed;
End.
```

```
/ * Código Lenguaje C: * /
#include "conio.h"
#include "stdio.h"
float c, f;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite el valor expresado en Grados °C");
    scanf("%f",&c);
    f= c*9/5+32;
    gotoxy(10,12);
    printf("El valor en grados Farenheith es: %f ", f);
    getch();
}
```

Bajo MatLab, guarda el siguiente código en un archivo y lo llama desde el prompt del MatLab.

```
C = input('Digite el valor expresado en Grados °C : ');
f = c*9/5+32;
fprintf('El valor en grados Farenheith es : %8.3f\n', f);
```

Nota: El formato 8.3f indica se utilizarán 8 espacios para escr5ibir el valor de la variable f, y de los ocho espacios se utilizan 3 campos para los decimales. El \n indica que cuando termine salte el prompt a una nueva línea (New Line). Si se omite la instrucción \n entonces el prompt quedara unido a la último digito escrito del valor f.

6. Desarrolle un algoritmo que le permita calcular el área de un triángulo en función de las longitudes de sus lados previamente leídos desde el teclado.

Análisis:

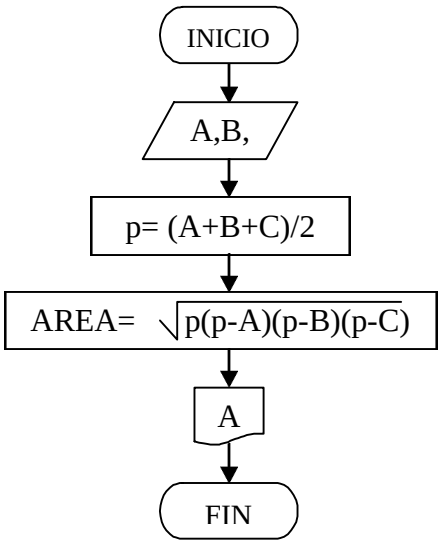
Para darle solución al problema planteado primero se leen los valores de los tres lados del triángulo, luego se procede a calcular el valor de su área en función de los valores leídos, aplicando la fórmula correspondiente y por último se da orden de escribir el valor calculado.

Tareas:

Leer los lados
Calcular el área
Escribir el valor calculado

Algoritmo:

Diagrama de flujo



Seudo Lenguaje

Inicio

Leer(a,b,c)

Hacer p=(A+B+C)/2

Calcular el valor del área dada por la fórmula

Escribir (a)

Fin

Código Turbo Pascal:

```
Program Area;
Uses crt;
A,B,C,AREA,p      : real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite el valor de A");
    readln(A);
    gotoxy(10,12)
    write("Digite el valor de B");
    readln(B);
    gotoxy(10,14)
    write("Digite el valor de C");
    readln(C);
    p=(A+B+C)/2
    AREA:= sqrt(p*(p-A)*(p-B)*(p-C))
    gotoxy(10,16);
    write("El valor del área es : ",AREA);
    repeat until keypressed;
End.
```

Código Lenguaje C++:

```
#include "conio.h"
#include "stdio.h"
float A,B,C,AREA,p;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite el valor de A");
    Scanf(&A);
    gotoxy(10,12)
    printf("Digite el valor de B");
    Scanf(&B);
    gotoxy(10,14)
    printf("Digite el valor de C");
    Scanf(&C);
    p=(A+B+C)/2
    AREA=
     sqrt(p*(p-A)*(p-B)*(p-C))
    gotoxy(10,16);
    printf("El valor del área es: %f ", AREA);
    getch();
}
```

En MAtLab se guardará las siguientes lineas en un archivo y luego se llama para ser ejecutado.

```
A = input('Digite el cateto a : ');
b = input('Digite el cateto b : ');
c = input('Digite el cateto c : ');
p = (a+b+c)/2;
area = sqrt(p*(p-a)*(p-b)*(p-c));
fprintf('El valor del área es : %8.3f\n', area);
```

7. Desarrolle un algoritmo que le permita determinar el área y volumen de un cilindro cuyo radio (R) y altura (H) se leen desde teclado.

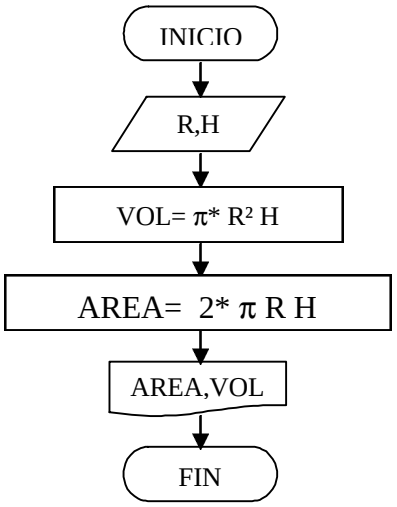
Análisis: Para dar solución al ejercicio planteado es necesario leer el valor del radio y el valor de la altura del cilindro desde algún medio de entrada de datos a la computadora, generalmente el teclado. Una vez se tengan almacenados los dos valores en memoria del computador se procede a calcular el área y el volumen aplicando las fórmulas respectivas.

Tareas:

Iniciar
Leer valores de R y H
Calcular el volumen y área
Escribir los valores respectivos
Finalizar

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer el valor de Radio (R) y Altura (H)

Calcular el Volumen aplicando la fórmula

Calcular el valor del área aplicando la fórmula respectiva

Escribir el valor del Área y del Volumen

Finalizar

Seudo código

Inicio
Leer(R, H)
VOL= π *R²H
AREA= 2*π *RH
Escribir(AREA, VOL)
Fin

Código Turbo Pascal:

```
Program Area_Volumen;
Uses crt;
R,H,AREA,VOL:real;
Begin
    clrscr;        gotoxy(10,10);
    write("Digite el valor de R");
    readln(R);   gotoxy(10,12)
    write("Digite el valor de H");
    readln(H);
    VOL := π* R*R *H
    AREA :=  2 *π * R* H
    gotoxy(10,16);
    write("El valor del area es : ",AREA);
    gotoxy(10,17);
    write("El valor del Volumen es : ",VOL);
    repeat until keypressed;
End.
```

Código Lenguaje C++:

```
/*  Programa Area_Volumen  */
#include "conio.h"      #include "math.h"
#include "stdio.h"
float R,H,AREA,VOL;
main()
{   clrscr();      gotoxy(10,10);
    printf("Digite el valor de R");
    readln(R);   gotoxy(10,12)
    printf("Digite el valor de H");
    readln(H);
    VOL= π*pow( R,2)* H
    AREA=  2 *π * R* H
    gotoxy(10,16);
    printf("El valor del área es :  %f",AREA);
    gotoxy(10,17);
    printf("El valor del volumen es :  %f",VOL);
    getch();      }
```

Para desarrollarlo en MatLab se graba el siguiente código en un archivo que después será ejecutado desde el prompt del MatLab.

```
R = input('Digite el valor de R :');
H = input('Digite el valor de H :');
VOL = pi*R^2*H;
AREA = 2*pi*R*H;
fprintf('El área del cilindro es : %f \n',AREA);
fprintf('El volumen del cilindro es : %f \n',VOL);
```

Tenga en cuenta de que si declara variables con mayúsculas se debe usar en el desarrollo con mayúsculas. La función pise puede utilizar así porque MatLab la trae definida.

8. Desarrolle un algoritmo que le permita calculas el área (A) de un segmento de círculo.

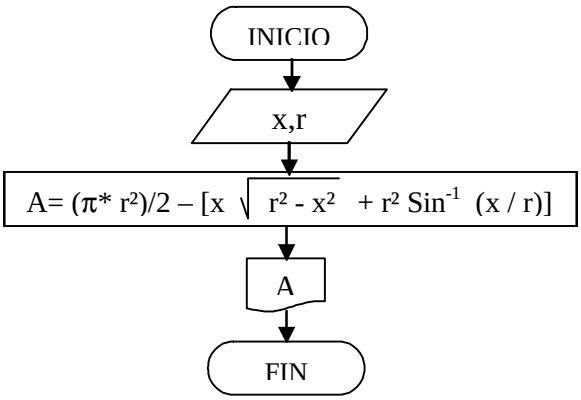
Análisis: Para calcular el área de un segmento de círculo lo primero que hay que hacer es leer el valor del radio del vinculo y leer el valor de X que es la distancia del centro al segmento. Una vez leído dichos valores se calcula aplicando la fórmula respectiva y por último se escribe el valor del área.

Tareas:

Leer (X y r)
Calcular (Area)
Escribir

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer el valor de X y r

Calcular el valor de área

Escribir el valor de Area (A)

Fin

Código Turbo Pascal:

```
Program Area_Segmento_Circular;
Uses crt;

x,r,A:real;
Begin
      clrscr;
      gotoxy(10,10);
      write("Digite el valor de R");
      Readln(r);
      gotoxy(10,12)
      write("Digite el valor de X");
      Readln(x);
      A= (π*r*r)/2 –(sqrt(r*r-x*x)+r*r*asin (x / r))
      gotoxy(10,16);
      write("El valor del area es : ",A);
      repeat until keypressed;
End.
```

Código Lenguaje C++:

```
/*  Programa Area_Segmento_Circular */
#include "conio.h"
#include "stdio.h"
float R,H,AREA,VOL;
main()
{    clrscr();
       gotoxy(10,10);
       printf("Digite el valor de R ");
       scanf("%f",&r);
       gotoxy(10,12)
       printf("Digite el valor de X ");
       scanf("%f",&x);
       A= (π* r²)/2 – [sqrt(r*r –x*x)+r*r*asin (x / r)]
       gotoxy(10,16);
       printf("El valor del area es : %f",A);
       gecth();
}
```

Otras salidas literales como salidas con datos numéricos reemplazando la función "printf" son:

printf("El valor del área es %f",A);
cout<<"el valor del área es"<<A;
puts (A);

Hay que tener en cuenta que hay que incluir las librerías de estas funciones.

Otras formas de entrar datos reemplazando la función "scanf" son:

scanf("%d",&A);
cin>>A;
gets(A);

ESTRUCTURA DE DECISIÓN

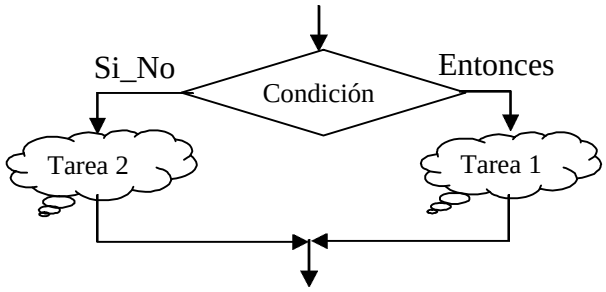
Cuando el programador desea especificar dos caminos alternativos en un algoritmo se deben utilizar estructuras de decisión.

Una estructura de decisión dirige el flujo de un programa en una cierta dirección, de entre dos posibles, en función de un valor booleano. En lenguajes de programación estructurados la estructura condicional es la IF / ELSE. La cláusula ELSE en esta estructura es optativa. La forma en que trabaja esta sentencia resulta casi evidente a partir de la lógica de la lengua inglesa: Si (IF) la expresión booleana resulta cierta (TRUE), entonces la sentencia se ejecuta. Si la expresión booleana resulta falsa (FALSE), el control pasa a la siguiente (en orden descendente) instrucción del programa.

Estructura Sencilla:

Forma general de uso:

Si (Condición) entonces
    Ejecuta bloque de instrucciones uno
Si_no
   Ejecuta bloque de instrucciones dos
Fin si



Una estructura de decisión por sencilla o compleja que sea debe tener solo una entrada y una salida. (Vea estructura de diagrama anterior).

Un bloque de instrucciones (Denominado tarea 1 y 2 en el diagrama anterior) puede ser un conjunto de estructuras de cualquier clase (asignación, Decisión, Cíclicas o de selección múltiple) que se ejecutan unas tras de otras.

El bloque de instrucciones tarea1 se ejecuta en caso de que la condición que se coloque sea verdadera, En caso de ser falsa la evaluación de la condición se ejecuta el bloque de instrucciones tarea2.

Operadores de Relación:

Una condición tiene que ver directamente con una pregunta. La pregunta se forma mínimo con dos operandos y un operador de relación.

Para tales efectos los operadores de relación utilizados en los lenguajes son:

	Pascal	Lenguaje C
Mayor	>	>
Menor	<	<
Mayor o igual	>=	>=
Menor o igual	<=	<=
Igual	=	==
Diferente	<>	!=

Así, con base en los operadores de relación descritos, algunos ejemplos de preguntas o condiciones son:

Pascal	Lenguaje C	
a>b	a>b	A es mayor que B?
b>c	b>c	B es mayor que C?
a=4	a==4	A es igual a 4?
b<>c	b!=c	B es diferente de C?

También se pueden desarrollar condiciones con varias preguntas. Para el efecto anterior es necesario plantear las dos preguntas tal como se indicó y relacionarlas con operadores lógicos.

Operadores Lógicos:

Los operadores lógicos utilizados en los lenguajes estructurados son:

	Pascal	Lenguaje C
y	and	&&
ó	or	||
No	not	!=

Con base en lo anteriormente expuesto, las preguntas o condiciones compuestas podrían quedar de la siguiente manera.

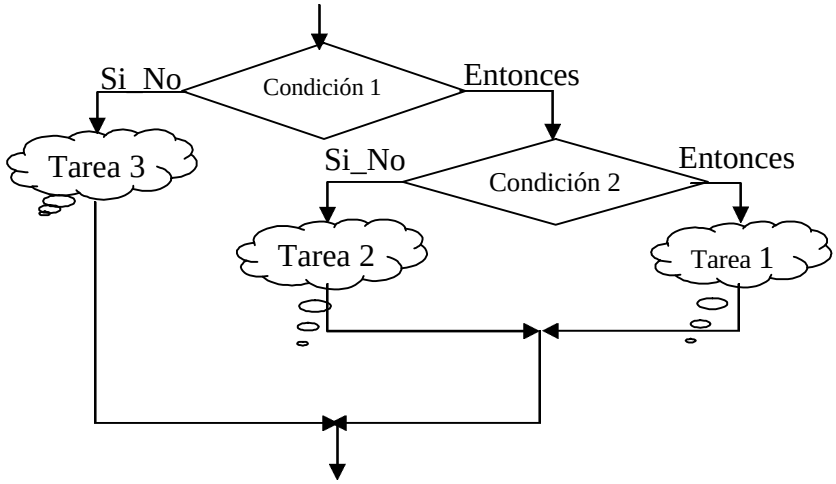
Pascal	Lenguaje C
(a>b) and (b>c)	((a>b)&&(b>c))

"A es mayor que B y B es mayor que C"

(a<b) or (b<c) ((a<b)11(b<c))
"A es menor que B y B es menor que C"

Para el lenguaje C Las condiciones es necesario que estén entre paréntesis.

Estructuras de decisión anidadas:



Una estructura de decisión puede estar anidada dentro de otra estructura de decisión. Hay que tener en cuenta que el anidamiento sea total.

El inicio y el final de la estructura anidada debe quedar totalmente dentro del inicio y el final de la estructura que permite dicho anidamiento.

Se debe mantener el concepto que una estructura de decisión debe tener una sola entrada y una sola salida tanto para la estructura que anida como para la estructura anidada.

En el diagrama anterior se puede observar una estructura anidada del lado verdadero de la condición de la primera decisión.

Seudo lenguaje de la estructura de decisión anidada.

```
Si (Condición 1) entonces
   Si (Condición 2) entonces
        Ejecuta bloque de instrucciones tarea 1
   Si_no
        Ejecuta bloque de instrucciones tarea 2
```

Fin_si
Si_no
Ejecuta bloque de instrucciones tarea 3
Fin si.

En el anterior Seudo lenguaje se puede observar que el anidamiento es total o sea que el inicio de la condición y el final están totalmente encajados dentro del lado verdadero de la primera estructura.

Ejercicios Aplicando Estructuras de Decisión

Para los siguientes ejercicios utilice el siguiente planteamiento

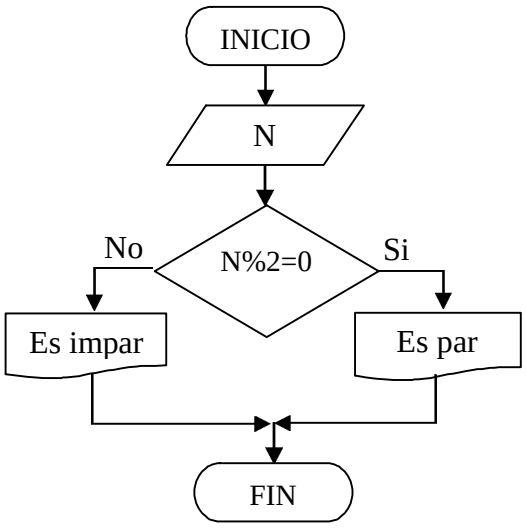
1. Desarrolle un algoritmo que le permita leer un valor cualquiera N y escribir si dicho número es par o impar.

Análisis:

Para resolver el ejercicio planteado hay que leer primero el valor y almacenarlo en una variable (N). Para saber si el numero es par, la división por dos (2) debe ser exacta. Por tanto la decisión si la división es exacta o no, la debe tomar el computador usando el símbolo respectivo en el diagrama. (En los diagramas se utilizará el símbolo % para representar el residuo de la división de dos operandos, en este caso de la división de N y 2). Una vez resuelva la aserción el computador escribirá si es par o impar el numero leído y almacenado en la variable N.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un numero y almacenarlo en variable N

Si el residuo de dividir a N entre 2 igual a cero

Entonces Escriba que es par
Sino Escriba que es impar

Fin

Seudo código

```
Inicio
Leer (N)
Si (N residuo 2 = 0) entonces
        Escriba (N es par)
Si_no
        Escriba (N es impar)
Fin_si
Fin
```

Código Turbo Pascal:

```
Program par_impar;

Uses crt;
N: integer;

Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un número ");
    readln(N);
    gotoxy(10,12)
    if N mod 2 = 0 then
       write("El numero es par")
    Esle
       write("El numero es impar");
    repeat until keypressed;
End.
```

Código Lenguaje C

```
/* programa para decir si un numero
es par o impar */
#include "conio.h"
#include "stdio.h"
int  N;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un número ");
    scanf("%d",&N);
    gotoxy(10,12);
    if (N%2==0)
        printf("el número es par");
    else
        printf("el número es impar");
    getch();
}
```

Digitado en MatLab, se guarda bajo un nombre de archivo y se ejecuta desde el prompt del MatLab.

```
N = input('Digite un número ');
if mod(N,2) == 0
  fprintf('el número es par \n');
else
  fprintf('el número es impar \n');
end
```

Es importante resaltar la diferencia que existe entre "=" y "==", el primero para asignar y elsegundo para comparar. En el casodel MatLab la sentencia if se cierra con un end, ya que no trabaja con marcas como lo hace C "{ y }"o Pascal "Begin y end". En el caso de C o Pascal omiten las marcas cuando hay una sola instrucción por alguna de las dos opciones.

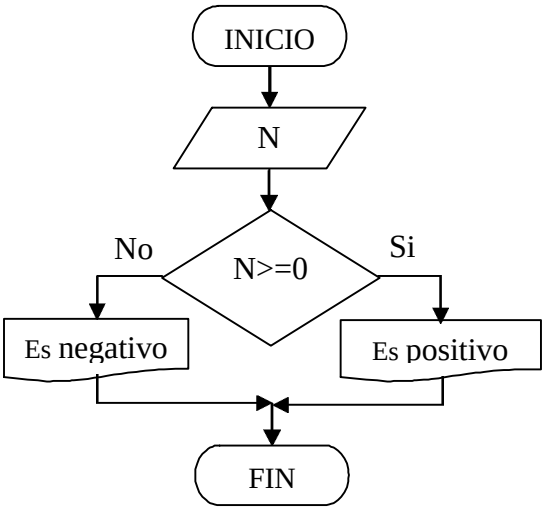
2. Desarrolle un algoritmo que le permita leer un valor cualquiera N y escribir en la pantalla si dicho número es Positivo o Negativo

Análisis:

Para resolver el ejercicio planteado hay que leer primero el valor del número y almacenarlo en una variable (N). Para saber si dicho número es positivo o no, simplemente lo compara con cero. En caso de que sea mayor o igual a cero entonces el número será positivo, y para el caso contrario el número será negativo.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer en número y almacenarlo en variable N

Si el número es mayor o igual a cero
Entonces
escriba que es positivo
Si no
escriba que es negativo

Fin

Seudo código

```
Inicio
Leer(N)
Si (N>=0) entonces
        Escriba("Es positivo")
Sino
```

```
        Escriba("Es negativo")
Fin si
Fin
```

Código en Turbo Pascal

```
Program positivo_Negativo;
Uses crt;

N: integer;
Begin
        clrscr;
        gotoxy(10,10);
        write("Digite un número ");
        Readln(N);
        gotoxy(10,12)
        if N >= 0 then
           write("El número es positivo")
        Esle
           write("El número es negativo");
        repeat until keypressed;
End.
```

Código Lenguaje C

```
/*  programa positivo o negativo  */
#include "conio.h"
#include "stdio.h"
int  N;
main()
{       clrscr();
        gotoxy(10,10);
        printf("Digite un numero ");
        scanf("%d",&N);
        gotoxy(10,12);
        if (N>=0)
           printf("el número es positivo");
        else
           printf("el número es negativo");
        getch();
}
```

El siguiente código lo guarda bajo un nombre de archivo.m y lo ejecuta desde el prompt del MatLab.

```
N = input('Digite un número ');
if N>=0
  fprintf('el número es positivo \n');
else
  fprintf('el número es negativo \n');
end
```

Nota: Recuerde que el archivo debe quedar en el directorio Work o un directorio que haya configurado en la opción Set Path de la opción File del menú del MatLab

3. Desarrolle un algoritmo que le permita leer un valor cualquiera N y escribir si dicho número es múltiplo de Z.

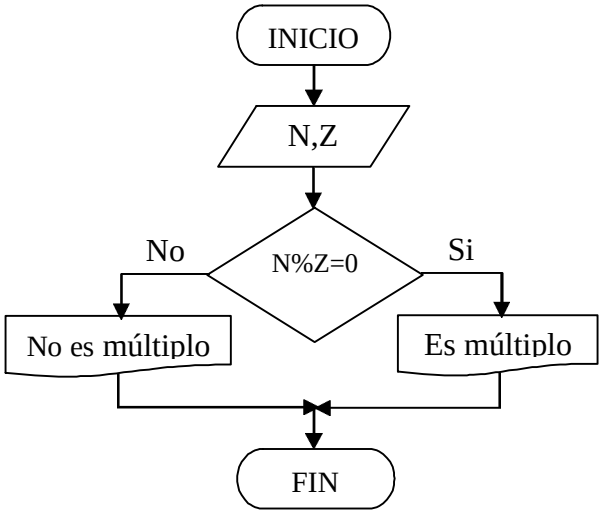
Análisis:

Para resolver el ejercicio planteado anteriormente, hay que leer primero el valor del número y almacenarlo en una variable (N). Luego leer otro valor y almacenarlo en la variable Z. Para saber si el número almacenado en la variable N

es múltiplo del número almacenado en Z, se hace la división entre Z y N, si la división es exacta entonces N es múltiplo de Z, de lo contrario N no será múltiplo de Z.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en la variable N y leer otro valor y almacenarlo en la variable Z

Si la división entre N y Z es exacta

Entonces escriba N es múltiplo de Z
Sino Escriba N no es múltiplo de Z

Fin

Seudo código

```
Inicio
Leer (N,Z)
Si (N residuo Z=0) entonces
        Escriba (N es múltiplo de Z)
Sino
        Escriba(N no es múltiplo de Z)
Fin si
Fin
```

Código Turbo Pascal:

```
Program multiplo_de_Z;
Uses crt;
Z,N: integer;
Begin
   clrscr;
   gotoxy(10,10);
   write("Digite un número N");
   Readln(N);
   gotoxy(10,11);
   write("Digite un número Z");
   Readln(Z);
   gotoxy(10,12);
   if N mod Z = 0 then
      write("N es múltiplo de Z")
   Esle
      write("N no es múltiplo de Z");
   repeat until keypressed;
```

Código Lenguaje C

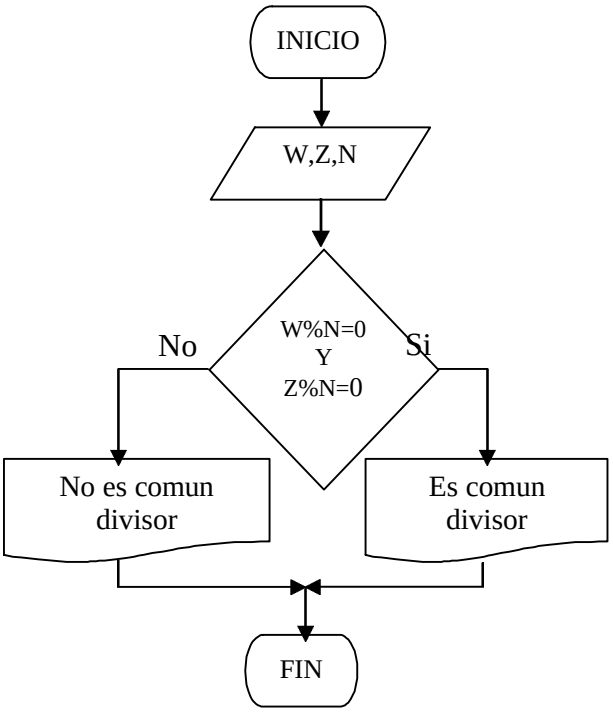
```
/*  Programa múltiplo_de_Z  */
#include "conio.h"
#include "stdio.h"
int  N,Z;
main()
{  clrscr();
   gotoxy(10,10);
   printf("Digite un número N ");
   scanf("%d",&N);
   gotoxy(10,11);
   printf("Digite un número Z ");
   scanf("%d",&Z);
   gotoxy(10,12);
   if (N%Z==0)
      printf("N es múltiplo de Z");
   else
      printf("N no es múltiplo de Z");
      getch();
```

4. Desarrolle un algoritmo que le permita leer un valor cualquiera N y escribir si dicho número es común divisor de otros dos valores leídos W y Z

Análisis: Para resolver el ejercicio planteado, hay que leer primero el valor del número y almacenarlo en una variable (N). Leer dos valores mas y almacenarlos en las variables W y Z respectivamente. Para saber si el valor almacenado en la variable N es común divisor de W y Z, se chequea para ver si la división entre W/n y Z/n son exactas. En caso de ser exactas entonces el valor numérico almacenado en la variable N es común divisor de los dos. En caso contrario no lo será.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer 3 valores y almacenarlos en las variables W, Z y N.

Si W residuo N es igual a 0 y Z residuo N es igual a 0 Entonces
    Escriba N es común divisor de W y Z
Sino
    Escriba N no es común divisor de W y Z
Fin.

Seudo código

Inicio
Leer (W,Z,N)
If (W residuo N=0 y Z residuo N=0) entonces

Escriba (N es común divisor de W y Z)
Sino
Escriba(N no es común divisor de W y Z)
Fin si
Fin

Código Turbo Pascal:

```
Program comun divisor;
Uses crt;
W,Z,N: integer;
Begin
   clrscr;  gotoxy(10,10);
   write("Digite un número N ");Readln(N);
   gotoxy(10,11);
   write("Digite un número W ");Readln(W);
   gotoxy(10,12);
   write("Digite un número Z ");Readln(Z);
   gotoxy(10,14);
   if (W mod N = 0) and (Z mod N=0) then
      write("N es común divisor de W y Z")
   Esle
      write("N no es común divisor de
      W y Z ");
   repeat until keypressed;
End
```

Código Lenguaje C

```
/*   Programa comun divisor  */
#include "conio.h"
#include "stdio.h"
int  W,Z,N;
main()
{   clrscr();  gotoxy(10,10);
    printf("Digite un número N ");scanf("%d",&N);
    gotoxy(10,11);
    printf("Digite un número W ");scanf("%d",&W);
    gotoxy(10,12);
    printf("Digite un número Z ");scanf("%d",&Z);
    gotoxy(10,14);
    if (W%N==0 && Z%N==0)
       printf("N es común divisor de W y Z");
    else
       printf("N no es común divisor de W y Z"');
    getch();
}
```

Para ejecutar este programa desde MatLab recuerde guardarlo bajo un nombre de archivo.m y llamarlo desde el prompt del MatLab.

```
N = input('Digite un número N ');
W = input('Digite un número W ');
Z = input('Digite un número Z ');
if (mod(W,N)==0) & (mod(Z,N)==0)
   fprintf('N es común divisor de W y Z \n');
else
   fprintf('N no es común divisor de W y Z \n');
end
```

5. Desarrolle un algoritmo que le permita leer un valor cualquiera N y escribir si dicho número es común múltiplo de M y P. M y P también se deben leer desde el teclado.

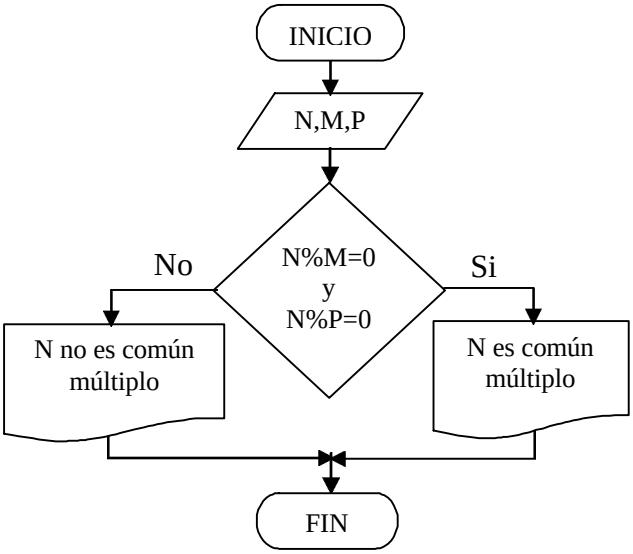
Análisis:

Para dar solución, primero se deben leer los valores. En N se almacena un valor y en las variables M y P se almacenarán los otros dos valores de los cuales se desea

saber si N es común múltiplo o no. Para poder saber si N es múltiplo habrá que realizar una división y preguntar si dicha división es exacta o no, con cada uno de los dos valores (N/M y N/P). Si cada división es exacta entonces escribir que N es común múltiplo de M y P o en caso contrario decir que N no es común múltiplo.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un numero (N) y otros dos almacenarlos en las variables M y P

Si la división entre (N,M) y (N,P) es exacta Entonces
    Escreba
    N es común múltiplo de M y P
Sino
    Escriba
    N no es común múltiplo de M y P

Fin

Seudo código

Inicio
    Leer(N,M,P)
    Si (N residuo M =0 y N residuo P =0) entonces
        Escriba(N es común múltiplo de M y P)
    Si no
        Escriba(N no es común múltiplo de M y P)
    Fin si
Fin

Código Turbo Pascal:

```
Program común_múltiplo_de_M_y_P;
Uses crt;
N,M,P: integer;
Begin
   clrscr;        gotoxy(10,10);
   write("Digite un número N");
   readln(N);   gotoxy(10,11);
   write("Digite un número M");
   readln(M);   gotoxy(10,12);
   write("Digite un número P");
   readln(P);   gotoxy(10,13);
   if (N mod M=0) and (N mod P=0)
   then
      write("N es múltiplo de M y P")
   Esle
      write("N no es múltiplo de M y P");
   repeat until keypressed;
```

Código Lenguaje C

```
/*  Programa común_múltiplo_de_M_y_P  */
#include "conio.h"
#include "stdio.h"
int  N,M,P;
main()
{  clrscr();      gotoxy(10,10);
   printf("Digite un número N ");
   scanf("%d",&N);    gotoxy(10,11);
   printf("Digite un número M ");
   scanf("%d",&M);    gotoxy(10,12);
   printf("Digite un número P ");
   scanf("%d",&P);    gotoxy(10,13);
   if (N%M==0 && N%P==0)
      printf("N es múltiplo de M y P");
   else
      printf("N no es múltiplo de M y P");
   getch();
```

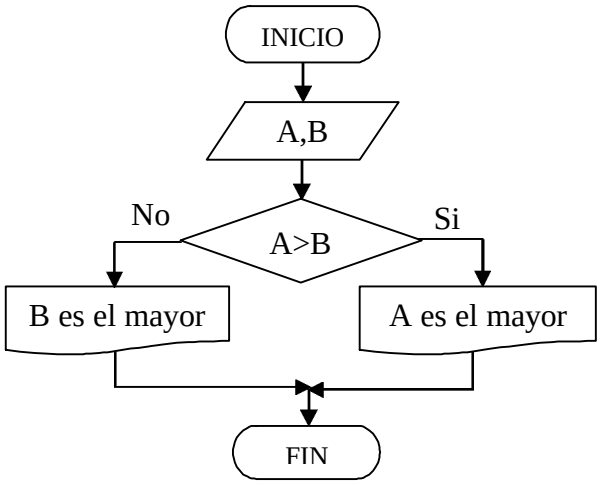
Para desarrollar los siguientes ejercicios se asume que los valores que se lean para las variables A y B son diferentes

6. Desarrolle un algoritmo que le permita leer dos valores (A y B) y que escriba cual de los dos valores leídos es el mayor

Análisis: Para dar solución al anterior ejercicio, primero se deben leer los dos valores y almacenar cada uno de ellos en una variable. Para el caso del desarrollo se almacenará un valor en la variable A y el otro en la variable B. Para poder saber cual de los dos valores es mayor simplemente se comparan las dos variables y se escribirá cual de las dos es la mayor. Se supone que los dos valores leídos son diferentes.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer dos valores y almacenarlos en las variables A y B

Si A>B

Entonces Escriba A es el mayor

Sino Escriba B es el mayor

Fin

Seudo código

```
Inicio
    Leer(A,B)
```

```
        Si (A>B) entonces   Escriba(A es el mayor)
        Sino                Escriba(B es el mayor)
        Fin Si
Fin
```

Código Turbo Pascal:

```
program cual_de_los_dos_es_mayor;
uses crt;
A,B: Integer;
begin
    clrscr;  gotoxy(10,10);
    write("Digite un número");
    readln(A);  gotoxy(10,11);
    write("Digite otro número");
    readln(B); gotoxy(10,12);
    if A>B then
       write("A es el mayor")
    else
       write("B es el mayor");
    repeat until keypressed;
end
```

Código Lenguaje C

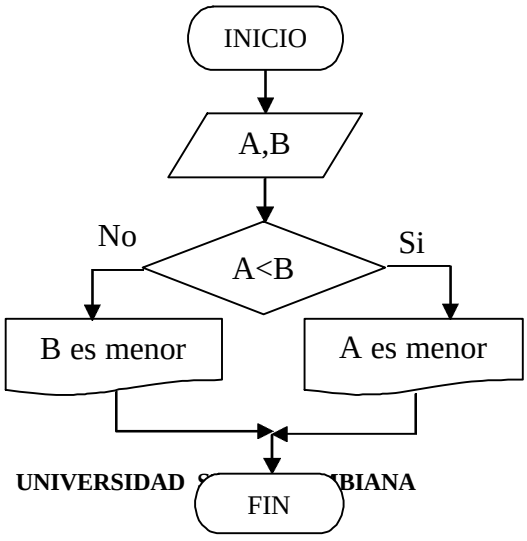
```
/*  programa cual_de_los_dos_es_mayor  */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{  clrscr();  gotoxy(10,10);
    printf("Digite un número A ");
    scanf("%d",&A);  gotoxy(10,11);
    printf("Digite un número B ");
    scanf("%d",&B);  gotoxy(10,12);
    if (A>B)
       printf("A es el mayor");
    else
       printf("B es el mayor");
    getch();
}
```

7. Desarrolle un algoritmo que le permita leer dos valores A y B y escriba cual de los dos valores leídos es el menor

Análisis: Este ejercicio es similar al anterior, primero se deben leer los dos valores y almacenar cada uno de ellos en una variable. Para el caso del desarrollo se almacenará un valor en la variable A y el otro en la variable B. Para poder saber cual de los dos valores es menor se comparan las dos variables y se escribirá cual de las dos es la menor. Se supone que los dos valores leídos son diferentes.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Lea dos valores y almacénelos en las variables A y B.

Si al evaluar la condición A<B es verdadero
Entonces Escriba A es el menor
Sino escriba B es el menor

Fin

Seudo código

```
Inicio
        Leer(A,B)
        Si (A<B) entonces
                Escriba(A es el menor)
        Sino
                Escriba(B es el menor)
        Fin Si
Fin
```

Código Turbo Pascal:

```
program cual_de_los_dos_es_menor;
Uses crt;
A,B: Integer;
begin
    clrscr;  gotoxy(10,10);
    write("Digite un número A ");

    readln(A); gotoxy(10,11);
    write("Digite un número B ");
    readln(B);  gotoxy(10,12);
    if A<B then
       write("A es el menor")
    else
       write("B es el menor");
    repeat until keypressed;
end
```

Código Lenguaje C

```
/*   Programa cual_de_los_dos_es_menor   */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{   clrscr();  gotoxy(10,10);
    printf("Digite un numero A ");
    scanf("%d",&A);  gotoxy(10,11);
    printf("Digite un numero B ");
    scanf("%d",&B);  gotoxy(10,12);
    if (A<B)
        printf("A es el menor");
    else
        printf("B es el menor");
    getch();
}
```

Digitado en Matlab quedará así:

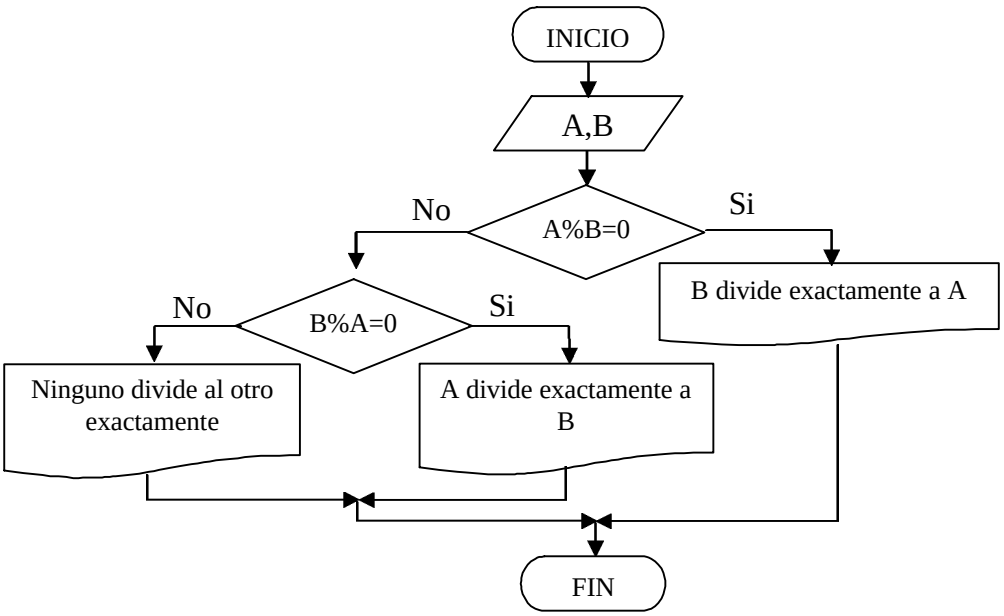
```
A = input('Digite un número A ');
B = input('Digite un número B ');
if (A>B)
   fprintf('A es mayor\n');
else
   fprintf('B es mayor\n');
end
```

8. Desarrolle un algoritmo que le permita leer 2 valores A y B e indicar si uno de los dos divide al otro exactamente

Análisis: Para dar solución al anterior ejercicio, primero se deben leer los dos valores y almacenar cada uno de ellos en una variable. Para el caso del desarrollo se almacenará un valor en la variable A y el otro en la variable B. Para saber si uno de los dos divide exactamente al otro se examina primero el caso en que B divida exactamente a A; se compara el residuo, si es cero se escribirá que divide exactamente a A de lo contrario se examina el caso en que A divida exactamente a B. Se compara nuevamente el residuo, si es cero se escribirá "A divide exactamente a B" sino "ninguno de los dos divide exactamente al otro". Se supone que los dos valores leídos son diferentes.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

```
Inicio
Leer dos valores. El uno almacenarlo en la variable A y el otro en la variable B
Si la división entre A y B es exacta Entonces
        Escribir que B divide exactamente a A
Sino
        Si la división entre B y A es exacta  Entonces
                Escribir que A divide exactamente a B
        Si_no
                Escribir que no se dividen exactamente.
```

```
        Fin_si
Fin_si
Fin
```

Seudo código

```
Inicio
  Leer (A,B)
  Si A residuo B=0 entonces
        Escriba (B divide exactamente a A)
  Si_no
        Si B residuo A=0 entonces
              Escriba (A divide exactamente a B)
        Si_no
              Escriba (No se dividen exactamente)
        Fin_si
  Fin_si
Fin
```

Código Turbo Pascal:

```
program división_exacta;
Uses crt;
A,B: integer;
begin
    clrscr;  gotoxy(10,10);
    write("Digite un número");
    readln(A);   gotoxy(10,11);
    write("Digite otro número");
    readln(B);   gotoxy(10,12);
    if A mod B=0 then
      write("B divide exactamente a  A")
    else
    if B mod A=0 then
      write("A divide exactamente a  B")
    else
      write("Ninguno de los dos divide
    exactamente al otro");
    repeat until keypressed;
end
```

Código Lenguaje C

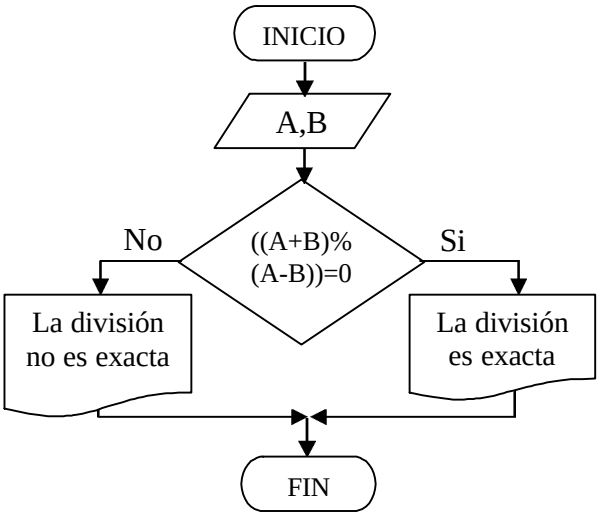
```
/*  Programa división exacta  */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{  clrscr();  gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);  gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);     gotoxy(10,12);
   if (A%B==0)
      printf("B divide exactamente a A");
   else
   if (B%A==0)
      printf("A divide exactamente a B");
   else
      printf("Ninguno de los dos divide
   exactamente al otro");
   getch();
}
```

9. Desarrolle un algoritmo que le permita leer dos valores A y B e indicar si el resultado de dividir la suma del los dos números entre la resta del primer número con el segundo es exacta.

Análisis: Primero se leen los dos valores almacenando cada uno de ellos en una variable, para el caso del ejercicio, el primer valor es almacenado en la variable A y el segundo valor es almacenado en la variable B. Para saber si la división es exactas se chequea el residuo. En caso de que el residuo sea igual a cero es porque la división fue exacta, de lo contrario la división no es exacta.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer dos valores. El uno almacenarlo en la variable A y el otro en la variable B
Si la división entre A+B y A-B es exacta
Entonces Escribir que La división es exacta
Sino
Escribir que no se dividen exactamente.
Fin_si
Fin

Seudo código

Inicio
Leer (A,B)
Si (A+B) residuo (A-B)=0 entonces
Escriba (La división es exacta)
Si_no
Escriba (La división es inexacta)

Fin_si
Fin

Código Turbo Pascal:

```
program división_exacta;
Uses crt;
A,B: integer;
begin
    clrscr;
    gotoxy(10,10);
    write("Digite un número");
    readln(A);
    gotoxy(10,11);
    write("Digite otro número");
    readln(B);
    gotoxy(10,12);
    if (A+B) mod (A-B)=0 then
       write("La división es exacta")
    else
       write("La división es inexacta")
```

Código Lenguaje C

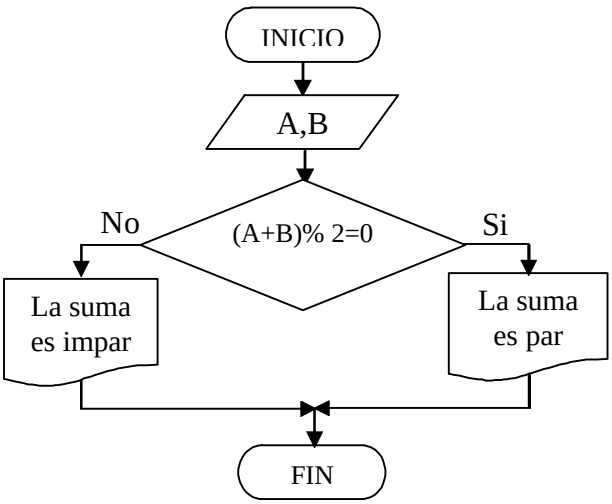
```
/*  Programa división exacta  */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{  clrscr();
   gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);
   gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);
   gotoxy(10,12);
   if ((A+B)%(A-B) == 0)
      printf("La división es exacta");
```

10. Desarrolle un algoritmo que le permita leer dos valores A y B e indicar si la suma de los dos números es par

Análisis: Primero se leen los dos datos almacenando cada uno de ellos en un variable, en el caso del ejercicio el primer valor se almacena en la variable A y el segundo se almacena en la variable B. Al sumarlos para saber si el resultado es par o impar se divide entre dos chequeando el residuo. Si el residuo es cero es porque el valor es par y si el residuo es uno es porque el valor es impar.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer dos valores y almacenarlos en las variables A y B

Si el residuo de (A+B)/2 es igual a =0 entonces
    Escribir La suma es par
Sino
    Escribir La suma es impar
Fin Si

Fin

Seudo código

```
Inicio
Leer (A y B)
Si (A+B)%2=0 entonces
        Escribir(La suma es par)
Sino
```

Escribir(La suma es impar)
Fin Si
Fin

Código Turbo Pascal:

```
program suma_par;
Uses crt;
A,B: integer;
begin

    clrscr;        gotoxy(10,10);
    write("Digite un número");
    readln(A);   gotoxy(10,11);
    write("Digite otro número");
    readln(B);   gotoxy(10,12);
    if ((A+B)mod 2=0 then
       write("La suma es par")
    else
       write("La suma es impar");
    repeat until keypressed;
end
```

Código Lenguaje C

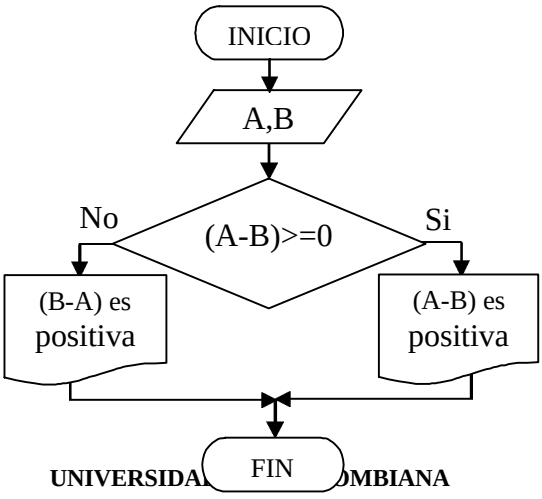
```
/*  Programa suma par  */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{  clrscr();      gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);     gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);     gotoxy(10,12);
   if ((A+B)%2== 0)
      printf("La suma es par");
   else
      printf("La suma es impar");
   getch();
}
```

11. Desarrolle un algoritmo que le permita leer dos valores A y B e indicar cual de las dos restas (B-A) o (A-B) es positiva

Análisis: Leídos los dos valores,y almacenados los valores en las variables respectivas A y B, se procede a comparar una de las dos restas, si dicha resta es positiva se escribe el mensaje o si no se escribe lo contrario.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio
Leer dos valores y almacenarlos en l variables A y B
Si la resta A menos B es mayor o igual a 0 entonces
Escribir la resta de A menos B es positiva
Sino
Escribir La resta de B menos A es positiva
Fin Si
Fin

Seudo código

```
Inicio
Leer (A y B)
Si A-B>=0 entonces
        Escribir(A - B es positiva)
Sino
        Escribir(B - A es positiva)
Fin Si
Fin
```

Código Turbo Pascal:

```
program resta_positiva;
Uses crt;
A,B: integer;
begin
   clrscr;        gotoxy(10,10);
   write("Digite un número A ");
   readln(A);   gotoxy(10,11);
   write("Digite un número B ");
   readln(B);   gotoxy(10,12);
   if (A  - B)>=0 then
      write("A - B es positiva")
   else
      write("B - A es positiva");
   repeat until keypressed;
end
```

Código Lenguaje C

```
/*   Programa resta positiva   */
#include "conio.h"
#include "stdio.h"
int  A,B;
main()
{  clrscr();      gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);     gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);     gotoxy(10,12);
   if (A-B>= 0)
      printf("A - B es positiva");
   else
      printf("B - A es positiva");
   getch();      }
```

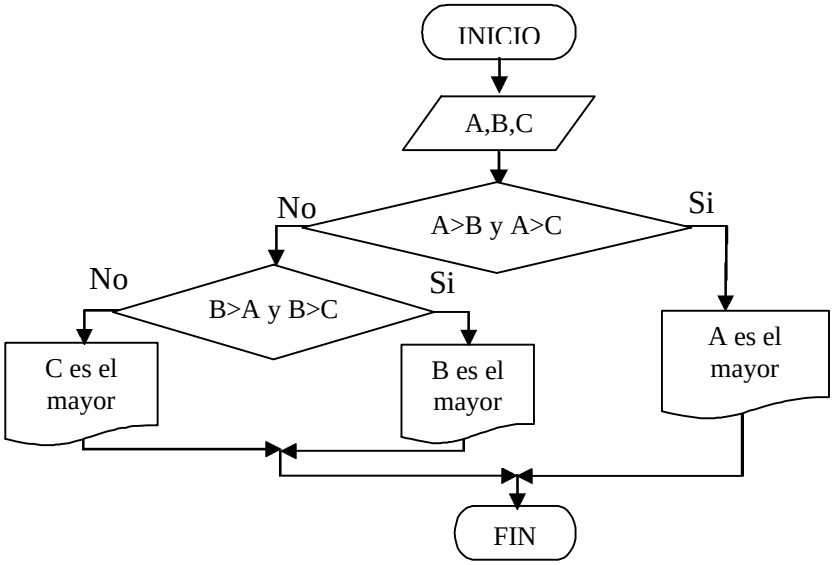
Para los siguientes algoritmos, los valores A, B y C deben ser ingresados diferentes.

12. Desarrolle un algoritmo que le permita leer tres valores y almacenarlos en las variables A,B,C respectivamente. El algoritmo debe indicar cual es el mayor. Para este caso se asume que los tres valores leídos por el teclado son valores distintos/

Análisis: Es necesario leer los tres valores a comparar, cada uno de ellos se almacena en una variable que para el ejercicio será A, B y C. Para saber si A es el valor mayor se compara con las variables B y C respectivamente. En caso de ser mayor se escribe el mensaje, en caso contrario se sigue verificando otra variable caso B y si no por

defecto se dirá que C es el mayor asumiendo que los tres valores almacenados son diferentes.

Algoritmo:
Diagrama de flujo



Seudo lenguaje

Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si A es mayor que B y A es mayor que C entonces
    Escribir A es el mayor
Sino
    Si B es mayor que A y B es mayor que C entonces
        Escribir B es el mayor
    Sino
        Escribir C es el mayor
    Fin Si
Fin Si
Fin

Seudo código

Inicio
Leer (A, B y C)
Si A>B y A>C entonces
    Escribir(A es el mayor)
Sino
    Si B>A y B>C entonces
        Escribir (B es el mayor)
    Sino
        Escribir (C es el mayor)
    Fin Si
Fin Si
Fin

Código Turbo Pascal:

```
program mayor;
Uses crt;
A,B,C: integer;
begin
    clrscr; gotoxy(10,10);
    write("Digite un número A")
    ;readln(A);gotoxy(10,11);
    write("Digite un número B")
    ;readln(B);gotoxy(10,12);
    write("Digite un número C");
    readln(C);gotoxy(10,13);
    if (A > B) and (A>C) then
       write("A es el mayor")
    else  if ( B>A) and (B>C) then
          write("B es el mayor")
    else
       write("C es el mayor");
    repeat until keypressed;
end.
```

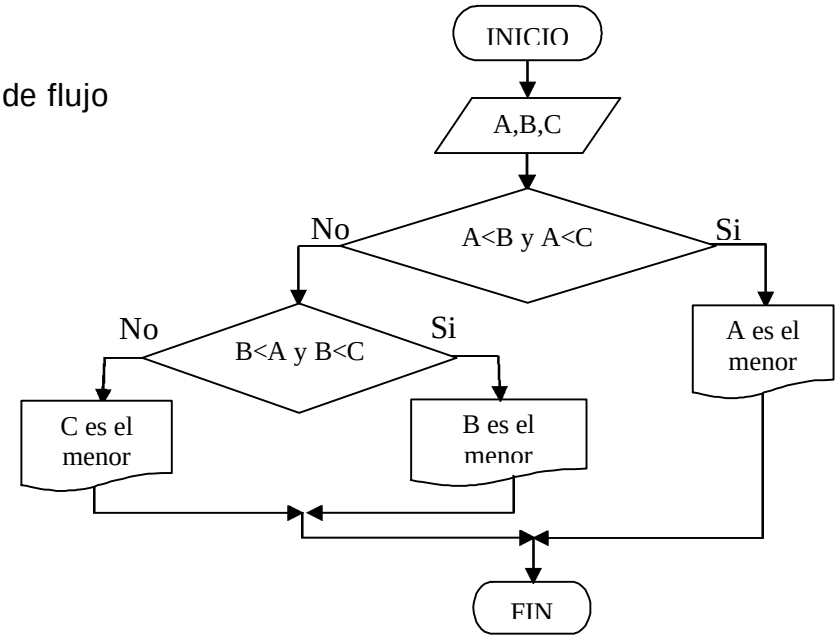
Código Lenguaje C

```
/*   Programa mayor   */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
{   clrscr();gotoxy(10,10);
    printf("Digite un número A ");
    scanf("%d",&A);gotoxy(10,11);
    printf("Digite un número B ");
    scanf("%d",&B);gotoxy(10,12)
    printf("Digite un número C ");
    scanf("%d",&C);gotoxy(10,13);
    if (A>B && A>C)
        printf("A es el mayor");
    else if (B>A && B>C)
        printf("B es el mayor");
    else
        printf("C es el mayor");
    getch(); }
```

13. Desarrolle un algoritmo que le permita leer tres valores A,B,C e indicar cual es el menor

Análisis: Es similar al ejercicio anterior con diferencia en la aserción realizada. En este se verificará si una variable contiene el menor valor con respecto a las otras dos. Igualmente se asume que los valores leídos son diferentes.

Algoritmo:
Diagrama de flujo



Código Turbo Pascal:

```
program menor;
Uses crt;
A,B,C: integer;
begin
   clrscr;
   gotoxy(10,10);
   write("Digite un número");
   readln(A);
   gotoxy(10,11);
   write("Digite otro número");
   readln(B);
   gotoxy(10,12);
   write("Digite otro número");
   readln(C);
   gotoxy(10,13);
   if (A< B) and (A<C) then
      write("A es el menor")
   else
      if ( B<A) and (B<C) then
         write("B es el menor")
   else
   write("C es el menor");
   repeat until keypressed;
end
```

Código Lenguaje C

```
/*  Programa menor  */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
{  clrscr();
   gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);
   gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);
   gotoxy(10,12);
   printf("Digite un número C ");
   scanf("%d",&C);
   gotoxy(10,13);
   if (A<B && A<C)
      printf("A es el menor");
   else
   if (B<A && B<C)
      printf("B es el menor");
   else
      printf("C es el menor");
   getch();
}
```

Seudo Lenguaje
Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si A es menor que B y A es menor que C entonces
    Escribir A es el menor
Sino
    Si B es menor que A y B es menor que C entonces
        Escribir B es el menor
    Sino
        Escribir C es el menor
    Fin Si
Fin Si
Fin

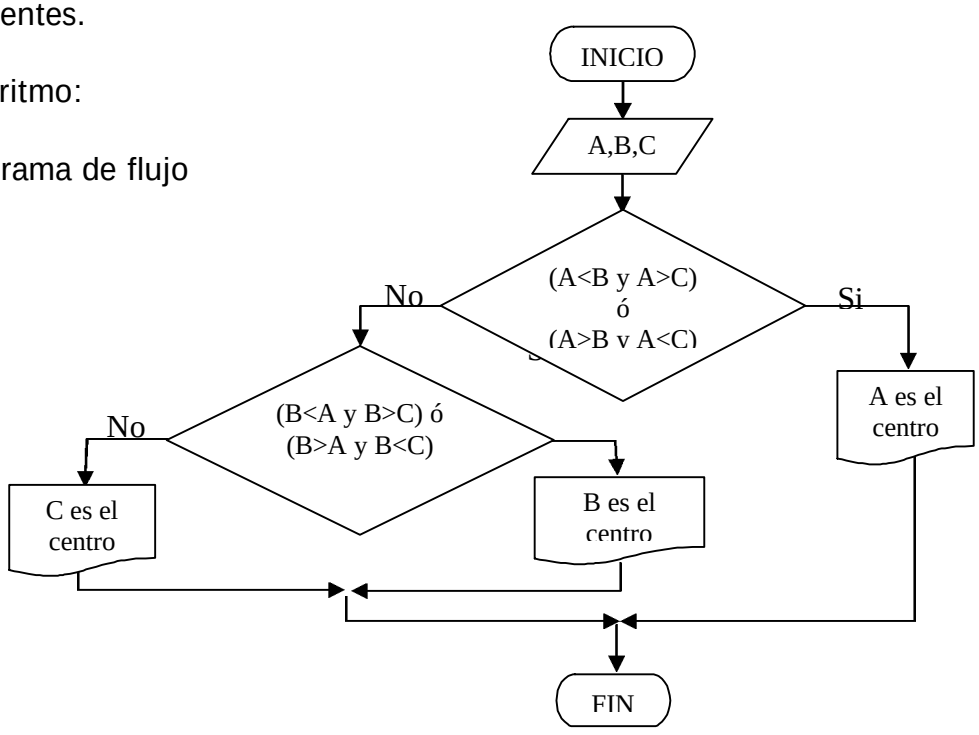
Seudo Código
Inicio
Leer (A, B y C)
Si A<B y A<C entonces
    Escribir(A es el menor)
Sino
    Si B<A y B<C entonces
        Escribir (B es el menor)
    Sino
        Escribir (C es el menor)
    Fin Si
Fin Si
Fin

14. Desarrolle un algoritmo que le permita leer tres valores A,B,C e indique cual es valor del centro

Análisis: Una vez leídos los valores en cada uno de los indicadores (variables A,B,C) se procede a comparar cada uno de ellos con los otros dos para verificar si es valor del centro o no. Un valor es del centro si es menor que uno y mayor que otro o el caso contrario. Igualmente se asume que los tres valores leídos son diferentes.

Algoritmo:

Diagrama de flujo



Seudo lenguaje
Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si A es menor que B y A es mayor que C ó
A es mayor que B y A es menor que C entonces
Escribir A es el valor del centro
Sino
Si B es menor que A y B es mayor que C ó
B es mayor que A y B es menor que C entonces
Escribir B es el valor del centro
Sino
Escribir C es el valor del centro
Fin Si
Fin Si Fin

Seudo código
Inicio
Leer (A, B y C)
Si A<B y A>C o A>B y A<C entonces
Escribir(A es el centro)
Sino
Si B<A y B>C ó B>A y B<C entonces
Escribir (B es el centro)
Sino
Escribir (C es el centro)
Fin Si
Fin Si Fin

Código Turbo Pascal:

```
program valor_centro;
Uses crt;
A,B,C: integer;
begin
   clrscr;  gotoxy(10,10);
   write("Digite un número A ");
   readln(A); gotoxy(10,11);
   write("Digite un número B ");
   readln(B); gotoxy(10,12);
   write("Digite un número C ");
   readln(C); gotoxy(10,13);
   if (A< B)and(A>C) or (A>B)and(A<C)
   then   write("A es el centro")
   else
      if (B< A) and (B>C) or (B>A) and (B<C)
   then   write("B es el centro")
   else   write("C es el centro");
   repeat until keypressed;
end
```

Código Lenguaje C

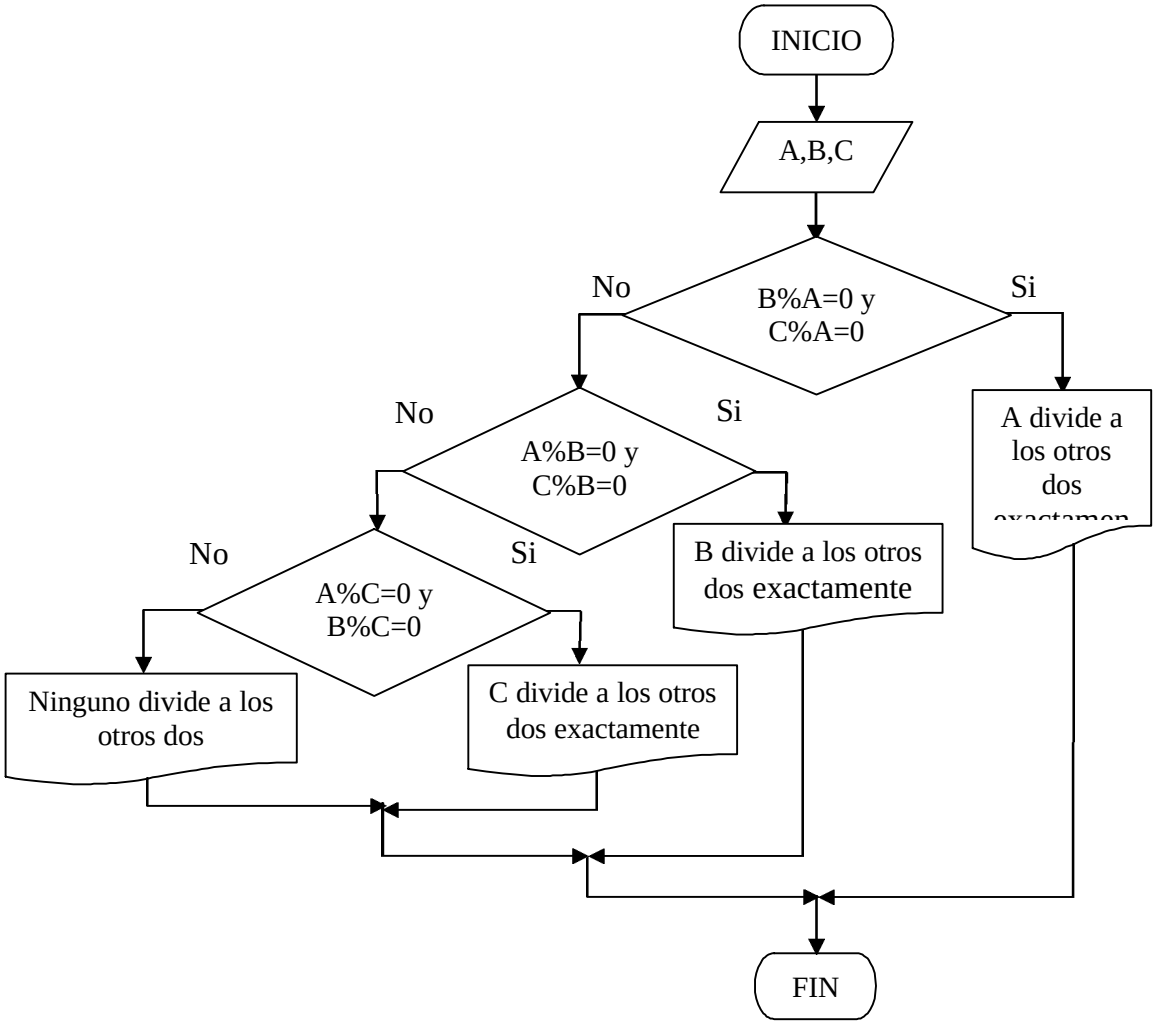
```
/*   Programa valor centro   */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
{   clrscr(); gotoxy(10,10);
    printf("Digite un número A ");
    scanf("%d",&A); gotoxy(10,11);
    printf("Digite un número B ");
    scanf("%d",&B); gotoxy(10,12);
    printf("Digite un número C ");
    scanf("%d",&C); gotoxy(10,13);
    if ((A<B && A>C) | (A>B && A<C))
        printf("A es el centro");
    else if ((B<A && B>C) | (B>A && B<C))
        printf("B es el centro");
    else
        printf("C es el centro");
    getch(); }
```

15. Desarrolle un algoritmo que le permita leer tres valores A,B,C e indicar si uno de los tres divide a los otros dos exactamente

Análisis: Leídos los tres valores y almacenados en cada una de las varaibles A,B y C respectivamente se procede a verificar si cada uno de ellos divide exactamente a los otros dos. La división es exacta si el residuo de la división respectiva es igual a cero.

Algoritmo:

Diagrama de flujo



Tanto el seudo lenguaje como el seudo código deben quedar de acuerdo como se haya realizado el diagrama de flujo. Es importante para que al realizar la prueba de escritorio nos de lo que diagramamos.

Seudo lenguaje

```
Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si B residuo A es igual a cero y C residuo A es igual a cero
Entonces
    A divide exactamente a B y C
Sino
    Si A residuo B es igual a cero y C residuo B es igual a cero
        Escribir B divide exactamente a A y C
    Sino
        Si A residuo C es igual a cero y B residuo C es igual a cero
          Escribir C divide exactamente a A y B
        Sino
          Escribir no hay división exacta
  Fin _Si
Fin_Si
Fin
```

Seudo código

```
Inicio
Leer (A, B y C)
Si B%A=0 y C%A=0 entonces
        Escribir(A divide exactamente a B y C)
Sino
        Si A%B=0 y C%B=0 entonces
              Escribir (B divide exactamente a A y C)
        Sino
             Si A%C=0 y B%C=0 entonces
                    Escribir (C divide exactamente a B y A)
             Sino
                    Escribir (No hay división exacta)
             Fin Si
        Fin Si
Fin Si
Fin
```

Código Turbo Pascal:

```
program division_exacta;
Uses crt;
A,B,C: integer;
begin
    clrscr;
    gotoxy(10,10);
    write("Digite un número");
    readln(A);
    gotoxy(10,11);
    write("Digite otro número");
    readln(B);
    gotoxy(10,12);
    write("Digite otro número");
    readln(C);
    gotoxy(10,13);
    if (B mod A=0)and(C mod A=0) then
       write("A divide exactamente a C y B")
    else
       if (A mod B=0) and (C mod B=0) then
          write("B divide exactamente a A y C")
       else
          if (A mod C=0) and (B mod C=0) then
             write("C divide exactamente a A y B");
          else
             write("no hay división exacta");
    repeat until keypressed;
end
```

Código Lenguaje C

```
/*  Programa división exacta  */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
 { clrscr();
   gotoxy(10,10);
   printf("Digite un número A ");
   scanf("%d",&A);
   gotoxy(10,11);
   printf("Digite un número B ");
   scanf("%d",&B);
   gotoxy(10,12);
   printf("Digite un número C ");
   scanf("%d",&C);
   gotoxy(10,13);
   if (B%A==0 && C%A==0)
       printf("A divide exactamente a C y B");
   else
   if (A%B==0 && C%B==0)
       printf("B divide exactamente a A y C");
   else
       if (A%C==0 && B%C==0)
           printf("C divide exactamente a A y B");
       else
           printf("no hay división exacta");
   getch();
 }
```

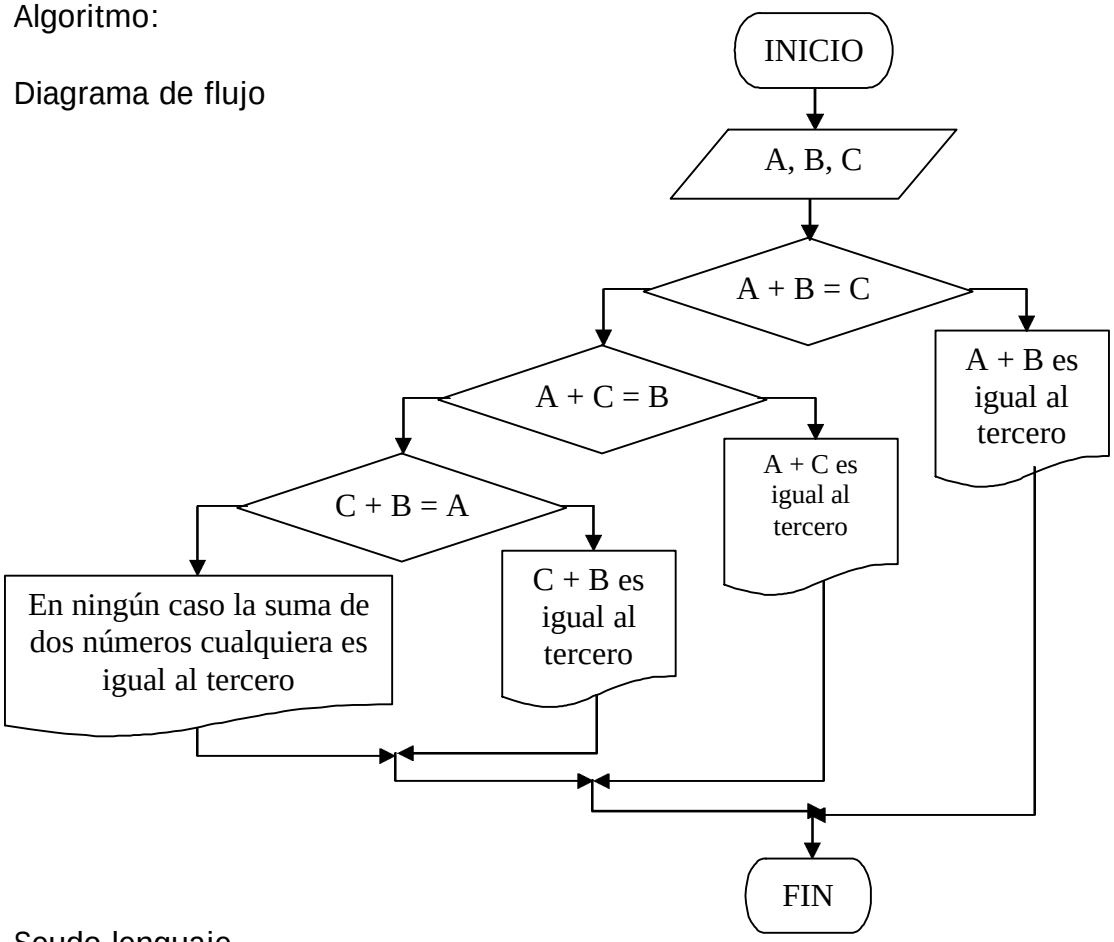
Nota: En el ejercicio anterior se hace uso de preguntas que involucran tanto a operadores tanto de relación como operadores lógicos. Esto es importante ya que la solución no queda tan compleja y no se queda ninguna opción sin contemplar.

16. Desarrolle un algoritmo que le permita leer tres valores A,B,C e indicar si la suma de dos números cualquiera es igual al tercero.

Análisis: Primero se deben leer los tres valores y almacenar cada valor en una variable. En el caso del ejemplo se guardaran los valores en los identificadores A,B, y C. luego se procederá a realizar las diferentes comparaciones.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

```
Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si A mas B es igual C entonces
Entonces
   Escribir La suma de A con  B es igual a C
Sino
   Si A+C es igual B entonces
        Escribir la suma de A con C es igual a B
   Sino
        Si C mas B es igual A
            Escribir la suma de C con B es igual a A
        Sino
           Escribir En ningún caso la suma es igual
   Fin _Si
Fin_Si
Fin
```

Seudo código

```
Inicio
Leer (A, B y C)
Si A+B=C entonces
        Escribir(la suma de A+B=C)
Sino
        Si A+C=B entonces
             Escribir (La suma de A+C=B)
        Sino
             Si C+B=A entonces
                  Escribir (La suma de C+B=A)
             Sino
                  Escribir (En ningún caso la suma es igual)
             Fin Si
        Fin Si
Fin Si
Fin
```

Código Turbo Pascal:

```
program valor_centro;
Uses crt;
A,B,C: integer;
begin
    clrscr;
    gotoxy(10,10);
    write("Digite un número A ");
    readln(A);
    gotoxy(10,11);
    write("Digite un número B ");
    readln(B);
    gotoxy(10,12);
    write("Digite un número C ");
    readln(C);
    gotoxy(10,13);
    if (A+B=C) then
       write("A + B = C")
    else
       if (A +C= B) then
          write("A + C = B")
       else
          if (C+B=A) then
             write("C + B = A")
          Else
             write("en ningun caso hay suma igual")
    repeat until keypressed;
end
```

Código Lenguaje C

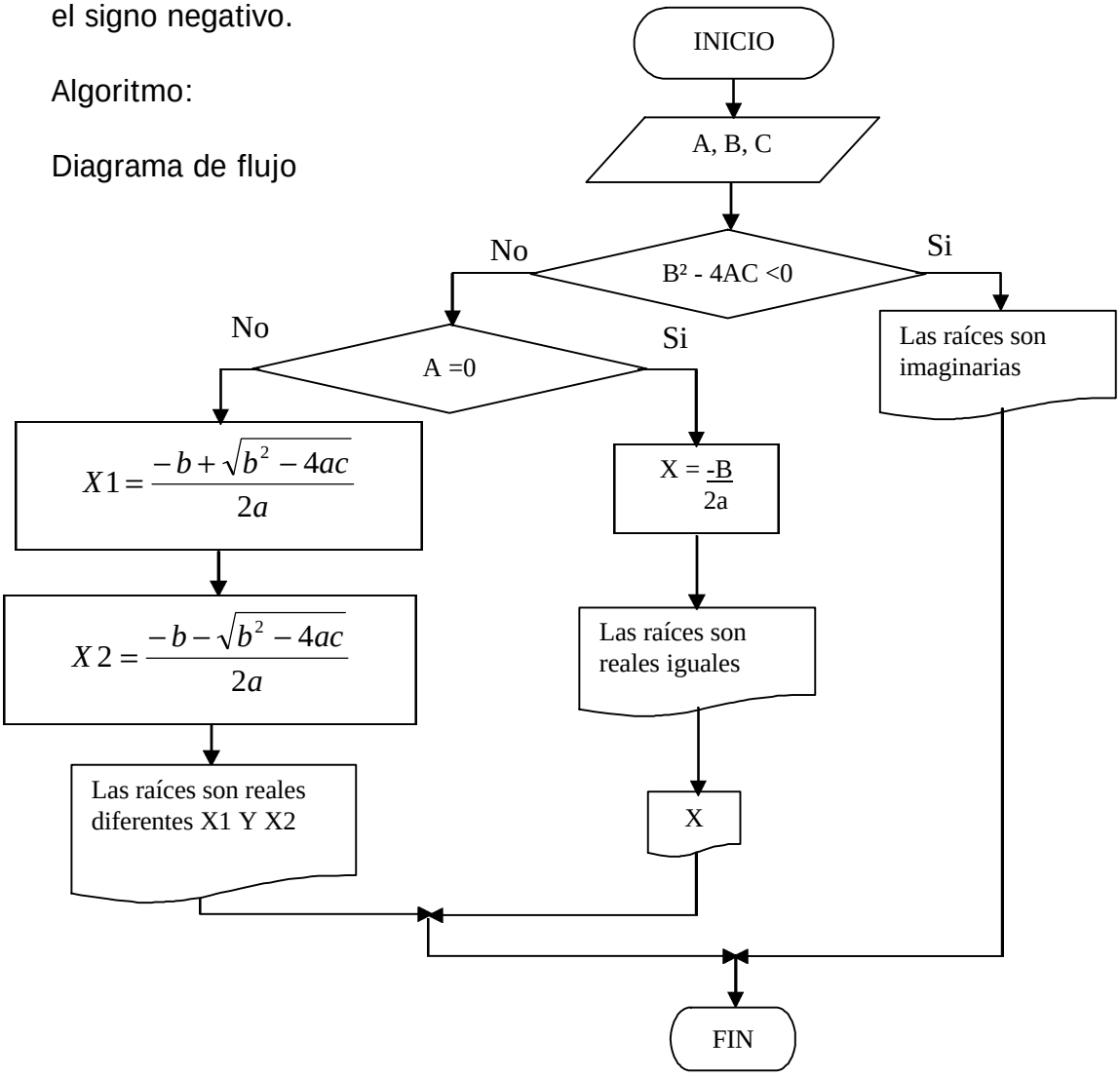
```
/*   Programa valor centro   */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un número A ");
    scanf("%d",&A);
    gotoxy(10,11);
    printf("Digite un número B ");
    scanf("%d",&B);
    gotoxy(10,12);
    printf("Digite un número C ");
    scanf("%d",&C);
    gotoxy(10,13);
    if (A+B==C)
        printf("La suma  de A+B=C");
    else
    if (A+C==B)
        printf("La suma de A+B=C");
    else
        if (C+B==A)
            printf("La suma de C+B= A");
        else
            printf("En ningun caso hay suma igual");
    getch();
}
```

17. Si se tiene la función: $f(x) = ax^2 + bx + c$, el valor de x se calcula así: $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$, la expresión dentro del radical se le llama discriminante de la ecuación. Las dos raíces son reales y desiguales, reales e iguales o imaginarias, según que el discriminante sea positivo, cero o negativo. Desarrolle un algoritmo que lea valores para a,b,c y determine si hay posible solución para x de dicha función.

Análisis: Leídos los tres valores en las variable a,b y c respectivamente se procede a realizar los chequeos tendientes a ver si las operaciones implícitas en el ejercicio se pueden o no realizar. Por ejemplo hay que verificar que la parte que está dentro del radical no sea negativa y que el valor de divisor "2a" no sea igual a cero. Si verificada esas dos acciones se pueden realizar las operaciones se procede a sacar cada uno de los valore de X, uno con el signo positivo y otro con el signo negativo.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio
Leer tres valores y almacenarlos en las variables A, B y C
Si B al cuadrado - 4 veces A por C es menor que cero entonces
    Escribir Las raíces son imaginarias
Sino
    Si B al cuadrado - 4 veces A por C es igual a cero entonces
        Hacer X igual a -B dividido 2 por a
        Escribir la raíces son reales e iguales a X
    Sino
        Hacer $X1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$
        Hacer $X2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$
        Escribir raíces reales iguales a X1 y x2
    Fin _Si
Fin_Si
Fin

Seudo código

Inicio
Leer (A, B y C)
Si $b^2 - 4ac < 0$ entonces
        Escribir(Raíces imaginarias)
Sino
        Si a=0 entonces
            $X = -b/2a$
            Escribir (Raíces reales iguales a X)
        Sino
            $X1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$
            $X2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$
            Escribir (Raíces reales X1 y X2)
        Fin Si
Fin Si
Fin

Código Turbo Pascal:

```
program valor_centro;
Uses crt;
A,B,C: integer;
begin
    clrscr; gotoxy(10,10);
    write("Digite un número A ");
    readln(A); gotoxy(10,11);
    write("Digite un número B ");
    readln(B); gotoxy(10,12);
    write("Digite un número C ");
    readln(C);
    gotoxy(10,13);
    if (b*b-4*a*c<0) then
       write("Raices imaginarias")
    else
       if (a=0) then
       begin
          X:=-b/(2*a)
          write("Raices reales iguales a X")
       end
       else
           begin
              X1:=(-b+(sqrt(b*b-4*a*c))/(2*a);
              X2:=(-b-(sqrt(b*b-4*a*c))/(2*a);
              write("raices reales igual a X1 y X2");
           end;
    repeat until keypressed;
end
```

Código Lenguaje C

```
/*   Programa valor centro   */
#include "conio.h"
#include "stdio.h"
int  A,B,C;
main()
{   clrscr(); gotoxy(10,10);
    printf("Digite un número A ");
    scanf("%d",&A); gotoxy(10,11);
    printf("Digite un número B ");
    scanf("%d",&B); gotoxy(10,12);
    printf("Digite un número C ");
    scanf("%d",&C);
    gotoxy(10,13);
    if (b*b-4*a*c<0)
        printf("Raices imaginarias");
    else
    if (a==0)
       {
         X=-b/(2*a);
         printf("Raices reales iguales a X");
       }
    else
       {
         X1=-b+sqrt(b*b-4*a*c)/(2*a);
         X2=-b-sqrt(b*b-4*a*c)/(2*a);
         printf("raices reales igual a X1 y X2");
       }
    getch();
}
```

El siguiente código soluciona el anterior ejercicio, con instrucciones para ser ejecutado desde el prompt del Matlab. Recuerde que tiene que guardarlo en un archivo tipo m y luego llamar el archivo desde el prompt colocando el nombre.

```
a = input('Digite un número A ');
b = input('Digite un número B ');
c = input('Digite un número C ');
if b^2-4*a*c<0
 fprintf('Raíces imaginarias\n');
else
if (a==0)
  x=-b/(2*a);
  fprintf('Raíces reales iguales a X = %f\m',x);
else
  x1=-b+sqrt(b*b-4*a*c)/(2*a);
  x2=-b-sqrt(b*b-4*a*c)/(2*a);
  fprintf('Raíces reales igual a X1 = %f y X2 = %f\n',x1,x2);
end
end
```

ESTRUCTURAS CÍCLICAS

Los procesos repetitivos son la base del uso de las computadoras. En estos procesos se necesita normalmente contar los sucesos, acciones o tareas internas del ciclo.

Una estructura cíclica o estructura repetitiva es aquella que le permite al programador repetir un conjunto o bloque de instrucciones un número determinado de veces mientras una condición dada sea cierta o hasta que una condición dada sea cierta.

Se debe establecer un mecanismo para terminar las tareas repetitivas. Dicho mecanismo es un control que se evalúa cada vez que se realiza un ciclo. La condición que sirve de control puede ser verificada antes o después de ejecutarse el conjunto de instrucciones o sentencias. En caso de que la verificación o evaluación resulte verdadera se repite el ciclo o caso de ser falsa lo terminará.

Ciclos con control antes

Las estructuras cíclicas cuyo control esta antes del ciclo, son estructuras que realizan la evaluación antes de ejecutar el bloque de instrucciones que tiene que repetir. Dependiendo de la evaluación que se realice se ejecutara o no dicho conjunto de instrucciones. Es posible que no se realice ni una sola vez ese conjunto de instrucciones. En el caso del MatLab solo trabaja con estructuras cuyo control está antes del ciclo (For y While)

Ciclos con control después:

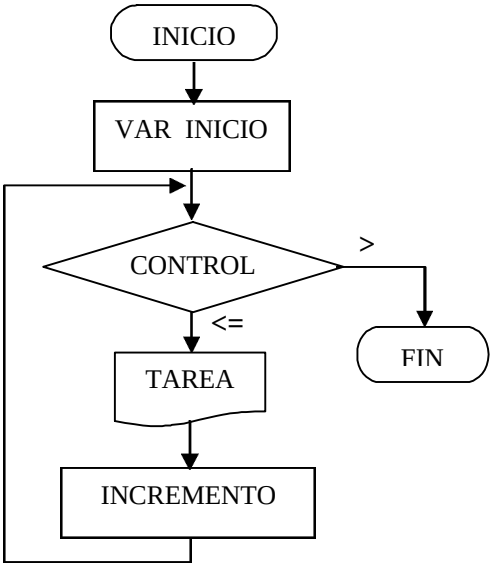
Las estructuras cíclicas cuyo control esta después del ciclo, son estructuras que realizan la evaluación después de ejecutar el bloque de instrucciones que se desea repetir. Con este tipo de control se obliga a la ejecución al menos de una vez del bloque de instrucciones.

El número de ciclos que se realiza puede ser definido previamente o no definido. Para el caso de que el ciclo sea definido en el número de veces que se repetirá, se debe trabajar un contador para que lleve la cuenta del número de tareas.

En caso de no tener definido el número de ciclos previamente, se tendrá que definir muy bien el control a fin de evitar que se quede en un número indefinido de ciclos y se bloquee la computadora (LOOP)

Estructuras cíclicas anidadas:

Una estructura cíclica puede estar anidada dentro de otra estructura cíclica o de decisión sin problemas. Hay que tener en cuenta que el anidamiento debe ser total.



Inicio

Inicialización de variable que cuenta las veces que se repetirá las tareas

Control de la variable inicio con el final del numero de tarea
Inicio

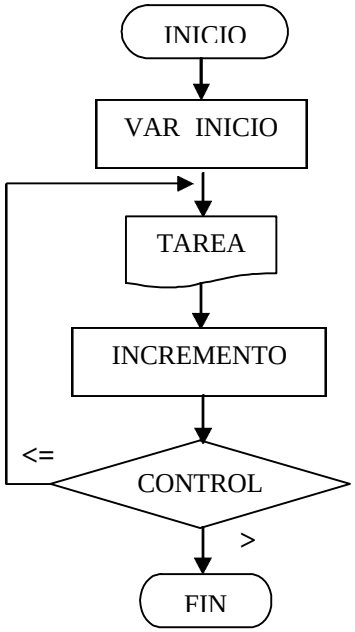
Tarea o tareas a realizar

Incremento de la variable que llevara el ciclo

Fin

FIN

Forma general de la estructura cíclica con control antes del ciclo



Inicio

Inicialización de variable que cuenta las veces que se repetirá las tareas

Hacer

Tarea o tareas a realizar

Incremento de la variable que llevara el ciclo

Control de la variable que inicia con el final del numero de tarea

FIN

Forma general de la estructura cíclica con control después del ciclo

Estructura cíclica FOR-ENDFOR

El enunciado FOR es una construcción de programación del C que ejecuta un bloque de uno o más enunciados una determinada cantidad de veces. A veces es llamado ciclo for, debido a que la ejecución del programa por lo general hace ciclos por los enunciados mas de una vez.

Si bien este bucle no se incluye en el Seudo código clásico, con ligeras variantes, es prácticamente adoptado por todos los lenguajes. Por ello, tratamos aquí este bloque con sus posibles modalidades y su Seudo código, no estándar, correspondiente.

Un enunciado FOR tiene la siguiente estructura:

```
for (inicial; condición; incremento)
        enunciado
```

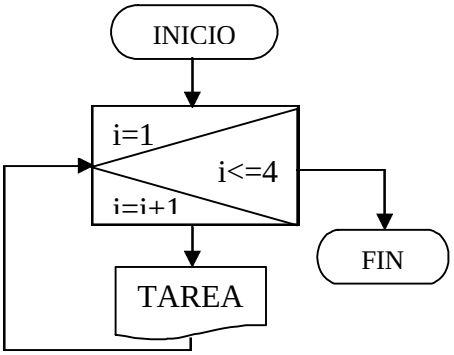
En el caso del MatLab el FOR esta definido así:

```
for (inicial: incremento: final)
   enunciado
end
```

Si no se define el incremento, el Matlab asume un valor de 1.

inicial, condición e incremento son expresiones del C, y enunciado es un enunciado simple compuesto del C. Cuando se encuentra un enunciado for durante la ejecución del programa sucede los siguientes eventos:

1. la expresión inicial es evaluada. Lo inicial es por lo general un enunciado de asignación que pone una variable a un valor determinado.
2. La expresión de condición es evaluada. La condición es típicamente una expresión relacional.
3. Si la condición evalúa a falso, el enunciado for termina y la ejecución pasa al primer enunciado que se encuentra a continuación del for.
4. Si la condición evalúa a cierto se ejecutan los enunciados que se encuentran dentro del for.
5. La expresión de incremento es evaluada y la evaluación regresa al paso dos.




```
/* Programa que escribe los cuatro primeros
números */

#include "conio.h"
#include "stdio.h"
int  i;
main()
{   clrscr();
    for(i=1; i<=4; i++)
      printf("%d ",i);
    getch();
}
```

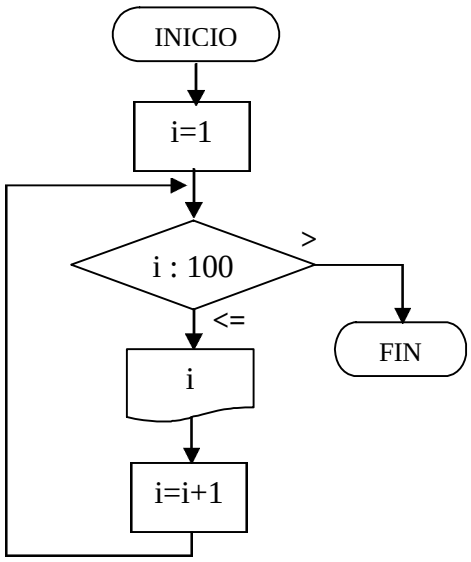
1. Desarrolle un algoritmo que le permita realizar la escritura de los primeros 100 números naturales.

Análisis: Para poder escribir los primeros 100 números primero hay que generar dichos valores. Una forma de generar los valores es con las estructuras cíclicas. Hacer una variable que se inicie en 1 que seria el primer valor a escribir y finalice en 100 que seria el último número necesitado incrementando de uno en uno dicha variable.

Inicia con I=1
Finaliza con I=100
Incrementa la I en uno o sea I=I+1
Dentro del ciclo se da la orden para escribir el valor de la variable I.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio
Asignar a la variable i el valor de 1

Mientras el valor de i sea menor o igual a 100
Haga

Inicie las instrucciones del ciclo

Escriba el valor de i

Incremente el valor de i en uno

Finalice las instrucciones del ciclo

Fin

Seudo código

```
Inicio
i=1
Mientras i<=100 haga
Inicio
    Escribir(i)
    i= i+1
Fin_Mientras
Fin
```

Código Turbo Pascal con el control antes del ciclo:

```
Program escribe_100_numeros;
Uses crt;
var
   I:integer;
Begin
    clrscr;
    i:=1;
    while i<=100 do
    begin
      write(i:3);
       i=i+1;
 end;
    repeat until keypressed;
End.
```

Código Turbo Pascal con el control después del ciclo:

```
Program escribe_100_numeros;
Uses crt;
Var
  I: integer;
Begin
    clrscr;
    i:=1;
    repeat
       write(i:3);
       i=i+1;
   until i>100;
    repeat until keypressed;
End.
```

/* Código Lenguaje C con Control antes del ciclo Escribe 100 numeros */

```
#include "conio.h"
#include "stdio.h"
int  i;
main()
{  clrscr();
   i=1;
   while (i<=100)
   {  printf("%d ",i);
      i++;
   }
   getch();  }
```

/* Código Lenguaje C con Control después del ciclo Escribe 100 numeros */

```
#include "conio.h"
#include "stdio.h"
int  i;
main()
{  clrscr(); i=1;
   do
   {  printf("%d ",i);
      i++;
   }
   while (i<=100);
   getch();  }
```

/* Código Lenguaje C con Estructura for Escribe 100 numeros */

```
#include "conio.h"
#include "stdio.h"
int  i;
main()
{  clrscr();
   for (i=1; i<=100 ; i++)
     printf("%d ",i);

   getch();
}
```

Para Matlab simplemente se define la variable ***y*** y se indica en el FOR donde inicia y donde termina, El incremento como no se define, el Matlab asume un incremento de 1.

```
for i=1:100
   fprintf('%d ',i);
end
```

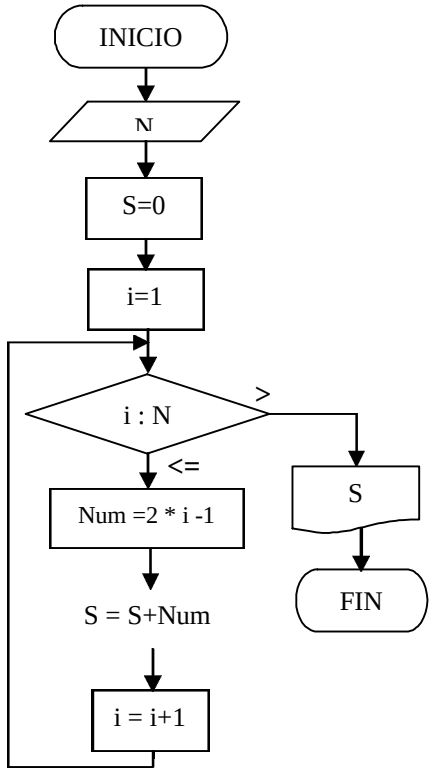
2. Desarrolle un algoritmo que le permita realizar la suma a los primeros N números impares.

Análisis: Al igual que en ejercicio anterior es necesario apoyarse en una estructura de tipo cíclica a fin de poder dar solución al problema. La idea es desarrollar la estructura para N veces y de la variable que lleve la cuenta generar los números impares buscando la relación entre la cuenta y el número como tal.

El primer término es el 1, el segundo el tres, el tercero el cinco y así sucesivamente hasta llegar al enésimo término que es el 2*N-1.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un numero y almacenarlo en la variable N

Asignar a la variable S el valor de 0

Asignar a la variable i el valor de 1

Mientras el valor de la variable i sea menor o igual al valor de la variable N haga

Inicio de las instrucciones del ciclo

Asigne a la variable Num el valor de 2 * i - 1
Asigne a la variable s el valor que hay
Almacenado en s mas el valor almacenado en la variable Num
Incremente el valor de la variable i en 1

Fin de las instrucciones del ciclo

Escriba el valor almacenado en S
Fin

Seudo código

```
Inicio
  Leer N;   S=0;   i=1
  Mientras i<=N hacer
  Inicio        Num=2i-1;  S=S+Num;   i= i+1  Fin_Mientras
  Escribir(S)
Fin
```

Código Turbo Pascal con el control antes del ciclo:

```
Program suma_N_impares;
Uses crt;
var
   num,s,N,I:integer;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    s:=0;
    i:=1;
    while i<=N do
    begin
      num:=2*i-1;
      s=s+num;
       i=i+1;
    end;
    gotoxy(10,20);
    write("La suma es ",s);
    repeat until keypressed;
End.
```

Código Turbo Pascal con el control después del ciclo:

```
Program suma_N_impares;
Uses crt;
Var
  num,s,N,I: integer;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    s:=0;
    i:=1;
    repeat
       num:=2*i-1;
       s:=s+num;
       i=i+1;
    until i>N;
    gotoxy(10,20);
    write("La suma es ",s);
    repeat until keypressed;
End.
```

Código Lenguaje C con Control antes del ciclo

```
/*  Programa suma N impares  */
#include "conio.h"
#include "stdio.h"
int  num,s,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    s=0;
    i=1;
    while (i<=N)
    {  num=2*i-1;
       s=s+num;
       i++;    }
    gotoxy(10,20);
    printf("La suma es %d",s);
    getch();
}
```

Código Lenguaje C con Control después del ciclo

```
/*  Programa suma N imapres */
#include "conio.h"
#include "stdio.h"
int  num,s,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    s=0;
    do
    {   num=2*i-1;
        s=s+num;
        i++;    }
    while (i<=N);
    gotoxy(10,20);
    printf("La suma es %d",s);
    getch();
}
```

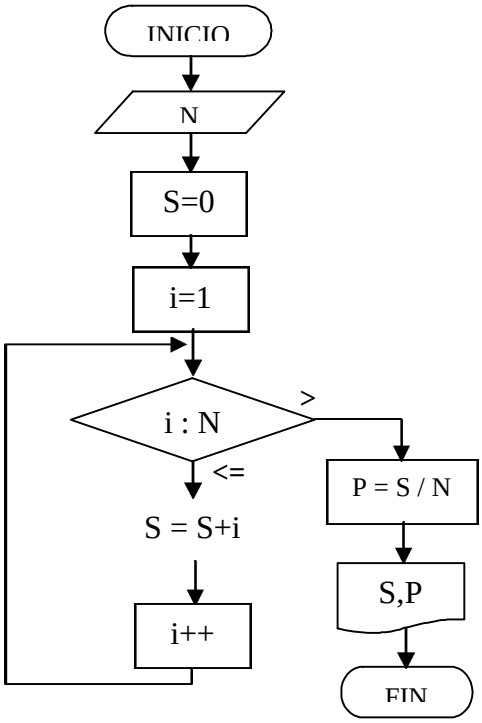
3. Desarrolle un algoritmo que calcule el promedio a los primeros N números naturales.

Análisis:

Para dar solución al ejercicio se procede de la siguiente forma: Se debe generar una estructura cíclica que se encargue de contar el numero de términos a sumar, en el caso del ejercicio será de N. La cuenta de los términos coincide con el valor a sumar, es decir la variable que se va a encargar de contar los términos sirve además como generadora de los términos a sumar. Se acumularán dichos números generados y al final se promediara dicha sumatoria entre el numero de términos sumados.

Algoritmo:

Diagrama de flujo



Seudo código

```
Inicio

Leer N

S=0

i=1


Mientras i<=N hacer
Inicio
   S=S+i
   i= i+1
Fin_Mientras

P=S/N
Escribir(S,P)
```

Código Turbo Pascal con el Control antes del ciclo:

```
Program promedio_de_N_números;
Uses crt;
var
   s,N,I : integer;

   p : real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    s:=0;
    i:=1;
    while i<=N do
    begin
      s:=s+i;
       i:=i+1;
   end;
   p:=s/N;
   gotoxy(10,20);
   write("El promedio es ",p);
    repeat until keypressed;
End.
```

Control Código Turbo Pascal con el después del ciclo:

```
Program promedio_de_N_números;
Uses crt;
Var
  s,N,I : integer;

  p : real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    s:=0;
    i:=1;
    repeat
      s:=s+num;
       i:=i+1;
   until i>N;
   p:=s/N;
   gotoxy(10,20);
   write("El promedio es ",p);
    repeat until keypressed;
End.
```

Código Lenguaje C con Control antes del ciclo

```
/*  Programa suma N impares  */

#include "conio.h"
#include "stdio.h"
int  s,N,i;
float p;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    s=0;
    i=1;
    while (i<=N)
    {  s=s+i;
       i++;
    }
    p=s/N;
    gotoxy(10,20);
    printf("El promedio es %f",p);
    getch();
}
```

Código Lenguaje C con Control después del ciclo

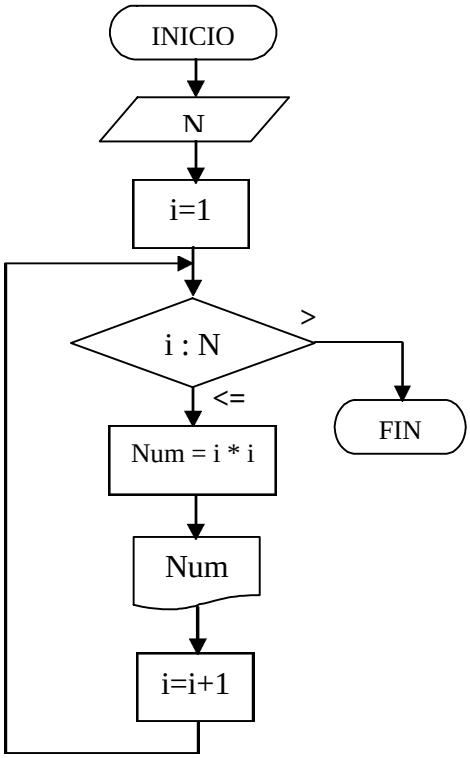
```
/*  Programa suma N imapres */

#include "conio.h"
#include "stdio.h"
int  s,N,i;
float p;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    s=0;
    do
    {  s=s+i;
       i++;
    }
    while (i<=N);
    p=s/N;
   gotoxy(10,20);
   printf("El promedio es %f",p);
    getch();
}
```

4. Desarrolle un algoritmo que le permita sacar y escribir el cuadrado de cada uno de los primeros N números naturales.

Análisis: Para dar solución al ejercicio se procede de la siguiente forma: Se debe generar una estructura cíclica que se encargue de generar cada uno de los términos a los cuales se les va a sacar cuadrado. La variable encargada de contar los términos en la estructura cíclica sirve como variable que guarda cada termino al cual se le saca el cuadrado.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en la variable N

Hacer i igual a 1

Mientras i sea menor o igual a N hacer

Inicio del ciclo
Hacer Num igual a i * i

Escribir el valor de Num

Hacer i igual a i mas 1

Fin del ciclo mientras
Fin

Seudo código

```
Inicio
Leer N;  i=1
Mientras i<=N hacer
Inicio
    Num= i * i;  Escribir(Num);  i= i+1
Fin_Mientras
Fin
```

Código Turbo Pascal con el control antes del ciclo:

```
Program cuadrado_de_N_numeros;
Uses crt;
var
    N,Num,I : integer;

 Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    i:=1;
    while i<=N do
    begin
      Num:=i*i;
      gotoxy(10,12+i);
      write("EL cuadrado es ",Num);
       i:=i+1;
    end;
    repeat until keypressed;
End.
```

Código Turbo Pascal con el control después del ciclo:

```
Program cuadrado_de_N_numeros;
Uses crt;
Var
  s,N,I : integer;

  p : real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    i:=1;
    repeat
      Num:=i*i;
      gotoxy(10,12+i);
      write("EL cuadrado es ",Num);
       i:=i+1;
    until i>N;
    repeat until keypressed;
End.
```

Código Lenguaje C con Control antes del ciclo

```
/*  Programa cuadrado de N numeros  */

#include "conio.h"
#include "stdio.h"
int  Num,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    while (i<=N)
    { Num=i*i;
      gotoxy(10,12+i);
      printf("EL cuadrado de %d es %d ",i,Num);
       i++;
    }
    getch();
}
```

Código Lenguaje C con Control después del ciclo

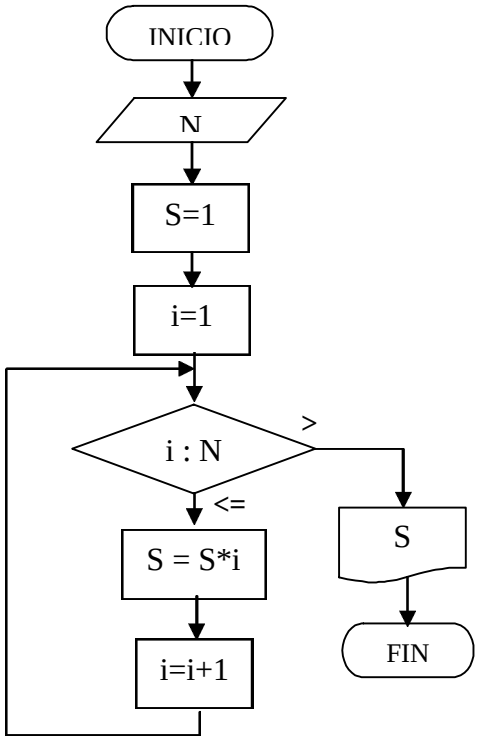
```
/*  Programa cuadrado de N Numeros */

#include "conio.h"
#include "stdio.h"
int  Num,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    do
    { Num=i*i;
      gotoxy(10,12+i);
      printf("EL cuadrado de %d es %d",i,Num);
       i++;
    }
    while (i<=N);
    getch();
}
```

5. Desarrolle un algoritmo que le permita leer un valor entero positivo N y calcule su factorial.

Análisis: El tipo de operación que se repite en este ejercicio es la multiplicación por tanto hay que iniciar una variable con el valor de 1 ya que este valor no afecta el resultado final. Dicha variable es S y como generador de la serie de términos a multiplicar se tiene la misma variable que llevara la cuenta del numero de tareas.

Algoritmo:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en la variable N.
A este valor se calculará su factorial
Asignar a la variable S el valor de 1

Asignar a la variable i el valor de 1

Mientras el valor de i sea menor o igual que el valor de N hacer
Inicio de instrucciones del ciclo mientras
Asignar a s el valor de S por i
Incrementar el valor de i en uno.
Finalizar las instrucciones del ciclo
Escribir el valor de S
Fin

Seudo código

```
Inicio
Leer N
S=1        i=1
Mientras i<=N hacer
Inicio
    S=S * i;  i= i+1
Fin
    Escribir(S)
Fin
```

Código Turbo Pascal con el control antes del ciclo:

```
Program Factorial_de_N;
Uses crt;
var
    N,S,i : integer;

 Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    i:=1;S:=1;
    while i<=N do
    begin
      S:=S*i;
      i:=i+1;
    end;
    gotoxy(10,12);
    write("EL factorial es ",S);
    repeat until keypressed;
End.
```

Código Turbo Pascal con el control después del ciclo:

```
Program Factorial_de_N;
Uses crt;
Var
  S,N,I : integer;

  p : real;
Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    i:=1; S:=1;
    repeat
      S:=S*i;
       i:=i+1;
    until i>N;
    gotoxy(10,12);
    write("EL factorial es ",S);
    repeat until keypressed;
End.
```

Código Lenguaje C con Control antes del ciclo

```
/*  Programa factorial de N  */

#include "conio.h"
#include "stdio.h"
int  S,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    S=1;
    while (i<=N)
    { S=S*i;
      i++;
    }
    gotoxy(10,12);
    printf("EL factorial es %d ",S);
    getch();
}
```

Código Lenguaje C con Control después del ciclo

```
/*  Programa Factorial de N */

#include "conio.h"
#include "stdio.h"
int  S,N,i;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor ");
    scanf("%d",&N);
    i=1;
    S=1;
    do
    { S=S*i;
      i++;
    }
    while (i<=N);
    gotoxy(10,12+i);
    printf("EL factorial es %d",S);
    getch();
}
```

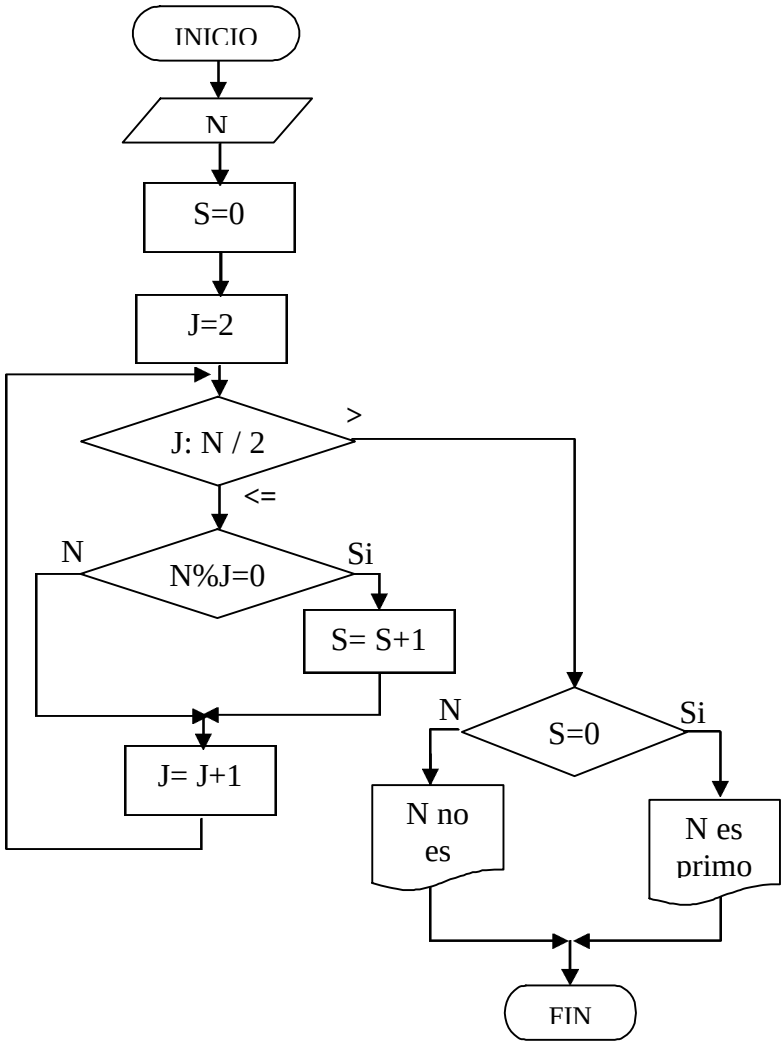
6. Desarrolle un algoritmo que le permita leer un valor entero positivo N y decir si es primo o no.

Análisis:

Un número es primo cuando es divisible tan solo por la unidad y por si mismo. Para determinar si un número es primo o no se realiza la verificación de la división de dicho número con el rango de datos comprendidos entre el dos y la mitad del número. Si existe algún valor de dicho rango que divide exactamente a nuestro número entonces este no será primo. Si al finalizar dicha revisión no hay ningún valor que lo divida exactamente entonces nuestro número será primo. La revisión se hace hasta la mitad del número ya que de la mitad hacia arriba ningún valor lo divide exactamente.

Algoritmo:

Diagrama de flujo



Seudo código

```
Inicio

Leer N

S=0

J=2

Mientras J<=N%2 hacer
Inicio

   Si N / J=0
      S=S+1

   J=J+1
Fin

Si S=0
   Escribir(N es primo)
Sino
   Escribir(N es primo)
Fin_Si

Fin
```

Código Turbo Pascal con control antes del ciclo:

```
Program número_primo;
Uses crt;
var
    N,S,J : integer;

 Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor N ");
    readln(N);
    J=2; S:=0;
    while J<=N mod 2 do
    begin
      if N mod J =0 then
          S:=S+1;
      J:=J+1;
    end;
    gotoxy(10,12);
     if S=0 then
       write("N es primo")
     else
       write("N no es primo[);
    repeat until keypressed;
End.
```

Código Turbo Pascal con control después del ciclo:

```
Program número_primo
Uses crt;
Var
  S,N,J : integer;

Begin
    clrscr;
    gotoxy(10,10);
    write("Digite un valor ");
    readln(N);
    J:=2; S:=0;
    Repeat
    begin
      if N mod J =0 then
          S:=S+1;
      J:=J+1;
    end;
    until J>N mod 2;
    gotoxy(10,12);
     if S=0 then
       write("N es primo")
     else
       write("N no es primo[);
    repeat until keypressed;
End.
```

Código Lenguaje C con control antes del ciclo

```
/*  Programa numero primo  */

#include "conio.h"
#include "stdio.h"
int  S,N,J;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor N ");
    scanf("%d",&N);
    J=2; S=0;
    while (J<=N/2)
    { if (N%J==0)
          S=S+1;
     J++; }
    gotoxy(10,12);
    if (S==0)
      printf("N es primo");
    else
      printf("N no es primo");
    getch();
}
```

Código Lenguaje C con control después del ciclo

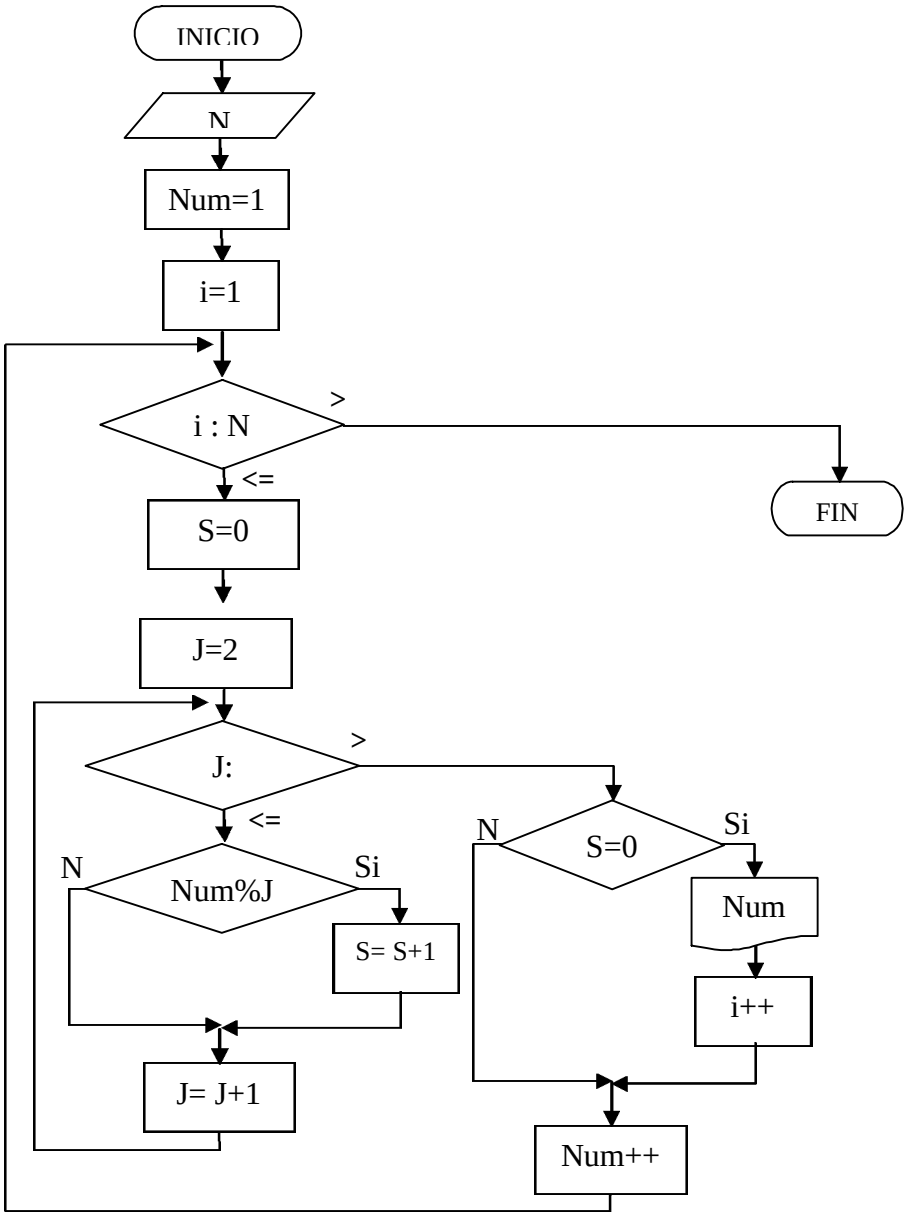
```
/*  Programa numero primo */

#include "conio.h"
#include "stdio.h"
int  S,N,J;
main()
{   clrscr();
    gotoxy(10,10);
    printf("Digite un valor N ");
    scanf("%d",&N);
    J=2; S=0;
    do
    { if (N%J==0)
          S=S+1;
      J++;    }
    while (J<N/2);
    gotoxy(10,12);
    if (S==0)
       printf("N es primo");
    else
       printf("N no es primo");
    getch();
}
```

7. Desarrolle un algoritmo que le permita realizar la escritura de los primeros N números Primos.

Análisis: En este ejercicio se involucra el concepto anterior de numero primo y se está adicionando una estructura cíclica que se encargara de contar los N números primos que se desean escribir.

Algoritmo:

Diagrama de flujo



Seudo código

```
Inicio
  Leer(N)       Num=1
  i=1
  mientras i<=1 hacer
  inicio
     S=0        J=2
     Mientras j<= Num/2 hacer
     inicio
       If Num%j=0 entonces
           S=S+1
        j=j+1
     fin_mientras
     si S=0 entonces
     inicio
        Escriba(num)
        I=i+1
     fin_si
     Num=Num+1
  fin_mientras
fin
```

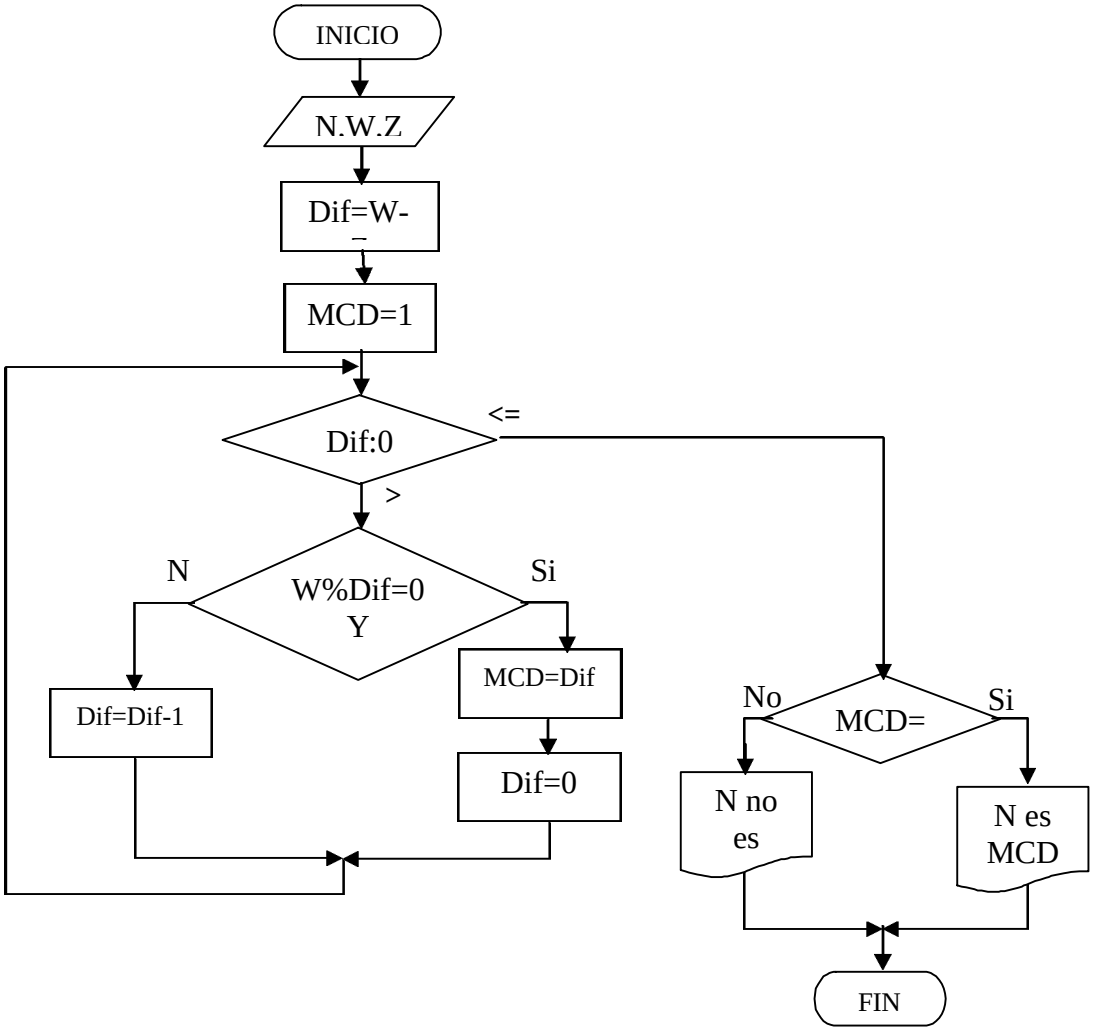
```
/* Programa que escribe los
primeros N números primos */
#include "conio.h"
#include "stdio.h"
int i,j,Num,N,S;
main()
{  clrscr();
   gotoxy(10,6); printf("Digite un valor N ");
   scanf("%d",&N);
   Num=1;
   i=1;
   while (i<=N)
   { S=0;
     j=2;
     while (j<=Num/2)
     {  if (Num%j==0)
           S=S+1;
        j++;}
     if (S==0)
     {  printf("%d ",Num);
        i++;
     }
     Num++;
   }
   getch();
}
```

8. Desarrolle un algoritmo que le permita leer un valor entero positivo N y verifique si es máximo común divisor de W y Z.

Análisis: Existen diferentes maneras de verificar si un número es el máximo común divisor de otros dos valores. Una manera es hallar la diferencia entre dichos valores y comenzar a verificar de esa diferencia hacia atrás si existe un valor que divida a los dos exactamente. En el momento de encontrar dicho valor el algoritmo no verifica más. En caso de existir un valor que los divida, al final se compara con el valor de N, si es igual es porque N es el MCD.

Algoritmo

Diagrama de flujo



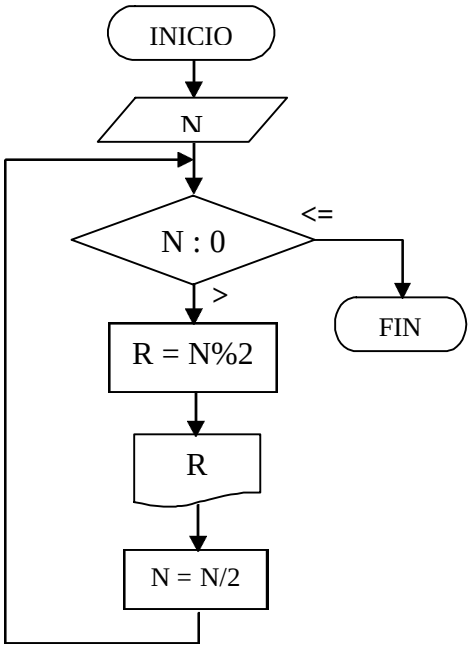
```
/* Programa que verifica si un número N
es máximo común divisor de W y Z */
#include "conio.h"
#include "stdio.h"
#include "math.h"
int Dif,N,W,Z,MCD;
main()
{ clrscr();
  gotoxy(10,6);
  printf("Digite un valor N ");
  scanf("%d",&N);
  gotoxy(10,8);
  printf("Digite un valor W ");
  scanf("%d",&W);
  gotoxy(10,10);
  printf("Digite un valor Z ");
  scanf("%d",&Z);
  Dif=abs(W-Z);   MCD=1;
  while (Dif>0)
  { if (W%Dif==0 && Z%Dif==0)
     {    MCD=Dif;
          Dif=0; }
    else
         Dif=Dif-1;
  }
  gotoxy(10,12);
  if (MCD==N)
     printf("N es MCD ");
  else
      printf("N no es MCD ");
  getch();
}
```

9. Desarrolle un algoritmo que le permita leer un valor entero positivo N dado en base decimal y convertirlo a base binaria.

Análisis: Para convertir un número representado en base decimal a base binaria es necesario dividir consecutivamente por dos el numero hasta llegar a un valor de 1. Se toma en una variable el residuo de dividir el número entre dos y el residuo de la división exacta se va escribiendo, Luego se retoma en la variable donde estaba el número inicialmente, el valor entero de la división.

Algoritmo

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en N

Mientras N >0 hacer
Inicio

Hacer R igual al residuo de dividir a N entre 2

Escribir el residuo R

Hacer el numero N igual a la parte entera de N Divido 2

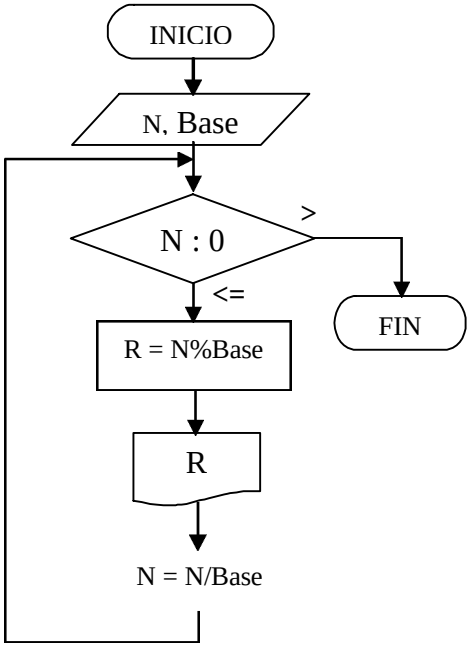
Fin mientras
Fin

```
/* Programa que lee un número
y dicho número a base binaria */
#include "conio.h"
#include "stdio.h"
int N,R,col;
main()
{  clrscr();
   col=70;
   gotoxy(10,6); printf("Digite un valor N ");
   scanf("%d",&N);
   gotoxy(10,8);
   printf("El decimal %d en base binaria es",N);
   while (N>0)
   { R=N%2;
     gotoxy(col,10);
     printf("%d",R);
     col=col-4;
     N=N/2; }
   getch();
}
```

10. Leer un número entero y almacenarlo en la variable N y leer una base numérica cualquiera en la variable B y pasar dicho número a esta Base.

Análisis: Para la solución del presente ejercicio se sigue el procedimiento anterior con la diferencia que se va a dividir consecutivamente por el valor de la base y no por dos como se hizo en el anterior.

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en N y una base y almacenarla en la variable Base

Mientras N >0 hacer
Inicio
Hacer R igual al residuo de dividir a N entre 2

Escribir el residuo R

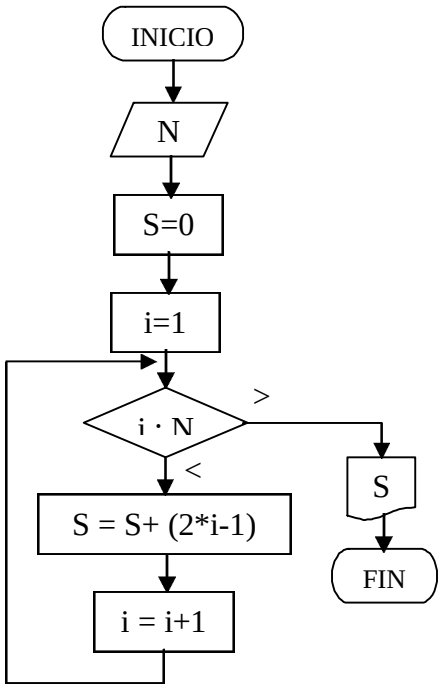
Hacer el numero N igual a la parte entera de N Divido 2
Fin mientras
Fin

```
/* Programa que lee un número
y una base y convierte dicho número
a esa base */
#include "conio.h"
#include "stdio.h"
int N,base,R,col;
main()
{  clrscr();
   col=70;
   gotoxy(10,6); printf("Digite un valor N ");   scanf("%d",&N);
   gotoxy(10,8); printf("Digite una base ");    scanf("%d",&base);
   gotoxy(10,10);
   printf("El decimal %d en base %d es :",N,base);
   while (N>0)
   { R=N%base;
     gotoxy(col,12);
     printf("%d",R);
     col=col-4;
     N=N/base; }
   getch();
}
```

11. Desarrolle un algoritmo que le permita leer un valor entero positivo N y sacar su cuadrado sumando los primeros N impares.

Analisis:

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en N

Hacer S igual a 0
Hacer i igual a 1

Mientras i sea menor o igual a N hacer
Inicio

Hacer S igual a S mas 2 por i menos 1

Hacer i igual a i mas 1

Fin mientras
Escribir el valor de S

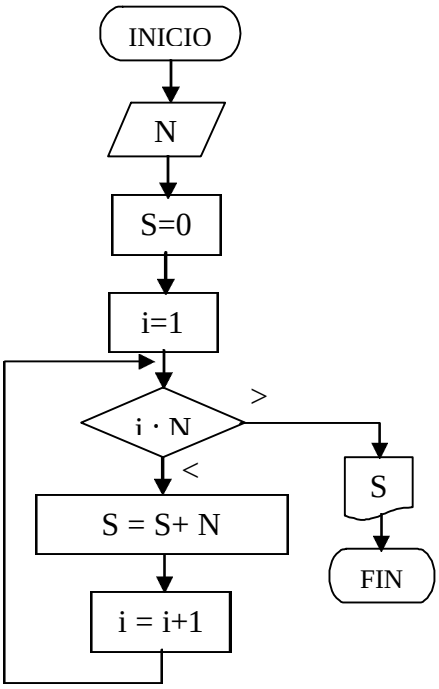
Fin

```
/* programa que escribe el cuadrado de
un número sumando N impares */
#include "conio.h"
/* programa que lee un valor N y
calcule su cuadrado */
#include "stdio.h"
int i,N,S;
main()
{  clrscr();
  gotoxy(10,6); printf("Digite un valor N ");
  scanf("%d",&N);
  S=0;
  i=1;
  while (i<=N)
  {   S=S+(2*i-1);
      i++; }
  gotoxy(10,8);
  printf("El cuadrado de %d es %d ",N,S);
  getch();
}
```

12. Desarrolle un algoritmo que le permita leer un valor entero positivo N y calcular su cuadrado sumando N veces N.

Análisis : Para desarrollar el ejercicio basta con declarar un acumulador para llevar la suma de los N términos y generar un ciclo que se repita esas N veces.

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en N

Hacer S igual a 0
Hacer i igual a 1

Mientras i sea menor o igual a N hacer
Inicio

Hacer S igual a S más N

Hacer i igual a i más 1

Fin mientras
Escribir el valor de S

Fin

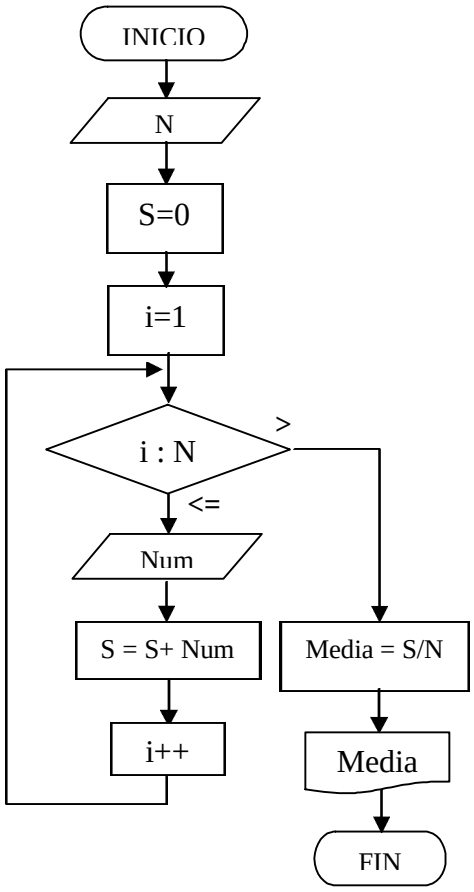
```
/* programa que escribe el cuadrado de un
número sumando N veces el número */
#include "conio.h"
/* programa que lee un valor N y
calcule su cuadrado sumando N veces */
#include "stdio.h"
int i,N,S;
main()
{  clrscr();
   gotoxy(10,6); printf("Digite un valor N ");
   scanf("%d",&N);
   S=0;
   i=1;
   while (i<=N)
   { S=S+N;
     i++; }
   gotoxy(10,8);
   printf("El cuadrado de %d es %d ",N,S);
   getch();
}
```

13. Desarrolle un algoritmo que le permita leer N valores y calcular la media aritmética $(x_1 + x_2 + x_3 + ...x_n)/n$

Análisis : para poder calcular el valor de la media aritmética a N valores es necesario dentro de una estructura cíclica leer y almacenar los valores y al finalizar el ciclo calcular el valor del promedio.

Las tareas a ejecutar Dentro del ciclo son: Leer
Acumular
Fuera del ciclo son : Promediar
Escribir

Diagrama de flujo



Seudo lenguaje

Inicio

Leer un valor y almacenarlo en N

Hacer S igual a 0

Hacer i igual a 1

Mientras i sea menor o igual a N hacer
Inicio

Leer un valor y almacenarlo en Num
Hacer S igual a S + Num
Hacer i igual a i más 1

Fin mientras

Hacer media igual a S dividido N

Escribir el valor de Media

Fin

```
/* programa Calcula la media aritmética */
#include "conio.h"
#include "stdio.h"
int N, i, Num;
float Media, S;
main()
{       clrscr();
        gotoxy(10,10);
        printf("Digite El numero de términos a promediar ");
        scanf("%d",&N);
        S=0;
        i=1;
        while (i<=N)
        {
         gotoxy(10,12);clreol();
         printf("Digite el %d número ",i);
         scanf("%d",&Num);
         S=S+ Num;
         i++;
        }
        Media=S/N;
        gotoxy(10,14);
        printf("La media Aritmética es %f",Media);
        getch();
}
```

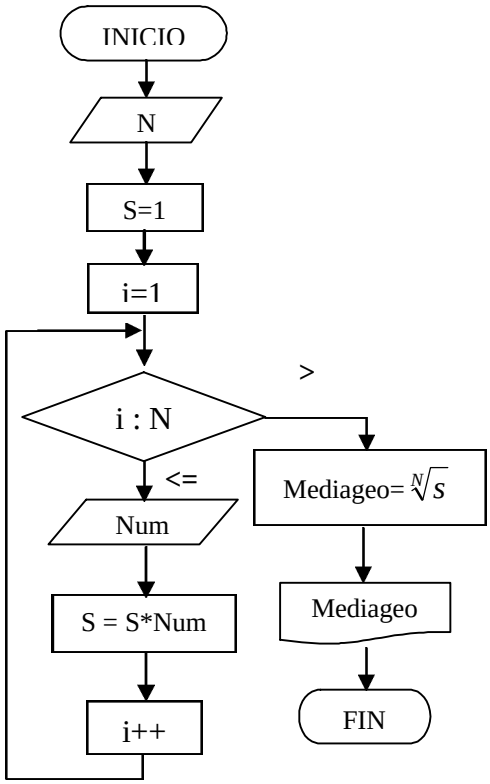
14. Desarrolle un algoritmo que le permita leer N valores y calcule con dichos valores la media geométrica $\sqrt[n]{x_1 * x_2 * x_3 * ... * x_n}$

Análisis: Leer los n valores no tendrá inconvenientes porque se sabe que con la ayuda de una estructura de programación cíclica se puede realizar esta labor. Ahora la tarea a realizar es la multiplicación sucesiva de los términos que se irán a leer. En razón de lo anterior es necesario declarar una variable que servirá de acumulador de la multiplicación sucesiva con un valor inicial de 1, ya que este valor no afectará el resultado final. Al final del ciclo se realizará el calculo de la media geométrica.

Las tareas a ejecutar	Dentro del ciclo son:	Leer
		Acumular multiplicación
	Fuera del ciclo son :	Calcula la media
		Escribir el resultado

Algoritmo

Diagrama de flujo



Código

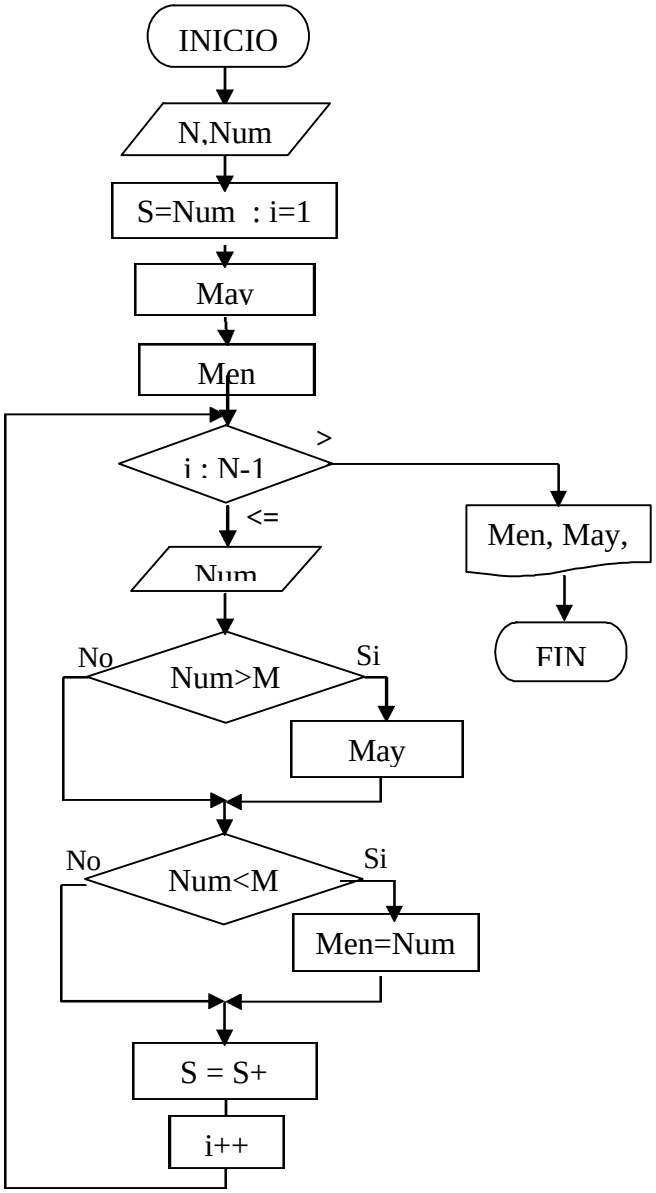
```
/* programa Calcula la media geométrica */
#include "conio.h"
#include "stdio.h"
#include "math.h"
int  N, i, Num;
float Media, S;
main()
{       clrscr();
        gotoxy(10,10);
        printf("Digite El número de términos a promediar ");
        scanf("%d",&N);
        S=0;
        i=1;
        while (i<=N)
        {
         gotoxy(10,12); clreol();
         printf("Digite el %d número ",i);
         scanf("%d",&Num);
         S=S*Num;
         i++;
        }
        Media = pow(S,1/N);
        gotoxy(10,14);
        printf("La media Geométrica es %f",Media);
        getch();
}
```

CODIFICAR EN C LOS SIGUIENTES EJERCICIOS

15. Desarrolle un algoritmo que le permita leer N valores, sumar todos los valores y decir cual es el número mayor, cual es el menor y cual es la suma.

Análisis: El desarrollar algoritmos nos permite plantearlos de múltiples maneras. En el caso del ejercicio planteado es necesario leer un primer valor y asumir a ese valor como mayor y a su vez como menor. Luego se desarrolla la lectura de los N-1 valor restantes y cada vez que se tenga un nuevo valor se compara para saber si es mayor o menor de los almacenados anteriormente. Esta tarea se desarrolla con la ayuda de una estructura de decisión.

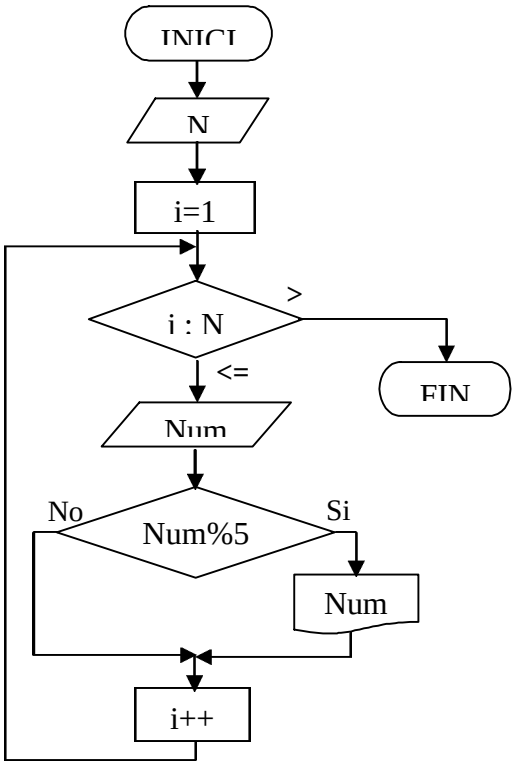
Diagrama de flujo



16. Desarrolle un algoritmo que le permita leer N valores y escriba los que sean múltiplos de 5.

Análisis: Apoyados en una estructura de programación cíclica se leen los N valores y con una estructura de decisión se decide si el numero leído es o no múltiplo de cinco para escribirlo en caso afirmativo. Es de anotar que se desarrolla el ejercicio con una estructura completamente anidada dentro de otra.

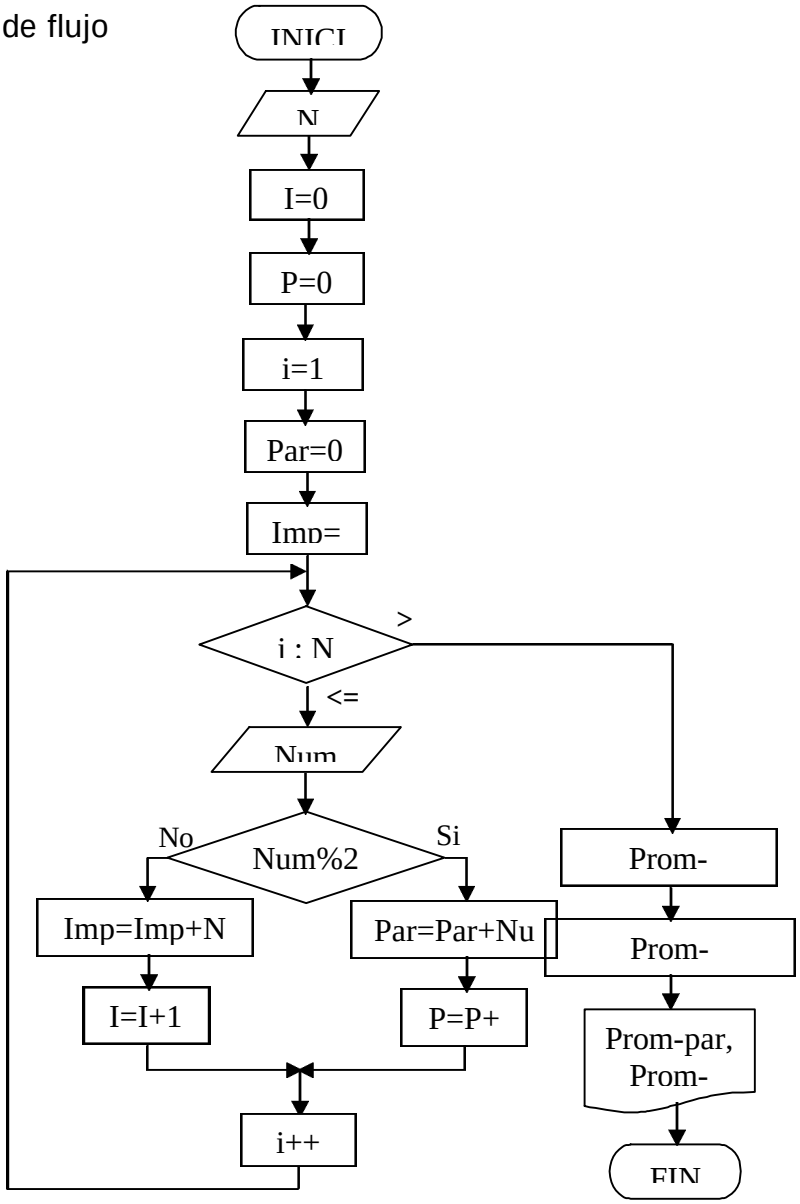
Diagrama de flujo



17. Desarrolle un algoritmo que le permita leer N valores y escribir independientemente el promedio de pares e impares.

Análisis: Apoyados en una estructura de programación cíclica se leen los N valores y dentro del ciclo se verifica si es par o impar para así mismo acumularlo en variables independientes. Igualmente se van contando el número de términos pares e impares por aparte. Al final se promediara la suma de cada uno entre el numero de términos. Cada tarea se hace por aparte con variables diferentes.

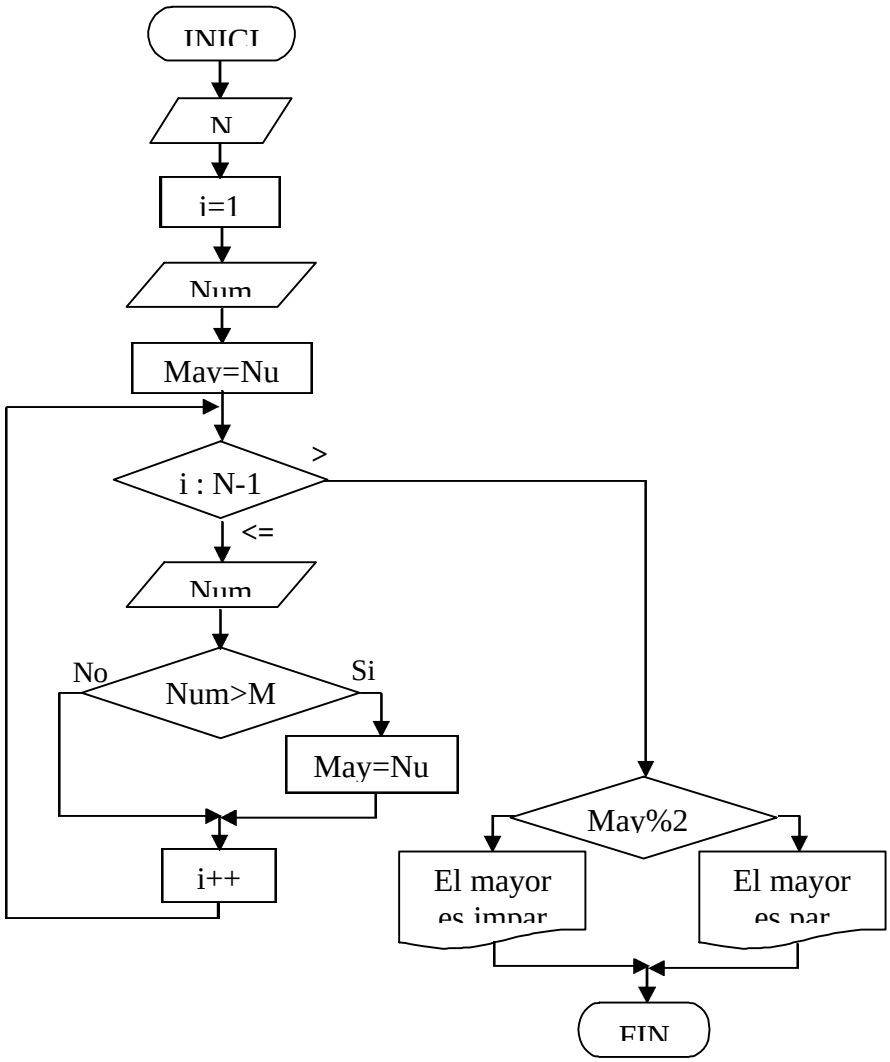
Diagrama de flujo



18. Desarrolle un algoritmo que le permita leer N valores y al final escribir si el mayor valor de los datos leídos es par o impar

Análisis: Se desarrolla la lectura de los N términos con una estructura de programación cíclica. Como se trata de mantener el mayor valor, es necesario asumir inicialmente el primer valor como mayor almacenándolo en la variable May, y de ahí en adelante cada vez que se lea un nuevo valor compararlo con May, por si es mayor entonces cambiar el valor a dicha variable. Al final del ciclo se chequea si dicho valor es par o impar.

Diagrama de flujo

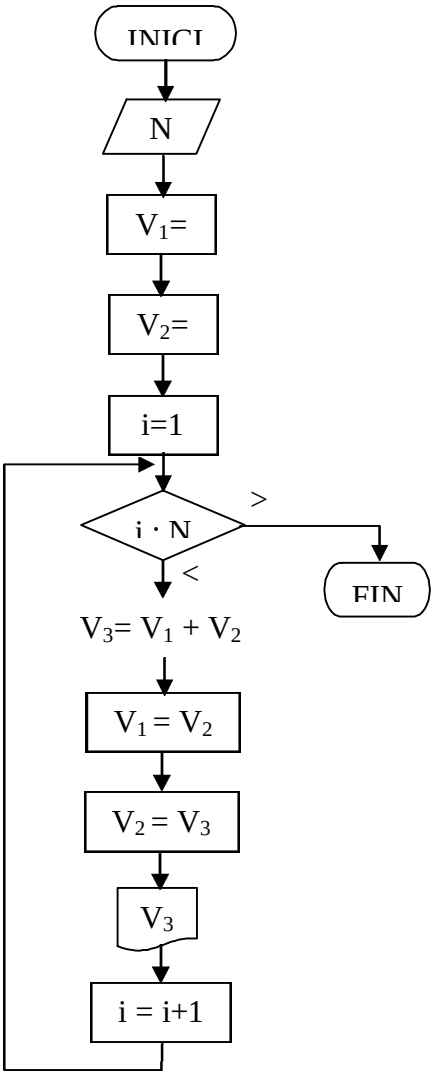


19. Genere la serie de fibonnacci iniciando con valores 1 y 2. Cada número de la serie generada es la suma de los dos anteriores.

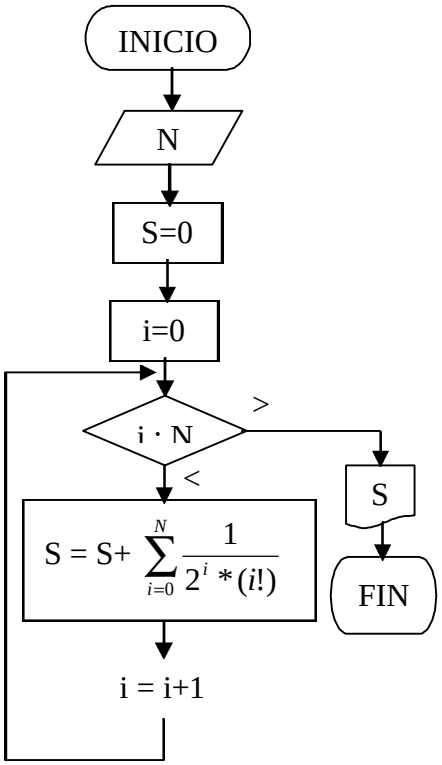
Análisis: La serie de fibonnacci inicia la generación de números a partir de dos valores iniciales. Cada nuevo valor generado es igual a la suma de los dos anteriores. En ese sentido el primer valor generado será igual a la suma de los dos valores iniciales. Si se trata de generar N valores pues se apoya en una estructura cíclica que le permita realizar esas N tareas.

Ej: 0,1 (valores iniciales) 1,2,3,5,8,13,21,34 (8 valores generados)

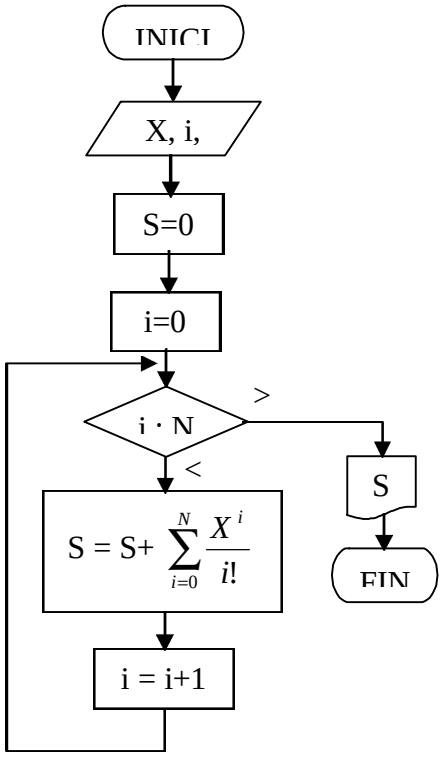
Diagrama de flujo



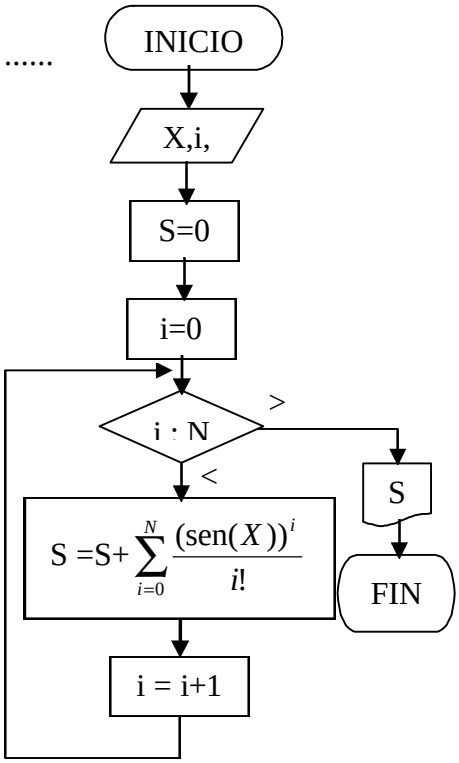
20. $\sqrt{e} = 1 + \frac{1}{2} + \frac{1}{2^2 2!} + \frac{1}{2^3 3!} + \ldots$



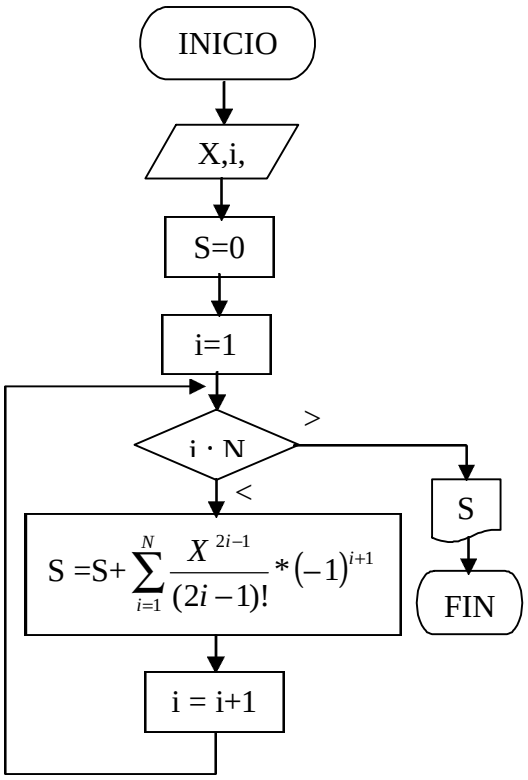
21. $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots\ldots$



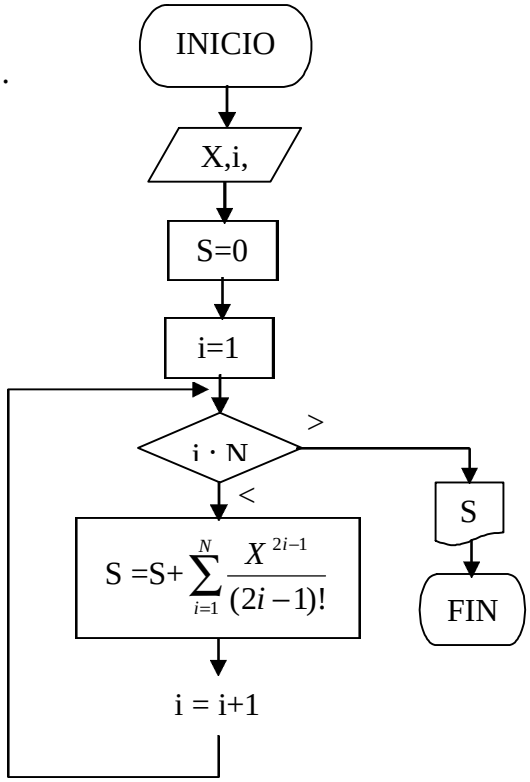
22. $e^{sin(x)} = 1 + sin(x) + \frac{sin^2(x)}{2!} + \frac{sin^3(x)}{3!} + \frac{sin^4(x)}{4!} + ...$



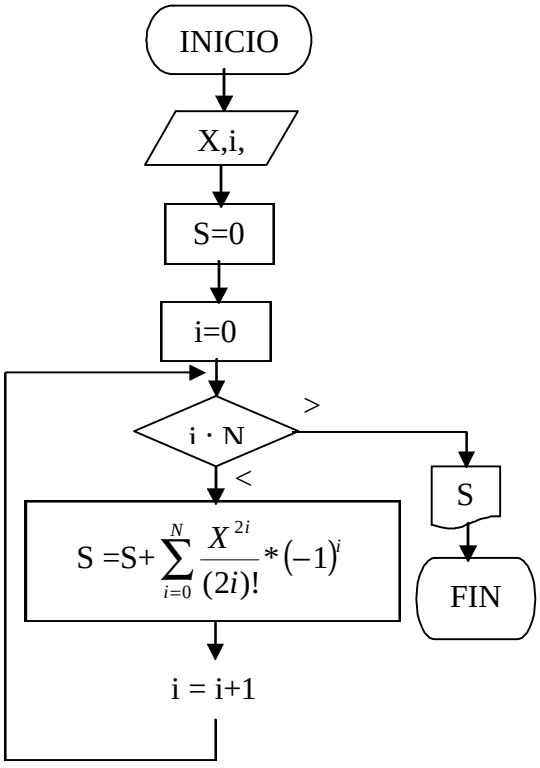
23. $Sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...$



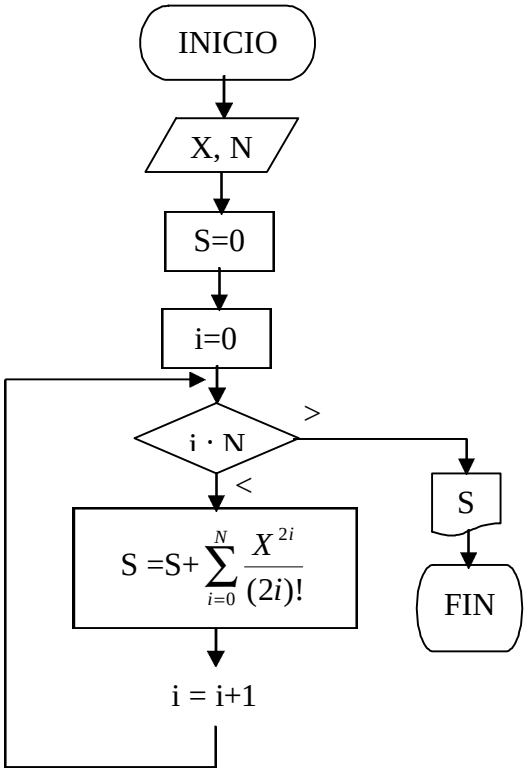
24. $Sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + ...$



25. $Cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ...$



26. $Cos(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + ...$



Los polinomios de legendre se calculan por medio de las fórmulas:

$p_0 = 1$

$p_1 = x$

.....

$p_n = (\frac{2n-1}{n}) x p_{n-1} - (\frac{n-1}{n}) p_{n-2}$

donde n=2,3,4... y X es cualquier número entre -1 y 1. Desarrolle un algoritmo que genere una tabla de p_n vs X, para cualquier valor de n hasta n=10; generando 201 valor de p_n en cada tabla, para valores igualmente distanciados de x (esto es hacer x= -1,-0.99,-0.98,....0,0.01,0.02,...0.99,1).

El coeficiente binomial $\frac{n}{i}$ para enteros no negativos n e i (donde i<=n) se define como: $\left(\frac{n}{i}\right)=\frac{n!}{(n-i)!i!}$

Se tienen dos números almacenados en las variables A y B. Desarrolle un algoritmo que permita dividir A con B pero utilizando el método de restas sucesivas. El computador debe mostrar el resultado de la división en su parte entera y en su parte residual.

Aplicaciones en Ingeniería

Las raíces reales de una ecuación se pueden calcular por diferentes métodos uno de ellos es de Newton Raphson

$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$ Donde Xn es el valor asumido inicialmente.

La Integral de una función se puede calcular de la siguiente manera.

Método rectangular: n= número de rectángulos

$\int_a^b f(x)dx = \Delta x(\sum_{i=1}^{n} f(x_i))$ donde $x_i = (a + (2*i-1)\Delta x/2)$

Método trapecial: n= número de trapecios

$\int_a^b f(x)dx = \frac{\Delta x}{2}(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i))$ donde $x_i = a + i\Delta x$

Método de simpson: n= número de arcos de parábola

$\int_a^b f(x)dx = \frac{h}{3}(f(a) - f(b) + 2\sum_{i=1}^{n}(f(x_{2i-1}) + 2f(x_{2i}))$ donde $h = \Delta x/2$

ESTRUCTURA DE SELECCIÓN MULTIPLE

Aunque el bloque if-else puede resolver múltiples comparaciones escalonadas, su uso puede ser engorroso y producir errores cuando las comparaciones son numerosas. El bloque switch permite un código mas compacto y de mayor claridad interpretativa.

El formato general del bloque es el siguiente:

```
Switch (expresión entera)
{
        case constante1:
                bloque de sentencias1;
                break;
        case constante2:
                bloque de sentencias2;
                break;
        case constante3:
                bloque de sentencias3;
                break;
        case constante4:
                bloque de sentencias4;
                break;
        default
                bloque de sentencias por defecto;
}
```

donde expresión entera es una expresión que devuelve un valor tipo entero o constante de caracter.
ConstanteN es un numero entero o una constante de carácter.
Break es una orden imperativa de salida de bloque.
Default indica que de no cumplirse ninguna de las condiciones el programa ejecutara el bloque de sentencias por defecto.

La diferencia de la estructura switch con el if son las siguientes:

- la sentencia switch solo puede comprobar la igualdad
- no pueden existir dos sentencias case en el mismo switch
- las constantes de tipo carácter se convierten automáticamente en enteros.

Ejemplo:

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

void menu();
void rectangular();
void trapezoidal();
void simpson();
void limites();
float funcion(float x);

char op;
float i,dx,xi,n,li,ls,area,s;
main()
{   do
    {
          menu();
          switch (op)
          { case '1':rectangular();   break;
            case '2':trapezoidal();    break;
            case '3':simpson();        break;
          }
    }
    while (op!='0');
}
void menu()
{ clrscr();
  gotoxy(10,6);printf("Menu Principal");
  gotoxy(10,10);printf("1. Integral modelo Rectangular");
  gotoxy(10,11);printf("2. Integral modelo Trapezoidal");
  gotoxy(10,12);printf("3. Integral modelo Simpson");
  gotoxy(10,13);printf("0. Opcion de Salir");
  gotoxy(10,20);printf("Digite una opcion [ ]");
  gotoxy(29,20);op=getch();
}
void limites()
{  clrscr();
   gotoxy(10,2);printf("Digitada de parametros para solucion a integral");
   gotoxy(10,5);printf("Digite el limite inferior : ");scanf("%f",&li);
   gotoxy(10,7);printf("Digite el limite superior : ");scanf("%f",&ls);
   gotoxy(10,9);printf("Digite el numero de subareas : ");scanf("%f",&n);
   dx=(ls-li)/n;
   gotoxy(10,11);printf("El valor del dx es %f ",dx);
}
```

```
void rectangular()
{  limites();
   s=0;
   i=1;
   while (i<=n)
   {  xi=(li+(2*i-1)*dx/2);
      s=s+funcion(xi);
      i++;
   }
   area=s*dx;
   gotoxy(10,15);printf("El valor de integral por rectangulos es %f",area);
   getch();
}
void trapezoidal()
{  limites();
   s=funcion(li)+funcion(ls);
   i=1;
   while (i<=n-1)
   {
     xi=(li+i*dx);
     s=s+2*funcion(xi);
     i++;
   }
   area=s*dx/2;
   gotoxy(10,15);printf("El valor de integral por trapecios es %f",area);
   getch();
}
void simpson()
{  limites();
   s=funcion(li)-funcion(ls);
   i=1;
   while (i<=n)
   {
     xi=(li+i*dx);
     s=s+4*funcion(li+(2*i-1)*dx/2)+2*funcion(li+i*dx);
     i++;
   }
   area=s*dx/6;
   gotoxy(10,15);printf("El valor de integral por arcos de parábola es %f",area);
   getch();
}

float funcion(float x)
{ float f;
  f= x*x-4;
  return f;}
```

FUNCIONES

Definición

En C, una función es un bloque de instrucciones que realizan una tarea específica la cual se maneja como una unidad lógica. Esta unidad, regresa opcionalmente un valor de acuerdo al proceso que realice.

C es un programa de funciones, todo se hace a partir de ellas. La principales main() la cual, utiliza a otras que se encuentran definidas en las bibliotecas (todas las instrucciones que maneja C).

Además de éstas, se puede definir nuestras propias funciones. De esta manera se dividen tareas grandes de computación en varias más pequeñas lo que da como resultado que el programa sea más fácil de entender y se pueda manejar más eficientemente.

Además, al subdividir los programas en funciones, éstas pueden ser reutilizadas en otros programas.

En esta unidad se muestra como crear nuestras propias funciones.

La forma general para definir una función es:

```
Especificador_tipo Nombre_de_la_función (declaración de parámetros)
{
cuerpo de la función
}
```

El especificador_de_tipo : este especificador de tipo de la función define la clase de valor que regresa la función. El valor puede ser cualquier tipo que maneje el lenguaje C. En caso de que no se especifique ninguno, la función devuelve por omisión un entero.

Ej: Si el valor a retornar es un decimal entonces definimos la función de tipo float.
Si el valor a retornar es un entero entonces definimos la función de tipo int.
Y así sucesivamente

Nombre_de_la_función : es la palabra con la que se identificará la función. Cada función tiene un nombre único. Con ese nombre, en cualquier otra parte del programa se pueden ejecutar los enunciados contenidos en la función. A esto se le conoce como la llamada de la función. Una función puede ser llamada desde el interior de otra función.

Una función es independiente. Una función puede ejecutar su trabajo sin interferencia de , y sin interferir con, otras partes del programa.

Una función ejecuta una tarea especifica. Esta es la parte fácil de la definición. Una tarea es un trabajo concreto que un programa debe ejecutar como parte de su operación general, como enviar una línea de texto a la impresora, ordenar un arreglo en orden numérico o calcular una raíz cubica.

Una función puede regresar un valor al programa que la llama. Cuando el programa llama a una función, se ejecutan los enunciados que contiene. Estos, en caso de que se desee, pueden pasar información de regreso al programa que la llama.

Declaración de parámetros : La declaración de parámetros es un conjunto de variables separados por comas y con un tipo de dato específico que reciben los valores de los argumentos cuando se llama a la función.

Una función puede carecer de parámetros en cuyo caso los paréntesis estarán vacíos tanto al declarar como al mandar llamar a la función.

Por ejemplo:

```
float multiplicación (float multiplicando,float multiplicador)
{
        multiplicando=multiplicando*multiplicador;
        return(multiplicando);
}
```

La función está declarada de tipo float porque es la clase de valor que va a regresar.

Multiplicando y multiplicador es el nombre que le vamos a darlos valores que recibirá la función para trabajar con ellos, los cuales son declarados dentro de los paréntesis.

Por último se define el cuerpo de la función (delimitándose con llaves)

En este caso, se asigna el resultado de la multiplicación a multiplicando para ahorrar memoria (se declara una variable menos) y regresa el valor obtenido por medio de la sentencia return.

Return

Esta sentencia se utiliza para devolver valores generados en una función al programa desde donde se hizo el llamado de la función.

También sirven para salir de la función donde se encuentra y continuar con la instrucción posterior a la función que lo llamó.

En la función anterior return(multiplicando); ó return multiplicando; devuelve el contenido de multiplicando al programa principal. En caso de que sólo se quiera salir de la función, no es necesario indicarle parámetros, basta con return();.

Todas la funciones, excepto las de tipo void (Es el tipo de datos que no tiene valor) generan valores que se transmiten al programa principal. Estos valores son de tipo int (entero) por omisión, pero puede regresarlos de todo tipo si así se declara; por ejemplo, en la función del ejemplo, la función regresa un valor de tipo float (flotante).

Cuando se quiere utilizar el valor que devuelve la función en el programa principal, es necesario asignar la función a una variable del tipo de dato que va a regresar la función.

```
#include "conio.h"
#include "stdio.h"
#include "math.h"
float fun(float x);
float X,Z;
main()
{  clrscr();
   gotoxy(10,6); printf("Digite un valor X ");
   scanf("%f",&X);
   Z=fun(X);
   printf("La funcion evaluada en %4.2f es = %10.3f",X,Z);
   getch();        }

float fun(float x)
{ float fx;
  fx=pow(x,7)-4*pow(x,3)+3*x;
  return fx;       }
```

Reglas de las funciones

- Una función puede tomar cualquier numero de parámetros, o ninguno. En caso de tener varios parámetros solo devolverá un valor.
- Una función se puede diseñar para que devuelva un valor, sin que sea obligatorio que lo haga.
- Si una función tiene un tipo de devolución void, no devolverá ningún valor. Cuando intenta obtener un valor de una función con tipo de devolución void, se produce un error de compilación. Una función que devuelva void no necesita contener una instrucción return, aunque puede tenerla si así lo desea. Ambos métodos son aceptables. Si no incluye una instrucción return, la función se devuelve automáticamente cuando llega al final del bloque de función (la llave de cierre).
- No se puede declarar funciones dentro de funciones ya que todas están al mismo nivel.
- Tampoco se puede ingresar al código de una función si estamos fuera de la misma.
- Las variables que se declaran en las funciones, son locales y no pueden ser utilizadas fuera de esa función.
- Se pueden pasar variables a las funciones por valor, por apuntador o por referencia.

Además, si se utiliza una función dos o más veces, la segunda vez que se llame a la función las variables locales no contendrán el valor que obtuvieron al ejecutar la primera vez la función y así sucesivamente ya que se crean al entrar a la función y se destruyen al salir.

Si el llamado a la función se hace desde dentro de la misma función, los datos de las variables que estén dentro de la función se mantendrán.

Argumentos de las funciones:

Es muy común que las funciones utilicen argumentos es decir, que necesiten de algún valor o valores externos dentro de su propio código. Estos valores se pasan mediante variables llamadas parámetros formales de la función las cuales se declaran dentro de los paréntesis que suceden al nombre de la función o bien, después de estos y antes de la llave de comienzo.

Asegúrese de que de que los argumentos utilizados al declarar la función sean del mismo tipo que los usados para llamar la función. Si hay algún error en los tipos, el compilador no mandará mensaje de error pero se obtendrán resultados inesperados.

Puedes utilizar a las variables que son parámetros formales como cualquier otra variable local es decir, se les puede hacer asignaciones o usarlos en cualquier expresión permitida por C.

Existen dos formas de pasar argumentos a una función:

La primera es por medio de las llamadas por valor:

Llamadas por valor

Consiste en sólo pasar el contenido de la variable utilizada como argumento a la subrutina. De esta manera, los cambios efectuados en los parámetros de la función no afectan a las variables (globales) que se utilizaron para hacer la llamada a la función.

```
#include "conio.h"
#include "stdio.h"
float cuadrado(float X);
float Z,X;
main()
{
   clrscr();
   gotoxy(10,2);printf("Cuadrado de un numero");
   gotoxy(10,5);printf("Digite el primer numero  : ");scanf("%f",&X);
   Z=cuadrado(X);
   gotoxy(10,9);printf("El cuadrado de %f es %f",X,Z);
   getch();
}

float cuadrado(float y)
{ float X;
  X=y*y;
  return X;
}
```

Llamadas por referencia

La segunda forma es mediante las llamadas por referencia en la cual, lo que se pasa a la subrutina es la dirección de la variable que se está mandando como parámetro. De esta manera, los cambios que sufra el parámetro dentro de la subrutina, se efectuarán también en la variable que se introdujo como parámetro.

La forma de pasar una llamada por referencia es pasando un puntero al argumento, de esta manera, lo que pasará es la dirección de la variable en vez de su contenido. Para esto, los parámetros se declaran de tipo puntero.

```
/*Ejemplo de funciones por valor y por referencia*/
/*Calcula dos veces el procentaje de gastos, la primera vez
utilizando una función por valor y la segunda por referencia*/


#include<stdio.h>
porcentaje_xvalor(float ingreso, float egreso)
{
   egreso=((egreso/ingreso)*100)
   printf("Usted gasta el %.2f por ciento de lo quegana",egreso);
}

porcentaje_xref(float *ingreso,float *egreso)
{
*egreso=(((*egreso)/(*ingreso))*100);
printf("Usted gasta el %.2f por ciento de lo quegana",egreso);
}

main()
{
float entrada,salida;
clrscr();
printf("Entradas: ");
scanf("%f",&entrada);
printf("Salidad: ");
scanf("%f",&salida);
porcentaje_xvalor(entrada,salida); /*Llamada a la función porcentaje utilizando
paso de parámetros
por valor*/

printf("\n\n");
porcentaje_xref(&entrada,&salida); /*Utilización de la función porcentaje con paso
de
parámetros por referencia*/
getch();
}
```

En el programa anterior, realizamos la misma tarea dos veces pero de diferente manera.
En porcentaje_xvalor(entrada,salida) mandamos el contenido de entrada y salida a la función donde son recibidos por ingreso y egreso respectivamente. De esta manera, el cálculo del porcentaje se hace internamente es decir, utilizando las variables definidas en la función. Tanto entrada como salida, no se modifican.

La función porcentaje_xref(&entrada,&salida) también obtiene el mismo resultado, pero en este caso, en vez de pasar los valores existentes en las variables, pasamos su dirección; por lo que trabajamos directamente con ellas dentro de la función aún cuando las llamemos de diferente manera(ingresos y egresos). En esta ocasión, las variables globales si se modifican.

Para comprobar esto, después de compilar el programa (que no muestre errores), en vez de ejecutarlo con Ctrl-F9, dale F8, esto te permitirá correr el programa paso a paso.

Para ver los valores que van tomando las variables, teclea Ctrl-F7 y escribe el nombre de la variable que desees observar tal cual aparezca en el programa luego pulsa <ENTER>. Aparecerá una ventana en la parte inferior de tu programa en la cual se indicará el estado o valor de la o las variables que hayas especificado.

Observarás que después de haber ejecutado, la función porcentaje_xvalor, el contenido de las variables no cambiará mientras que después de haber ejecutado porcentaje_xref, salida se verá modificada porque le asignamos un valor dentro de la función.

En realidad, tanto las funciones que utilizan paso de parámetros por valor como las que utilizan el paso por referencia, pueden hacer las mismas cosas. Lo único que cambia es la manera en que trabajan en la memoria de la computadora.

Lo que tenemos que tener en cuenta para saber cuál utilizar, es si queremos que las variables que utilizaremos en el parámetro de la función se modifiquen o no.

No olvide que las variables que pase como parámetros deben ser del mismo tipo de las que están declaradas dentro de los paréntesis de la función.

Ejemplo2:

```
#include "conio.h"
#include "stdio.h"
void cambio(float *num1,float *num2);
float numero1,numero2;
```

```
main()
{ clrscr();
   gotoxy(10,2);printf("Digitada de dos números");
   gotoxy(10,5);printf("Digite el primer numero  : ");scanf("%f",&numero1);
   gotoxy(10,7);printf("Digite el segundo numero : ");scanf("%f",&numero2);
   cambio(&numero1,&numero2);
   gotoxy(10,9);printf("El primer numero ahora es %f\n",numero1);
   gotoxy(10,10);printf("El segundo numero ahora es %f\n",numero2);
   getch();
}

void cambio(float *num1,float *num2)
{ float *temp;
  *temp=*num1;
  *num1=*num2;
  *num2=*temp;
}
```

Creación de Bibliotecas propias:

Si se quiere utilizar funciones definidas por el usuario en varios programas, se puede hacer, creando una biblioteca propia. Esto se logra de la siguiente manera:

- Define tus funciones en un nuevo archivo. Manda llamar las librerías estándar de C, que necesites.
- No utilice la función main().
- Compílalo
- Cuando la compilación sea exitosa, se generará un archivo con el mismo nombre que el tuyo pero con la terminación Obj. Este archivo deberá ser incluido preferentemente en el mismo directorio que se encuentre la biblioteca. En caso contrario, se debe dar la ruta en la sección Directorios del menú Options.
- En el programa donde quieras utilizar esta unidad sólo tendrás que mandarla llamar al principio del programa de la siguiente manera:

```
#include "nombre_archivo"
```

Después de esto, puedes llamar a las funciones que tengas definidas en esta librería normalmente sin tener que declararlas al principio del programa.

Además de funciones, en la biblioteca también puede definir constantes, macros, etc.
Intenta pasar a una biblioteca las funciones que hemos visto en este módulo para que puedas utilizarlas posteriormente.

Programa ejemplo:

En el programa que se muestra a continuación, se convierte un número en hexadecimal a su representación decimal. Es una demostración muy sencilla de lo que es una función ya que en este manual existen otros programas que contienen funciones en los cuales podrás reafirmar este conocimiento.

Compílalo y ejecútalo para que puedas comprenderlo mejor.

```
/*Programa que convierte un número en hexadecimal a decimal*/
#include<math.h>
#include<string.h>
#include<conio.h>
void main()
{       char hexa[10];
        float numero;
        clrscr();
        printf("Numero hexadecimal (mayúsculas): ");
        gets(hexa);
        numero=hex_dec(hexa);
        printf("\nEn decimal es : %.0f",numero);
}

float hex_dec(char cadena[])
{ int i,j;
  char letra;
  float decimal=0;
  float temp=0;
  i=strlen(cadena);
  for (j=0;i>0;j++,i--)
  { letra=cadena[i-1];
    switch(letra)
{ case 1:temp=(1*pow(16,j)); break;
  case 2:temp=(2*pow(16,j)); break;
  case 3:temp=(3*pow(16,j)); break;
  case 4:temp=(4*pow(16,j)); break;
  case 5:temp=(5*pow(16,j)); break;
  case 6:temp=(6*pow(16,j)); break;
  case 7:temp=(7*pow(16,j)); break;
  case 8:temp=(8*pow(16,j)); break;
  case 9:temp=(9*pow(16,j));    break;
  case 0:temp=(0*pow(16,j));    break;
```

```
  case 'A':temp=(10*pow(16,j)); break;
  case 'B':temp=(11*pow(16,j)); break;
  case 'C':temp=(12*pow(16,j)); break;
  case 'D':temp=(13*pow(16,j)); break;
  case 'E':temp=(14*pow(16,j)); break;
  case 'F':temp=(15*pow(16,j)); break;  }
  decimal+=temp;  }
  return(decimal);   }
```

Funciones recursivas y Algoritmos Recursivos

Una función que se llama a si misma un numero indefinido de veces, que se controla por la variación de un cierto valor de una o varias variables dentro de la misma función, recibe el nombre de función recursiva. Un clásico ejemplo de recursividad es el calculo del factorial de un numero entero.

Si bien la recursividad puede añadir brillantez a los programas, tenga en cuenta que su uso aumenta el trabajo de la pila y en muchos casos podrá realizar la tarea con mayor facilidad acudiendo a un método de iteración clásica

Ejemplo:

5! = 5*4*3*2*1
un pequeño análisis de este ejemplo bastara para entender la recursividad:
5! Es lo mismo que 5*4!
4! Es lo mismo que 4*3! y asi sucesivamente.

En general N! = N*(N-1)!. Utilizando esta expresión se puede obtener directamente un algoritmo recursivo para definirlo en términos de factorial.

```
#include "conio.h"
#include "stdio.h"
int fun(int x);
int X,Z;
main()
{  clrscr();
   gotoxy(10,6); printf("Digite un valor X ");
   scanf("%d",&X);
   Z=fun(X);
   gotoxy(10,8);   printf("el factorial es %d",Z);
   getch();
}

int fun(int N)
{  if (N==1)
```

```
      return 1;
   else
   {  N=N*fun(N-1);
      return N;
   }
}
```

ARREGLOS

- Definición
- Arreglos unidimensionales
- Forma de acceso a un elemento específico del arreglo
- - Paso de arreglos a funciones
- - Utilización de arrays unidimensionales como cadenas
- - Arreglos bidimensionales
- - Arreglos multidimensionales
- - Inicialización de arreglos con tamaño
- - Inicialización de arreglos sin tamaño

Definición

Un arreglo es un conjunto de elementos del mismo tipo agrupados en una sola variable. También se les conoce con el nombre de arreglos. Cada posición de almacenamiento en un arreglo es llamada un elemento del arreglo.

Para ingresar a un elemento en particular, utilizamos un índice. Existen arreglos unidimensionales, bidimensionales y tridimensionales.

Su uso más común es en la implementación de cadenas de caracteres. Recuerda que en C no existen variables de tipo cadena por lo cual se utiliza un arreglo de caracteres.

Físicamente, un arreglo es un conjunto de localidades de memoria contiguas donde la dirección más baja corresponde al primer elemento y la dirección más alta al último.

En un arreglo de n elementos, éstos ocuparan desde la casilla 0 hasta la n-1. Por si mismo, el nombre del arreglo apunta a la dirección del primer elemento del arreglo.

Arreglos Unidimensionales

un arreglo de una sola dimensión es un arreglo que tiene solamente un subíndice. Un subíndice es un numero encerrado en corchetes a continuación del nombre del arreglo. Este numero puede identificar el numero de elementos individuales en el arreglo.

La forma general para definir un arreglo de sólo una dimensión es la siguiente:

Tipo_de_dato nombre_variable [tamaño]

ej:

```
float x [10];
int y [10];
```

tipo_de_dato se refiere al tipo de dato de cada elemento del arreglo y tamaño es la cantidad de elementos agrupados en la misma variable.

Forma de acceso a un elemento específico del arreglo

Para accesar a uno de los elementos del arreglo en particular, basta con invocar el nombre del arreglo y especificar entre corchetes el número de casilla que ocupa el elemento en el arreglo.
Por ejemplo, si queremos accesar al cuarto elemento de un arreglo de 10, se invocaría de la siguiente manera:

nombre_variable[10] se refiere al nombre con el cual se identificaran todos los datos que pertenezcan al arreglo y estén almacenados en memoria.

Recuerde que el arreglo almacena desde la casilla 0. Por tanto, en un arreglo de 10 casillas, éstas están numeradas del 0 al 9.

Paso de arreglos a funciones

La forma de pasar un arreglo a una función consiste en llamar a la función y en el argumento, especificar el nombre del arreglo sin ninguna indexación. Esto hace que se pase a la función la dirección del primer elemento del arreglo ya que en C no es posible pasar el arreglo completo como argumento. Por ejemplo:

```
main()
int conjunto[20];

clrscr();
.
.
funcion(conjunto);
.
.
}
```

Aquí, al pasar el arreglo conjunto a función, estamos pasando la dirección en memoria del primer elemento de conjunto. En caso de que dentro de la función

tuviésemos que accesar a algún elemento del arreglo, se pasa de la misma manera sólo que dentro de la función utilizaremos los corchetes para accesar al elemento deseado.
Hay tres formas de declarar un arreglo como parámetro formal: como un arreglo delimitado, como un arreglo no delimitado y como un puntero. Por ejemplo:

```
#include<stdio.h>

funcion1(int conjunto[20]) /*Delimitando el array*/
{
.
}
```

o como

```
funcion1(int conjunto[]) /*arreglo no delimitado*/
{
.
}
```

o se puede declarar como

```
funcion1(int *conjunto) /* como un puntero */
{
.
}
```

El resultado de los tres métodos de declaración es idéntico.

Utilización de arreglos unidimensionales como cadenas

El uso más común de los arreglos unidimensionales es la implementación de una cadena (conjunto) de caracteres porque recuerde que en C no existe este tipo de datos. Por tanto, definimos una cadena en C como un arreglo de caracteres que al final tiene un caracter nulo ('\0'). Por esta razón es necesario que al declarar los arreglos estos sean de un caracter más que la cadena más larga que pueda contener.

Por ejemplo si deseamos crear un cadena que contenga 5 caracteres la declaración debe hacerse como sigue:

```
char cadena[6];
```

Esto es con el fin de dejar el último espacio para el caracter nulo.
No es necesario añadir explícitamente el caracter nulo de las constantes de cadena porque el compilador de C lo hace automáticamente.

Algunas de las principales funciones que soporta C para el manejo de cadenas de caracteres son las siguientes:

Nombre Definición
strcpy(s1,s2) Copia s2 en s1
strcat(s1,s2) Concatena s2 al final de s1
strlen(s1) Devuelve la longitud de s1
strcmp(s1,s2) Compara la cantidad de elementos de s1 y s2

Si son iguales, devuelve 0; menor que cero si s1 es

Menor que s2 y mayor que 0 si s1>s2.

Para una referencia más amplia sobre estas y otras órdenes, busca estas funciones en el apéndice B.

arreglos bidimensionales

Un arreglo bidimensional es un arreglo de arreglos unidimensionales. Constituyen la forma más simple de los arreglos multidimensionales. Un arreglo bidimensional tiene dos subíndices.

Su forma general de declaración es

tipo_dato variable[primer índice][segundo índice];

El primer índice corresponde a la filas y el segundo a las columnas.

Cuando se utiliza un arreglo bidimensional como argumento de una función, realmente se pasa sólo la dirección del primer elemento (el[0][0]). Sin embargo, la función que recibe un arreglo bidimensional como parámetro tiene que definir al menos la longitud de la segunda dimensión. Esto es necesario debido a que el compilador de C necesita "conocer" la longitud de cada fila para ordenar el arreglo correctamente. Por ejemplo, una función que recibe un arreglo bidimensional de 5,9 se declara así:

```
funcion(int matriz[][9])
{
.
.
}
```

No es necesario especificar la primera dimensión pero la segunda sí ya que el compilador de C la necesita para saber donde empieza la segunda fila.

También podemos utilizar arreglos bidimensionales para crear arreglos de cadenas. El primer índice indicaría el número de cadenas y el segundo la longitud máxima de las cadenas.

```
char mensajes[5][20];
```

En la declaración anterior se especifica que tenemos un arreglo llamado mensajes el cual contiene 5 cadenas de 20 caracteres cada una.

Para acceder a una cadena en especial, sólo especificamos el número de cadena (de 0 al número de cadenas menos 1). Ejemplo:

```
printf("%s",mensajes[3]);
```

Aquí mandamos imprimir la cadena número 3 de la variable mensajes. Esto sería equivalente a

```
printf("%s",mensajes[3][0]);
```

aunque es más común utilizar la primera forma.

Arreglos multidimensionales

En C, podemos crear arreglos de dos o más dimensiones el límite de dimensiones, viene dado por el compilador. Su forma general de declaración es

```
tipo_dato variable [long ind 1][longindice 2]…[long indice N]
```

donde tipo_dato es el tipo de dato de los elementos del arreglo y long ind 1, long ind 2...long ind N es la longitud de cada dimensión del arreglo. Este tipo de arreglos no se utiliza muy frecuentemente debido a el gran espacio en memoria que ocupan. Otra desventaja es que el acceso a un arreglo multidimensional dura más tiempo que el requerido por uno del tipo unidimensional.

Cuando se pasan arreglos multidimensionales a funciones, se tiene que declarar todo excepto la primera dimensión. Por ejemplo:

```
#include<stdio.h>
funcion1(int multiarreglo[][3][[4][5])
{
.
}

main()
{
int m[2][3][4][5];


funcion(m[][3][4][5]);
{.
.
}
```

Claro que si se desea, se puede especificar también la longitud de la primera dimensión.

Inicialización de arreglos con tamaño

En C, podemos inicializar (dar un valor determinado a la variable antes de usarla) arreglos globales y arreglos estáticos locales en el momento de declararlos. No es posible inicializar arreglos globales no estáticos.

Su forma general de inicialización es

tipo_dato variable [tamaño 1][tamaño2]...[tamaño] = {lista de valores};

Lista de valores es un conjunto de constantes, separadas comas, cuyo tipo es compatible con tipo_dato. La primera constante se coloca en la primera posición del arreglo, la segunda constante en la segunda posición, y así sucesivamente.
Fíjese que un punto y coma sigue a }.

A continuación tenemos la inicialización de un arreglo unidimensional:

int dígitos[5]={'0','1','2','3','4','5','6','7','8','9'};

En el caso de los arreglos unidimensionales de caracteres podemos inicializarlos abreviadamente con la forma:

char variable [tamaño]="cadena";

Por ejemplo:

char nombre[6]="clase";

Lo anterior es lo mismo que si se inicializara nombre caracter a caracter como en el ejemplo de dígitos.

char nombre[6]={'c','l','a','s','e'.'\0'};

Importante: Se debe estar seguro de que el arreglo que se declara es suficientemente largo para incluirlo. Esto es por lo que nombre tiene16 caracteres de longitud en vez de 15 que es la cantidad de letras ya que cuando se utiliza una cadena constante, el compilador proporciona la terminación nula automáticamente.

Inicialización de arreglos sin tamaño

En los arreglos con tamaño, tenemos que calcular que la longitud del arreglo fuera lo suficientemente grande para que fueran almacenados todos los elementos que deseábamos. Si tuviéramos que inicializar varios arreglos de cadena seria fastidioso contar cuantos caracteres ocupa cada arreglo.

Es posible que C calcule automáticamente la longitud del arreglo utilizando la inicialización de arreglos indeterminados la cual permite que el compilador de C cree automáticamente un arreglo suficientemente grande para mantener todos los inicializadores presentes si el tamaño del arreglo no está especificado.

El uso de la inicialización de los arreglos indeterminados permite al programador cambiar el contenido de cualquiera de las cadenas sin tener que reconsiderar el tamaño del arreglo. También puede utilizarse también en arreglos multidimensionales (2 o más). En este caso, se debe especificar todo, sin considerar la dimensión que se encuentra más a la izquierda para permitir al compilador de C indexar el arreglo adecuadamente. El método es similar a la especificación de parámetros de un arreglo. De este modo se pueden construir tablas de longitudes variables, y el compilador asignará automáticamente el espacio suficiente para ellas.

La ventaja de este tipo de declaración sobre la versión del tamaño determinado es que la tabla puede alargarse o acortarse sin cambiar las dimensiones del arreglo.