Instituto Politécnico

Universidad Nacional de Rosario Universidad Nacional de

Programación, nociones básicas

3º Año

Cód. 6301-16

Introducción a la Programación

Alejandro Schujman

Dpto. de Informática

Masterización: RECURSOS PEDAGOGICOS

Índice

Conceptos Básicos

Qué es un programa

Un programa de computadora es una secuencia ordenada de instrucciones que una computadora es capaz de interpretar y ejecutar.

Los programas se escriben utilizando distintos lenguajes de programación. Los lenguajes son un medio de comunicación entre nuestro lenguaje cotidiano y el juego de instrucciones que puede ejecutar la computadora. Este último en general es una secuencia de ceros y unos mientras que un lenguaje de programación está compuesto por palabras del idioma inglés o abreviaturas de las mismas.

Entonces, escribir un programa consistirá en escribir un conjunto de instrucciones en un orden determinado que indicarán a la computadora lo que deseamos que haga. Para ello debemos conocer algún lenguaje, la manera en que funciona la computadora y como se almacena la información dentro de la misma.

A lo largo de este curso utilizaremos el lenguaje de programación C.

Para escribir un programa utilizaremos un procesador de texto en el que escribiremos una a una las instrucciones que creamos necesarias y de acuerdo con las reglas del lenguaje elegido. Luego convertiremos este archivo de texto en un programa ejecutable por medio de un compilador.

Un compilador es un programa que convertirá nuestro código o secuencia de instrucciones en el lenguaje compuesto de ceros y unos que la computadora es capaz de ejecutar

Por ejemplo, este es un programa en C

```
int main () {
        printf("HOLA \n");
}
```

Si escribimos esto en un procesador de texto, lo guardamos en un archivo y compilamos obtendremos otro archivo con un programa ejecutable. Al ejecutar este programa veremos el resultado. En este caso aparecerá en la pantalla la palabra **HOLA**

Entonces para continuar, debemos conocer el conjunto de instrucciones que componen el lenguaje.

Que es un algoritmo

Un algoritmo es la secuencia de pasos que debemos dar para resolver un problema o para ejecutar una tarea. Un buen ejemplo de esto es una receta de cocina, en la misma tenemos una secuencia detallada de pasos a seguir para obtener un plato elaborado de comida.

Al escribir programas intentamos aplicar algoritmos conocidos o crear nuevos para resolver el problema encomendado. Los algoritmos no son únicos ya que siempre podremos encontrar otra manera de resolver el mismo problema. Sin embargo, habrá algoritmos más o menos eficientes teniendo en cuenta aspectos como el tiempo que requiere para obtener una solución o la cantidad de recursos (memoria, disco rígido, núcleos del microprocesador, etc) que necesitará para llegar a un resultado.

Una buena técnica de programación será además escribir estos algoritmos de manera que podamos hacer uso de ellos en otros programas. Lo debemos hacer de la manera más general posible de forma de obtener una pieza de código cerrada por la que no tengamos que preocuparnos al insertarla en un nuevo programa en el futuro.

De esta manera podremos construir nuestras partes de programas y no tendremos que empezar cada uno desde cero sino que podremos reutilizar lo que ya hemos escrito en instancias anteriores.

Variables y Tipos

Los programas que escribiremos van a manipular información. Esta información o estos datos deben ser alojados en la memoria de la computadora. Para ello disponemos de variables.

Una variable es un casillero o porción de la memoria del equipo y la representaremos y distinguiremos de otras por su nombre. Los nombres son secuencias de caracteres como letras, números o algunos signos (por ejemplo _) SIN ESPACIOS EN BLANCO. Intentaremos que este nombre recuerde lo que la variable almacena para facilitar la comprensión del código. Por ejemplo, si almacenamos la cantidad de números de un conjunto llamaremos a la variable **cantidad** o **cardinal** o **cant_num.** Podemos utilizar en principio cuantas variables queramos en nuestros programas y la primer limitación será la cantidad de memoria de que dispongamos.

Las variables tienen asociado un tipo de dato a almacenar. Este tipo se vincula con la variable al momento de declararla. El tipo de dato asociado indica qué información puede ser almacenada en esa variable. Esto es importante ya que el sistema operativo (quien administra la memoria disponible) necesita saber cuánto espacio debe reservar para cada variable. No ocupa el mismo espacio un dato numérico entero que un decimal o un caracter.

En C contamos con estos tipos de dato básicos. La lista no incluye los incluye a todos.

<table>
<tr><th colspan="2">Para Almacenar</th><th>Nombre del Tipo de Dato</th></tr>
<tr><td rowspan="4">Números</td><td rowspan="2">Enteros</td><td>int</td></tr>
<tr><td>long</td></tr>
<tr><td rowspan="2">Decimales o flotantes</td><td>float</td></tr>
<tr><td>double</td></tr>
<tr><td>Caracteres</td><td></td><td>char</td></tr>
</table>

En el caso de valores numéricos disponemos de distintos tipos que admitirán rangos de valores más grandes pero sacrificando más memoria. Por ejemplo, en un sistema de 32 bits, un int ocupa 32 bits (4 bytes) mientras que un long ocupa 64 bits (8 bytes). Si el conjunto de valores que pretendemos almacenar en memoria es lo suficientemente pequeño como para caber en un int no es necesario reservar espacio para un long.

En resumen, podemos utilizar variables en nuestros programas para almacenar información. Para poder utilizar una variable debemos declararla. Declarar una variable implica asignarle un nombre y un tipo de dato. Esto se hace de la siguiente manera

int a, b, c; Declara 3 variables de tipo int de nombres a, b y c respectivamente.
float pesos; Declara una variable de tipo float y le asigna el nombre pesos

Del tipo de dato **char** para almacenar caracteres nos ocuparemos más adelante.

Funciones

Un programa en C consta de una o más funciones. Todo programa tiene por lo menos una función denominada **main**. Para los alcances de este documento, todos los programas comenzarán a ejecutar en la primer instrucción de la función main y finalizarán en la última instrucción de esta. La función main podrá, de ser necesario, llamar a otras funciones para resolver las tareas encomendadas.

Una función en informática es un conjunto de instrucciones que realizan una determinada tarea. Esta forma de escribir código es muy útil ya que permite contar con problemas resueltos. Si en un programa necesito borrar la pantalla podré utilizar una función que realice esta tarea sin tener que escribir el código de la misma y puedo despreocuparme. Este enfoque también nos permite dividir un problema complejo en varios problemas más simples. que pude haber resuelto con anterioridad para otros programas y evitar reescribir código.

Las funciones pueden recibir información tal como las funciones matemáticas y pueden también devolver un resultado. Por ejemplo, si escribo el código para calcular la raíz cuadrada de un número decimal, para obtener el resultado deberé pasar a la función como dato o parámetro el valor decimal del cual deseo obtener la raíz. Esta función devolverá un número decimal que, resultará ser la raíz del número que pasamos como parámetro. Sin embargo existen funciones que no devuelven resultado alguno. Por ejemplo, la función que borra la pantalla simplemente ejecutará esa tarea sin devolver resultado alguno. Asimismo tenemos funciones que no reciben ningún parámetro o dato tal como la función que borra la pantalla.

La función main tendrá por ahora esta forma

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

Lo primero que vemos es la palabra **int**. Esta indica que la función devolverá como resultado un entero. La función main utiliza este entero para informar si hubo algún problema en la ejecución del programa. Notar que el programa puede ser ejecutado por el usuario o por otro programa. En este último caso, nuestro programa le informará a quien lo llamó en qué condiciones finalizó. Por norma, si el programa devuelve como resultado 0 indica que todo funcionó correctamente. Se puede construir una tabla indicando algún tipo de problema y el valor numérico correspondiente. Por ejemplo si devuelve un 3 podremos decir que no pudo conectarse a internet o si devuelve un 4 podremos decir que no pudo imprimir, etc.

La forma de indicar que valor devolverá una función es mediante el uso de la instrucción **return**. Cuando la función main llega a la instrucción return finaliza la ejecución de nuestro programa.

Luego de la palabra main (que no es ni más ni menos que el nombre de la función) podemos ver un juego de paréntesis. Estos paréntesis delimitan la lista de parámetros o información que la función recibe. En este caso, la función no espera información de quien la llame o de quien ejecute el programa y por lo tanto está vacía. Ahondaremos en este punto cuando analicemos con más profundidad el tema funciones.

El conjunto de instrucciones que conforman la función va encerrado entre llaves **{}** En este caso representamos este conjunto con una serie de líneas (----------). Lo que finalmente haga nuestro programa será lo que indiquemos mediante estas instrucciones. No existe en principio un límite para la cantidad de instrucciones que podemos escribir dentro del cuerpo de la función.

Expresiones Numéricas

Los distintos programas que generamos y utilizamos a diario manejan números. Reproducir música o videos es para la computadora transferir copiar o mover números de un archivo en el disco o sobre internet a algún periférico como puede ser la placa de video o la de sonido. También tenemos programas que calculan en base a datos numéricos o que procesan textos que en la memoria de la computadora son almacenados de forma numérica.
Entonces, si pretendemos escribir programas debemos aprender a lidiar con números, debemos aprender a almacenarlos en memoria y a realizar cálculos con los mismos.
Para almacenar valores en memoria nos valdremos de las variables que vimos anteriormente.

Operador de asignación
El operador de asignación sirve para dar valor a una variable. Este valor quedará almacenado en memoria hasta que lo cambiemos o hasta que termine el programa. Está representado de distintas maneras en distintos lenguajes de programación. En C utilizamos el signo =. Por ejemplo, el siguiente código

```
a = 3;
```

tendrá como resultado almacenar en la variable de nombre **a** el valor numérico 3. Debemos leer la instrucción de la siguiente manera

"Almacenar el valor 3 en la posición de memoria referida por la variable de nombre a"

Nótese que el operador de asignación NO es conmutativo ya que siempre almacenará lo que esté a la derecha en la posición de memoria de la izquierda. No podemos escribir

```
3 = a;
```

ya que 3 no es una posición válida de memoria ni un nombre válido para una variable.
El operador de asignación devuelve el valor asignado como resultado. Esto nos permite anidar las asignaciones. Por ejemplo

```
a = b = c = 2;
```

trabajará de la siguiente manera.
.Se almacena el valor 2 en c.
.Esta asignación devuelve como resultado el valor almacenado y este resultado es utilizado por el operador de asignación sobre b y guardará un 2 en b. Esta asignación nuevamente devolverá el valor 2 como resultado y este será utilizado por la asignación sobre a
De esta forma puedo almacenar el mismo valor en varias variables en una sola instrucción.

Operadores numéricos
Los operadores numéricos nos permiten realizar cuentas. Los que tenemos disponibles en C son los siguientes

Operador	Devuelve
+	Suma
-	Resta
*	Producto
/	División
%	Resto o módulo

Entonces, por ejemplo, 3 + 3 devolverá 6 y 3 % 2 devolverá 1 (el resto de la división entre 3 y 2)
Notar que NO EXISTE un operador para la potencia.

Orden de Precedencia
Las operaciones se resuelven de acuerdo a la siguiente regla.
Primero se resuelven la división, el producto y el resto. Luego se resuelven las sumas y restas.
Por ejemplo
3 + 4 * 2 devolverá 11 ya que primero se resuelve el producto entre 4 y 2 y luego la suma.
Si queremos forzar otro orden de resolución podemos utilizar paréntesis.
(3 + 4) * 2 devolverá 14 ya que primero se resuelve el paréntesis y luego el producto.

Expresiones numéricas
Una expresión numérica es una secuencia de números, operadores y funciones que producen como resultado un valor numérico
Por ejemplo

1) 2 + 3 * 4 - (17 - 3)
2) a * 2 + b * 4
3) 23 * abs(a)

son expresiones numéricas válidas
en la segunda expresión a y b representan variables. Para resolver la expresión la computadora reemplaza los nombres de variables por el valor que tengan almacenado en ese momento.
en la tercer expresión utilizamos la función **abs** que devuelve el valor absoluto del número almacenado en la variable **a**

Conversión de tipos
Como ya vimos, para almacenar valores numéricos contamos con dos grupos de tipos de datos. Los enteros y los flotantes. Es posible almacenar un valor entero en una variable de tipo flotante y a la inversa. Sin embargo, almacenar muchos valores enteros en variables de tipo flotantes es por lo menos un desperdicio de memoria. Adicionalmente almacenar valores de tipo flotante en variables de tipo entero conlleva el truncamiento del valor almacenado, es decir, se almacenará la parte entera del valor.
Si asignamos a una variable llamada num de tipo int

num = 3.574

en ella se almacenará el número 3

Si realizamos operaciones entre valores enteros el resultado será también un entero. Por ejemplo

3 / 2 devolverá como resultado 1

ya que devuelve la parte entera del resultado. Para obtener el resto de la división podemos usar el operador correspondiente

5 % 3 devolverá como resultado 2 (que es el resto de dividir 5 por 3)

El compilador utilizará siempre el tipo de dato más general posible para devolver un resultado de acuerdo con los tipos de dato de los operandos involucrados en la operación. Si la operación es entre enteros devolverá como resultado un entero. Si la operación involucra algún número de tipo flotante entonces el resultado se devolverá como flotante.
Si deseamos obtener el resultado de una división entre enteros como un número decimal o flotante debemos forzar alguno de los dos operandos a decimal. Podemos hacer esto de diversas maneras.

1) multiplicar alguno de los operandos por 1.0 Esta operación no modifica el valor ya que 1.0

es el elemento neutro del producto y forzará al compilador a considerar el número como decimal ya que 1.0 es un operando decimal
cociente = 18 * 1.0 / 10

2) Utilizar un **cast**. Esto significa forzar un dato o una variable o una expresión a ser de un tipo de dato en particular para la operación involucrada (y solamente para esta).
cociente = float(a) / 10
Aquí a será considerada como variable de tipo flotante para la operación en cuestión aunque la misma sea de tipo entero.

Funciones de Entrada y Salida

No nos servirá de nada realizar cálculos si no logramos que nuestro programa se comunique con el mundo exterior, es decir, necesitamos ingresar información al mismo durante su ejecución y poder mostrar los resultados obtenidos.
El ingreso de información podrá realizarse por varios medios. Podemos por ejemplo consultar algún archivo almacenado en la máquina o consultar archivos en otros equipos conectados en nuestra red o incluso otros a través de internet. Sin embargo y por ahora para ingresar información nos valdremos del teclado y guardaremos en variables la información que de este surja
Asimismo, la salida de información puede ser a un archivo en algún equipo o a una impresora pero haremos la misma por ahora sobre la pantalla.
El lenguaje C no poseen instrucciones para estas tareas. Sin embargo, podemos valernos de los recursos de la biblioteca estándar. Una biblioteca es un conjunto de funciones provistas por quien genera el compilador o por terceros para facilitar diversas tareas. En nuestro caso utilizaremos la biblioteca stdio y para ello debemos incluir el archivo cabecera de la misma antes que nada en nuestro programa mediante esta declaración

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

A partir de la ejecución de esta línea tendremos disponibles todas las funciones que componen esta biblioteca. Más adelante veremos con mayor detalle este tema

scanf
Podemos ingresar valores desde el teclado utilizando cin de la siguiente manera

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

Esto hará que el programa espere a que el operador escriba un valor y presione enter. Luego el valor que haya escrito el operador será almacenado en la variable a.
scanf lleva en este caso 2 parámetros. El primero (%d) indica que se está esperando el ingreso de un dato numérico de tipo **int**. El segundo indica el nombre de la variable donde debe ser alojado. Para esta función debemos agregar un símbolo & antes del nombre de cada variable que usemos. La razón de esto se verá más adelante.
Utilizamos la siguiente tabla para los distintos tipos de dato

Tipo de dato	letra que lo identifica después de %
int	**d**
float	**f**
char	**c**

printf
Podemos mostrar carteles y resultados utilizando printf. Para mostrar un cartel haremos lo siguiente

```
printf("Hola, buenos días");
```

Esto hará que el cartel encerrado entre comillas se muestre en pantalla. Las comillas no serán parte del cartel y solamente sirven para indicar los límites del cartel a mostrar.
A todo conjunto de caracteres encerrados entre comillas lo llamaremos **Constante** de caracteres o

Cadena de caracteres constante. La denominación de constante viene dada por la imposibilidad de cambiar el contenido durante la ejecución del programa.

También podemos mostrar el contenido de variables utilizando printf. Por ejemplo

```
a = 17;
printf("Mi edad es %d años\n", a);
```

producirá el siguiente cartel

Mi edad es 17 años

Notar aquí que cuando nos referimos a una variable NO UTILIZAMOS comillas. Además, el valor de la variable a se coloca dentro del cartel en reemplazo de %d. La tabla de reemplazo de % es la misma que para scanf.

Por último, si usamos dos printf el segundo continuará mostrando desde el lugar donde terminó el primero.

```
printf("Buenos ");
printf(" días");
```

producirá el siguiente cartel

Buenos días

Para agregar un retorno de carro usamos **\n** de la siguiente manera

```
printf("El día está\n");
printf("Soleado");
```

producirá el siguiente cartel

El día está
Soleado

Primeros Programas

Intentemos ahora agrupar todas las partes y escribir nuestro primer programa. La idea es sencilla y solo vamos a **mostrar la suma de los números 3 y 5 en pantalla**.
Veamos primero el código y luego analicemos una a una las instrucciones utilizadas.

```
#include <stdio.h>
int main () {
        int a, b, suma;

        a = 3;
        b = 5;
        suma = a + b;
        printf("La suma de %d y %d es %d\n", a, b, suma);
        return 0;
}
```

En primer lugar para tener un programa en C debemos tener por lo menos la función main. Aquí repetimos la estructura que vimos anteriormente. Dentro de la misma tenemos:
La declaración de tres variables de tipo int denominadas a, b y suma respectivamente. Dentro de las mismas almacenaremos toda la información que necesitamos.
Luego tenemos 3 asignaciones. Asignamos un 3 a a, un 5 a b y la suma del contenido de a y b a suma
Por último, mostramos el resultado en pantalla con printf y veremos el siguiente cartel

La suma de 3 y 5 es 8

Si tenemos que mostrar varios valores enteros con printf, simplemente colocamos la cantidad de %d que necesitemos y al cerrar las comillas colocamos las variables a reemplazar en el cartel separadas por , en el orden correspondiente.
Intentemos modificar el programa anterior para permitir que **sume cualquier par de enteros**. Para ello debemos permitir al usuario del programa ingresar los valores que desea sumar en vez de fijarlos en el código. lograremos esto utilizando scanf para darles valor a las variables a y b respectivamente

```
#include <stdio.h>
int main () {
        int a, b, suma;

        printf("Ingrese un número   ");
        scanf("%d", &a);
        printf("Ingrese otro   ");
        scanf("%d", &b);
        suma = a + b;
        printf ("La suma de %d y %d es %d", a, b, suma);
        return 0;
}
```

ahora nuestro programa solicitará al ejecutarse los valores que corresponden a ambos números a sumar. El primero se almacenará en a y el segundo en b y mostrará el cartel correspondiente. Notar que no podemos anticipar el contenido del cartel ya que no sabemos qué valores va a utilizar quien ejecute el programa. Esto le da más generalidad a nuestro programa ya que no solo será capaz de sumar 3 y 5 sino que podrá hacerlo con cualquier par de valores enteros.

Dado el radio de una circunferencia mostrar en pantalla el perímetro de la misma y la superficie del círculo que define.
En este caso los datos deben almacenarse en variables de tipo float ya que representan magnitudes reales.

```
#include <stdio.h>
int main () {
        float radio, perimetro, superficie;

        printf("Ingrese el radio   ");
        scanf("%f", &radio);
        perímetro = 2 * 3.14 * radio;
        superficie = 3.14 * radio * radio;
        printf ("El perimetro de la circunferencia es %f\n", perimetro);
        printf ("La superficie del círculo es %f\n", superficie);
        return 0;
}
```

Notar que tanto en scanf como en printf debemos utilizar %f ya que estamos tratando con variables de tipo float.
Notar que como no tenemos un operador para la potencia la forma de elevar radio al cuadrado es multiplicar la variable radio por si misma.

Expresiones Lógicas

Nuestros programas son capaces no sólo de realizar cálculos numéricos mediante expresiones numéricas sino de comparar valores. Para esto contamos con los siguientes operadores.

Operadores Lógicos

Dados dos valores a y b podemos utilizar los siguientes operadores.

a Operador b	Devuelve verdadero si…..
a > b	a es mayor que b
a < b	a es menor que b
a >= b	a es mayor o igual a b
a <= b	a es menor o igual a b
a == b	a es igual a b
a != b	a es distinto que b

Notar que para saber si dos valores son iguales utilizamos un DOBLE signo = ya que si colocamos uno solo hacemos una asignación.

Por ejemplo.

a > 5 devolverá verdadero si la variable a contiene un valor mayor a 5

a >= (2 * b) devolverá verdadero si tenemos en a un valor mayor o igual que el doble de lo que esté en b.

Compuertas Lógicas

En muchos casos necesitaremos combinar varias preguntas para determinar un resultado. Por ejemplo, para determinar si el valor almacenado en una variable está dentro de un determinado rango necesitamos saber si es mayor al límite inferior y a la vez menor al límite superior. Esto en matemáticas se escribe

$$a \in (3, 5) \Leftrightarrow a > 3 \wedge a < 5$$

Para resolver estas situaciones contamos con las compuertas lógicas. De todas las disponibles estudiaremos el funcionamiento de tres.

Compuerta AND

Esta compuerta combina dos preguntas y devolverá un resultado de acuerdo con la siguiente tabla. En la misma F simboliza Falso y V verdadero.

Pregunta 1	Pregunta 2	Resultado
V	F	F
V	V	V
F	V	F
F	F	F

En C la compuerta AND se simboliza mediante &&, es decir, se utilizan dos símbolos ampersand. Notar que esta compuerta devolverá verdadero únicamente si las dos preguntas resultan verdaderas. Volviendo a nuestro ejemplo,

a > 3 && a < 5

Si a resulta mayor que 3 Y AL MISMO TIEMPO menor que 5 entonces el resultado será verdadero. En cualquier otro caso el resultado será falso

Compuerta OR

Esta compuerta combina dos preguntas y devolverá un resultado de acuerdo con la siguiente tabla

Pregunta 1	Pregunta 2	Resultado
V	F	V
V	V	V
F	V	V
F	F	F

En C la compuerta OR se simboliza mediante ||, es decir, se utilizan dos símbolos pipe (líneas verticales). Notar que esta compuerta devolverá falso únicamente si las dos preguntas resultan falsas. Por ejemplo,

a == 3 || a == 5

Si a resulta ser 3 o 5 entonces el resultado será verdadero. En cualquier otro caso el resultado será falso

Compuerta NOT

Esta compuerta se utiliza para NEGAR una pregunta de acuerdo con la siguiente tabla

Pregunta	Resultado
V	F
F	V

En C la compuerta NOT se simboliza mediante !, es decir, se utiliza el signo de admiración. La utilidad de esta compuerta es permitir preguntar por la negativa. Existen muchas situaciones en donde resulta más simple preguntar si algo NO sucede que preguntar si lo mismo sucede. Por ejemplo, si queremos determinar si el valor de a está fuera del intervalo (3 , 5)

! (a > 3 && a < 5)

Verdadero y Falso como valores numéricos

La computadora, tal como dijimos anteriormente, maneja números. Es por esto que los operadores lógicos devuelven un valor numérico como resultado. La regla en C es que si un operador lógico devuelve 0 como resultado entonces se interpreta como FALSO y si devuelve cualquier otro número esto se interpreta como VERDADERO.

Entonces es posible, por ejemplo, guardar el resultado de un operador lógico en una variable numérica

resultado = a > 5;

si esto resultara falso (en el caso que a contenga un valor menor o igual a 5) entonces se guardará un 0 en la variable resultado , de otra forma se guardará un valor distinto a 0.

Toma de Decisión

Instrucción if

En muchos casos debemos elegir si ejecutamos o no un conjunto de instrucciones en un programa dependiendo de alguna condición. Por ejemplo, si pretendemos automatizar el control de la temperatura de una caldera podríamos hacer algo de esto.

. Cada 30 segundos medir la temperatura en la caldera.

. Luego de medir fijarse si el valor es inferior a 70 grados. En ese caso encender el mechero.

. Luego de medir fijarse si el valor es superior a 90 grados. En ese caso apagar el mechero.

Aquí vemos que existen dos instrucciones (encender y apagar) que se deberán ejecutar solo si se cumplen ciertas condiciones. Para esto aprovecharemos la instrucción if.

El formato general para if es el siguiente

```
if (expresión lógica) {
        ----------
        ----------
        ----------
}
```

y el funcionamiento es el siguiente.

El bloque de instrucciones encerrado entre llaves será ejecutado si la expresión lógica es evaluada a verdadero. Luego de esto el programa continúa Para seguir nuestro ejemplo

```
if (temp < 70) {
        encender();
}
if (temp > 90) {
        apagar();
}
```

Notar que la función **encender** se ejecutará solamente si en la variable temp hay un valor menor a 70 y la función **apagar** solamente se ejecutará si en la variable temp hay un valor mayor a 90. Notar además que no se pueden dar las dos situaciones de forma simultánea.

En el caso que dentro del if tenga SOLO una instrucción podremos obviar las llaves.

```
if (temp < 70)
        encender();
if (temp > 90)
        apagar();
```

La instrucción if tal como el resto del lenguaje evaluará las expresiones de acuerdo con su valor numérico. Por ejemplo

```
if (b % 2)
        printf("El número es par \n");
if (! (b % 2))
        printf("El número es impar \n");
```

son expresiones válidas. La expresión b % 2 será evaluada y su resultado será interpretado de acuerdo con la regla que indica que si es 0 es falso y cualquier otro valor es verdadero.

Uso de tabuladores

Cuando utilizamos una instrucción como if que maneja o controla la ejecución de varias instrucciones más, solemos colocar a estas últimas desplazadas una tabulación a la derecha. Eso no es obligatorio pero ayuda mucho a comprender el programa. Si la cantidad de instrucciones controladas por el if es grande se puede apreciar a simple vista desde donde y hasta donde abarca el if.

Uso opcional de else

La instrucción if permite ejecutar o no un bloque de instrucciones de acuerdo con una condición lógica. Podemos de manera opcional (no es obligatorio hacerlo) indicar otro bloque de instrucciones a ejecutar en caso que la condición lógica resulte falsa. Este bloque se agrega luego de la palabra else (“si no” en inglés)

Ejemplo

```
if (b % 2)
        printf(“El numero %d es impar”, b);
else
        printf(“El numero %d NO es par”, b);
```

los carteles indicarán si el contenido de la variable b es par o impar. Notar que SOLO uno de los dos carteles aparecerá en pantalla

Operador ?

El operador **?** es un operador ternario, es decir, trabaja sobre 3 operandos. El primer operando es una expresión lógica. Si esta es evaluada a verdadero devolverá como resultado el que resulte de evaluar la segunda expresión. Por el contrario, si es evaluada a falso devolverá como resultado el que resulte de evaluar la tercer expresión.

Por ejemplo:

```
if (a > b)
        result = a;
else
        result = b;
```

se puede escribir de forma más sintética

```
result = a > b ? a : b ;
```

esto guardará en la variable result el valor del mayor entre a y b. Utilizamos el **?** para separar la expresión lógica de las expresiones a evaluar como resultado y **:** para separar estas entre sí.

Notar que no es necesario el uso de paréntesis para encerrar la primer expresión ya que los operadores lógicos se resuelven antes que el operador ?

Ejemplos

1. Escribir un programa que muestre en pantalla el mayor de dos números enteros distintos

```
#include <stdio.h>
int main () {
        int a, b;

        printf("Ingrese un número entero  ");
        scanf("%d", &a);
        printf("Ingrese otro   ");
        scanf("%d", &b);
        printf("El mayor entre %d y %d es ", a, b);
        if (a > b)
                printf("%d\n", a);
        else
                printf("%d\n", b);
        return 0;;
}
```

Ingresamos dos valores numéricos que se almacenarán en las variables a y b respectivamente. Luego imprimimos un cartel que es común a ambas situaciones (que el primer número sea mayor o menor que el segundo), luego y dependiendo de la comparación en la instrucción **if** se mostrará el contenido de la variable a o el de b.

Otra forma de resolver el problema

```
#include <stdio.h>
int main () {
        int a, b, c;

        printf("Ingrese un número entero  ");
        scanf("%d", &a);
        printf("Ingrese otro   ");
        scanf("%d", &b);
        c = a > b ? a : b;
        printf("El mayor entre %d y %d es %d\n", a, b, c);
        return 0;;
}
```

Aquí reemplazamos la instrucción **if** por el operador **?** almacenando el resultado en la variable c.
Si quitamos al enunciado la exigencia de números distintos, es decir, busquemos al mayor entre dos enteros que podrían ser iguales.

```
#include <stdio.h>
int main () {
        int a, b;

        printf("Ingrese un número entero  ");
        scanf("%d", &a);
```

```
        printf("Ingrese otro   ");
        scanf("%d", &b);
        if (a == b)
                printf("Son iguales");
        else {
                printf("El mayor entre %d y %d es ",a ,b);
                if (a > b)
                        printf("%d\n", a);
                else
                        printf("%d\n", b);
        }
        return 0;
}
```

Aquí utilizamos dos instrucciones **if**. Una dentro de otra. En caso que los valores sean iguales se mostrará el cartel indicatorio. Si no es ese el caso, siguiendo el **else** mostrará el cartel correcto de acuerdo con el segundo **if**. Notar que está claramente identificado que **else** corresponde a cada **if**. Notar además que como el **else** del primer **if** abarca más de una instrucción debemos encerrarlas entre llaves.

Otra manera de resolver

```
#include <stdio.h>
int main () {
        int a, b;

        printf("Ingrese un número entero  ");
        scanf("%d", &a);
        printf("Ingrese otro   ");
        scanf("%d", &b);
        if (a == b)
                printf("Son iguales");
        if (a > b)
                printf("El mayor entre %d y %d es %d",a ,b, a);
        if (b > a)
                printf("El mayor entre %d y %d es %d",a ,b, b);
        return 0;
}
```

Aquí hemos utilizado 3 instrucciones **if** y en ninguna de ellas utilizamos **else**. Notar que cada caso excluye a los otros, es decir, si son iguales no se dará ninguno de los otros dos. Si a es mayor que b no se dará el caso de igualdad ni el de b mayor que a y lo mismo para el último.

2 Mostrar al mayor de tres números enteros distintos

```
#include <stdio.h>
int main () {
        int a, b, c;

        printf(“Ingrese un número entero  ”);
        scanf(“%d”, &a);
        printf(“Ingrese otro   ”);
        scanf(“%d”, &b);
        printf(“Ingrese el último   ”);
        scanf(“%d”, &c);

        printf(“El mayor es “);
        if (a > b && a > c)
                printf(“%d\n”,a);
        else
                if (b > c)
                        printf(“%d\n”, b);
                else
                        printf(“%d\n”, c);
        return 0;
}
```

Si a es mayor a b y a c entonces es nuestro ganador. En caso contrario el mayor estará entre b y c y se resolverá con el **if** que se encuentra dentro del primer **else**. Notar que el segundo **if** es en sí mismo una sola instrucción y por ende el primer **else** no necesita llaves.

Otra manera

```
#include <stdio.h>
int main () {
        int a, b, c;

        printf(“Ingrese un número entero  ”);
        scanf(“%d”, &a);
        printf(“Ingrese otro   ”);
        scanf(“%d”, &b);
        printf(“Ingrese el último   ”);
        scanf(“%d”, &c);

        printf(“El mayor es “);
        if (a > b && a > c)
                printf(“%d\n”,a);
        if (b > a && b > c)
                printf(“%d\n”, b);
        if (c > a && c > b)
                printf(“%d\n”, c);
        return 0;
}
```

Aquí nuevamente nos fue posible prescindir de **else** en todos los **if**. Notar que las tres condiciones son excluyentes, es decir, si resulta afirmativa alguna de las condiciones automáticamente serán falsas las otras dos. Notar también que siempre una de las tres será verdadera.

3. Dada la longitud de los lados de un triángulo determinar si es equilátero, isósceles o escaleno.

Para este problema debemos ingresar y comparar tres valores numéricos decimales entre sí. Si los tres lados de un triángulo son iguales entonces será equilátero. Si dos de ellos son iguales y distintos al tercero será isósceles y por último si todos son distintos será escaleno.
La cuestión más difícil en principio será determinar la condición de isósceles ya que debemos verificar tres condiciones de igualdad y desigualdad. Resulta más simple determinar si es equilátero, en caso de NO serlo verificar si es escaleno y por descarte declarar al triángulo isósceles.

```
#include <stdio.h>
int main () {
        float la1, la2, la3;

        printf("Ingrese longitud de primer lado  ");
        scanf("%f", &la1);
        printf("Ingrese longitud del segundo   ");
        scanf("%f", &la2);
        printf("Ingrese longitud del último   ");
        scanf("%f", &la3);

        printf("El triángulo es ");
        if (la1 == la2 && la2 == la3)
                printf("Equilátero\n");
        else
                if (la1 != la2 && la2 != la3 && la3 != la1)
                        printf("Escaleno\n");
                else
                        printf("Isósceles\n");
        return 0;
}
```

El primer if descarta la opción de equilátero. Para ello utilizamos la propiedad transitiva. El segundo if plantea la opción de escaleno. Notar que la propiedad transitiva NO aplica al operador distinto. Por descarte, si no es ni equilátero ni escaleno será isósceles.

4 Tres equipos de fútbol disputan un torneo entre sí en el formato "todos contra todos". Si conocemos los goles convertidos por cada equipo en cada partido mostrar en pantalla al ganador del torneo.

En torneos "todos contra todos" con 3 equipos tendremos 3 partidos y nuestros datos son entonces 6 valores que corresponden a los goles convertidos por cada equipo en cada partido. Si llamamos a los equipos A, B y C respectivamente entonces para facilitar el seguimiento en el programa llamaremos a nuestros datos ab, ac, ba, bc, ca, cb y serán los goles convertidos por el equipo A cuando enfrentó al B, los de A cuando enfrentó a C, etc.
Entonces debemos ingresar los datos, en base a estos asignar los puntos a cada equipo por cada partido jugado y luego determinar por mayoría de puntos al ganador

```
#include <stdio.h>
int main () {
        int ab, bc, ba, bc, ca, cb;   /* Variables para almacenar los goles */
        int puntosA, puntosB, puntosC        /* Variables para acumular los puntos de cada equipo
*/

        printf("Partido entre A y B\n");
        printf("Ingrese cant, de goles de A a B");
        scanf("%d", &ab);
        printf("Ingrese cant, de goles de B a A");
        scanf("%d", &ba);
        printf("Partido entre A y C\n");
        printf("Ingrese cant, de goles de A a C");
        scanf("%d", &ac);
        printf("Ingrese cant, de goles de C a A");
        scanf("%d", &ca);
        printf("Partido entre B y C\n");
        printf("Ingrese cant, de goles de B a C");
        scanf("%d", &bc);
        printf("Ingrese cant, de goles de C a B");
        scanf("%d", &cb);

        /* Asignación  de puntos  */
        puntosA = puntosB = puntosC = 0;   /* antes de jugar todos tienen 0 puntos */

        /* Partido entre A y B  */
        if (ab > ba)
                puntosA = puntosA + 3;    /* ganó A */
        if (ab == ba) {
                puntosA = puntosA + 1;   /* Empate  */
                puntosB = puntosB + 1;
        }
        if (ab < ba)
                puntosB = puntosB + 3;    /* ganó B */

        /* Partido entre A y C */
```

```
        if (ac > ca)
                puntosA = puntosA + 3;    /* ganó A */
        if (ac == ca) {
                puntosA = puntosA + 1;   /* Empate  */
                puntosC = puntosC + 1;
        }
        if (ac < ca)
                puntosC = puntosC + 3;    /* ganó C */

        /* Partido entre B y C */
        if (bc > cb)
                puntosB = puntosB + 3;    /* ganó B */
        if (bc == cb) {
                puntosB = puntosB + 1;   /* Empate  */
                puntosC = puntosC + 1;
        }
        if (bc < cb)
                puntosC = puntosC + 3;    /* ganó C */

        /* Tabla de posiciones */
        if (puntosA == puntosB && puntosB == puntosC)
                printf("No hay ganador. Todos los equipos tienen el mismo puntaje\n");
        if (puntosA > puntosB && puntosA > puntosC)
                printf("El ganador es el equipo A");
        if (puntosB > puntosA && puntosB > puntosC)
                printf("El ganador es el equipo B");
        if (puntosC > puntosB && puntosC > puntosA)
                printf("El ganador es el equipo C");
        return 0;
}
```

si necesitamos hacer comentarios dentro de nuestro código para facilitar su lectura utilizamos /* ….. */. Todo lo que coloquemos entre "barra asterisco" y "asterisco barra " será ignorado por el compilador.

Iteraciones

Supongamos que deseamos mostrar en pantalla todos los valores enteros entre 0 y 9. Con los conocimientos adquiridos hasta aquí el programa sería el siguiente

```
#include <stdio.h>
int main () {
        printf (“0\n”);
        printf (“1\n”);
        printf (“2\n”);
        printf (“3\n”);
        printf (“4\n”);
        printf (“5\n”);
        printf (“6\n”);
        printf (“7\n”);
        printf (“8\n”);
        printf (“9\n”);

        return 0;
}
```

Esto resulta poco práctico, además en el caso de pretender mostrar en pantalla los valores enteros entre N y M siendo N y M dos números enteros distintos tendríamos que modificar el código de nuestro programa para cada par de valores N y M.
Por suerte para resolver esta situación contamos con mecanismos que permiten repetir una o varias instrucciones una cierta cantidad de veces.

Instrucción while
La instrucción while (mientras en inglés) tiene la siguiente sintaxis

```
while (expresión lógica) {
        ----------------
        ----------------
}
```

y funciona de la siguientes manera
Mientras la expresión lógica sea verdadera se ejecutará el bloque de instrucciones encerrado entre llaves. Si en algún momento la expresión resulta falsa se continuará el programa desde la primer instrucción posterior al bloque definido por las llaves. Tal como en la instrucción if, podemos obviar las llaves si solo nos interesa repetir una instrucción.

Introducción a la Programación I

1 Mostrar en pantalla los números del 0 al 9

```
#include <stdio.h>
int main () {
        int i = 0;

        while (i < 10) {
                printf(“%d\n”,i};
                i = i + 1;
        }
        return 0;
}
```

La instrucción while se ejecuta ya que i comienza con el valor 0 que es menor a 10 y continúa ejecutando el bloque de instrucciones MIENTRAS la condición permanece verdadera. Notar que en cada vuelta el valor de i crece en 1 y por lo tanto se llegará a 10, lo que vuelve falsa la expresión lógica y terminará el while.
La función printf dentro del while muestra a cada vuelta el valor de i. también a cada vuelta cambia el valor de i. Por lo tanto se muestran en pantalla todos los valores del 0 al 9 que serán todos los valores que toma la variable i a medida que se repiten las instrucciones abarcadas por while. Cuando la variable i llega al valor 10 la condición lógica deja de ser verdadera y por lo tanto se interrumpe la iteración y el programa continuará con la instrucción que sigue, en este caso return 0.

2 **Mostrar los números del 9 al 0, es decir, invertir el orden**

```
#include <stdio.h>
int main () {
        int i = 9;

        while (i >= 0) {
                printf(“%d\n”,i};
                i = i - 1;
        }
        return 0;
}
```

Igual que en ejemplo 1, la expresión lógica es verdadera al inicio ya que i vale 9 y se pide que sea mayor o igual a 0. En cada iteración se muestra el valor de i y en este caso se reduce en 1. Cuando i toma el valor -1 la expresión se vuelve falsa y se interrumpe el while. Notar que el valor -1 nunca se muestra en pantalla.
Otra forma

```
#include <stdio.h>
int main () {
        int i = 0;

        while (i < 10) {
                printf(“%d\n”,9 - i};
                i = i + 1;
        }
        return 0;
}
```

Comenzamos con i en 0 e incrementamos a cada iteración pero mostramos en pantalla la diferencia entre 9 e i. Notar que a medida que i crece el valor a mostrar decrece comenzando en 9 y terminando en 0

El preprocesador

En la introducción de este documento vimos que los programas que nosotros escribimos deben ser compilados antes de usarse. El primer paso en la compilación de un programa en C es el preprocesador.

El preprocesador analiza nuestro código o archivo fuente y realiza diversas tareas. Es posible dar directivas al preprocesador y ya hemos visto una de ellas. Las directivas al preprocesador comienzan con un #.

Para incluir una biblioteca de funciones en nuestro programa debemos utilizar la orden #include. Esta orden le indica al preprocesador que debe buscar dentro de la carpeta que corresponda la biblioteca de funciones que indicamos y agregar al archivo ejecutable el código que corresponda.

Tenemos más directivas del preprocesador para utilizar. Una de ellas es la que permite definir macros. Una macro es un valor o una porción de código asociado a un nombre. Al momento de compilar el preprocesador reemplaza este nombre por el código o el valor asociado al mismo. Esto nos permite por ejemplo, definir constantes dentro de nuestro programa y darles un valor determinado una sola vez para evitar luego tener que buscar dentro del código cada aparición de la misma. Veamos un ejemplo

```
#include <stdio.h>
#define TOPE 10
int main () {
        int i = 0;

        while (i < TOPE) {
                printf("%d\n",TOPE - i -1};
                i = i + 1;
        }
        return 0;
}
```

el preprocesador reemplazará el valor de TOPE por el número 10 antes de compilar el programa. Este ejemplo pretende mostrar el funcionamiento de una macro simple. En programas más complejos las macros permiten controlar un determinado parámetro sin tener que buscar dentro del código cada aparición del mismo, facilitando la lectura del código y ayudando a evitar errores.

Notar que las macros NO son variables. Su valor se define cuando se escribe el código y no puede ser alterado una vez compilado el programa. Para cambiar el valor de una macro debemos corregir el código fuente y volver a compilar. Se estila aunque no es obligatorio, utilizar mayúsculas para los nombres de macros. Esto facilita la lectura del código.

Con macros se pueden hacer cosas mucho más complicadas que solo reemplazar texto. Volveremos más adelante sobre este tema.

3 **Mostrar en pantalla los enteros entre INICIO y FIN**

```
#include <stdio.h>
#define INICIO 100
#define FIN 234
int main () {
        int i = INICIO;

        while (i < FIN) {
                printf("%d\n", i};
                i = i + 1;
        }
        return 0;
}
```

otra forma

```
#include <stdio.h>
int main () {
        int i, a, b,  ini, fin;

        printf("ingrese el límite inferior");
        scanf("%d\n",&inii);
        printf("ingrese el límite superior");
        scanf("%d\n",&fin);

        i = ini;
        while (i < fin) {
                printf("%d\n", i};
                i = i + 1;
        }
        return 0;
}
```

En este ejemplo en vez de usar macros dejamos que el usuario de nuestro programa elija los valores extremos del intervalo a mostrar cada vez que utilice el programa.

4 **Mostrar en pantalla los enteros entre A y B**

```
#include <stdio.h>
int main () {
        int i, a, b,  ini, fin;

        printf("ingrese el límite inferior");
        scanf("%d\n",&a);
        printf("ingrese el límite superior");
        scanf("%d\n",&b);

        if (a < b) {
                ini = a;
                fin = b;
        } else {
```

```
                ini = b;
                fin = a;
        }

        i = ini;
        while (i < fin) {
                printf("%d\n", i};
                i = i + 1;
        }
        return 0;
}
```

En este ejemplo, los extremos pueden o no venir dados en el orden correcto. Como la idea es mostrar en forma ascendente debemos asegurarnos que se inicie por el menor de ellos.

Instrucción for

La instrucción **for** permite realizar las mismas tareas que la instrucción while pero presenta un formato más compacto ya que incluye la inicialización, la expresión lógica de control y el incremento en una misma línea. La utilización de una u otra es elección del programador y no presenta una ventajas en eficiencia con respecto a la otra.

Tiene la siguiente sintaxis

```
for (inicialización ; expresión lógica ; incremento )  {
                ----------------------------
                ----------------------------
}
```

Al igual que en while, una instrucción for repite las instrucciones encerradas entre llaves mientras la expresión lógica sea evaluada a verdadero.

La inicialización se ejecuta solo una vez al principio y sirve para dar valores iniciales a variables.

El incremento se ejecuta a cada vuelta o iteración y antes de evaluar la expresión lógica

Ejemplo

1 Mostrar en pantalla los números del 0 al 9

```
#include <stdio.h>
int main () {
        int i ;

        for ( i=0 ; i < 10 ; i = i+1)
                printf("%d\n",i};
        return 0;
}
```

En esta caso cuando llegamos a la instrucciòn **for**, la variable i toma el valor 0 (inicialización), se comprueba la condición y como es verdadera se ejecuta printf. Luego volvemos al **for**, se ejecuta el incremento (i toma el valor 1), se evalúa nuevamente la condición y como sigue siendo verdadera se ejecuta el printf. Esto continúa hasta que la variable i toma el valor 10 y la condición deja de ser verdadera. Notar que si bien la variable i finaliza con el valor 10, el printf correspondiente no se ejecuta.

Más Operadores de asignación
Cuando debemos incrementar el contenido de una variable, es decir, sumar algo al valor que ya tiene haremos lo siguiente por ejemplo:
i = i + 1
una_variable_simpatica = una_variable_simpatica + 253

podemos escribir estos ejemplos de la siguiente manera
i += 1
una_variable_simpatica += 253

que resulta una forma simplificada de escribir.

también existen operadores para acumular productos, restas y divisiones de acuerdo a la siguiente tabla

Operador	Se escribe	Efecto
+=	x += 23	Suma 23 al valor de x
-=	x -= 12	Resta 12 del contenido de x
***=**	x *= 21	Multiplica por 21 el valor de x
/=	x /= 2	divide por 2 el contenido de x

Notar que estos operadores no son imprescindibles pero ayudan a entender más claramente que se desea hacer y naturalmente a escribir menos. Existen más de estos operadores y los veremos a medida que los necesitemos.

Operadores de incremento y decremento
Si necesitamos incrementar en 1 el contenido de una variable podemos escribir la operación de las siguientes maneras

var = var +1
var += 1
var ++

cualquiera de las tres maneras tendrá el mismo efecto sobre la variable **var.** Si deseamos que se le reste 1 al contenido de var tenemos el operador --. vale decir que var -- disminuirá en 1 el contenido de var.
Los operadores ++ y -- pueden colocarse antes del nombre de una variable (prefijo) o después del mismo (postfijo) y el efecto es ligeramente diferente, veamos

postfijo	prefijo
a = 5; b = a ++;	a = 5; b = ++ a;
a guarda 6 y b guarda 5	a guarda 6 y b guarda 6

Si colocamos el operador de forma prefija el valor de la variable se incrementa antes de la asignación. Por el contrario, si el operador es postfijo primero se asigna y luego se incrementa

Instrucciones break y continue

Estas instrucciones complementan el control de los ciclos while y for. Pueden ser utilizadas en ambos casos con el mismo resultado.

Si dentro de un ciclo **while** o f**or** colocamos una instrucción b**reak e**l resultado será la inmediata interrupción del mismo. Vale decir que la computadora continuará ejecutando el programa en la primer instrucción que siga al while o for correspondiente. Esto nos permite salir antes de que un ciclo concluya y es útil en distintas situaciones, por ejemplo

ingresar a lo sumo 20 números y sumarlos. Interrumpir el ingreso si la suma es mayor a 100. Mostrar la suma y la cantidad de números ingresados

```
#include <stdio.h>
int main () {
        int i, num, suma = 0 ;

        for ( i=0 ; i < 20 ; i++) {
                scanf (“%d”, num);
                suma += num;
                if (suma > 100)
                        break;
        }
        printf (“Se ingresaron %d números y su suma es %d\n”,i, suma};
        return 0;
}
```

Aquí vemos que **for** da 20 vueltas ingresando números y luego termina. Sin embargo, si la suma de los números que se ingresan supera 100 se interrumpe aún sin haber completado las 20 vueltas.

La instrucción **continue** tiene el mismo ámbito de aplicación que **break** y su efecto es dar por concluida la presente iteración para continuar con la siguiente, vale decir que cuando se ejecuta **continue** se da por terminada la ejecución de las instrucciones incluidas en el **for** o el **while** y se continúa con la siguiente iteración. Ejemplo

Mostrar en pantalla los múltiplos de 3 menores a 100

```
#include <stdio.h>
int main () {
        int i ;

        for ( i=3 ; i < 100 ; i++)  {
                if (i % 3)
                        continue;
                printf(“%d\n”,i};
        }
        return 0;
}
```

Notar que sólo para los múltiplos de 3 no se ejecutará **continue**

Ejemplos Resueltos

1) **Mostrar en pantalla los divisores de un número natural**

La estrategia será probar entre todos los números naturales menores a N cuáles son divisores. Para saber si un cierto número es divisor nos valdremos del operador %

```
#include <stdio.h>
int main () {
        int i, N ;

        printf("Ingrese un número natural  ");
        scanf ("%d", N);
        for ( i=1 ; i <= N ; i++)
                if ( ! (N % i))
                        printf("%d\n",i};
        return 0;
}
```

El ciclo for recorre todos los valores entre 1 y N. En caso que alguno de estos valores sea divisor de N (resto 0) se lo muestra en pantalla. Notar que como 0 representa el valor lógico FALSO debemos colocar una compuerta NOT para lograr el efecto deseado

¿Podremos de alguna manera mejorar este programa?.
La respuesta es si. Una forma de hacerlo es trabajar sobre la cantidad de vueltas que da el ciclo for para lograr el objetivo. Notar que, por ejemplo, no podremos encontrar divisores más haya de la mitad del número (excepto el mismo número). Si buscamos los divisores de 1000000 no vamos a encontrar ninguno más grande que 500000. Luego y con algunas modificaciones

```
#include <stdio.h>
int main () {
        int i, N ;

        printf("Ingrese un número natural  ");
        scanf ("%d", N);
        printf ("1\n%d\n", N);
        for ( i=2 ; i <= N / 2 ; i++)
                if ( ! (N % i))
                        printf("%d\n",i};
        return 0;
}
```

Ahora mostramos 1 y N por fuera del for y hacemos que este concluya en la mitad de las vueltas obteniendo el mismo resultado. Hemos logrado mejorar la eficiencia de nuestro programa.

¿Podremos mejorar aún más este programa?.
Si, podemos pero para esto debemos pensar un poco más en cómo se obtienen los divisores. Cada vez que obtenemos un divisor i, también obtenemos otro divisor que es el resultado de N / i. Veamos ejemplos

28		100	
i	N / i	i	N / i
1	28	1	100
2	14	2	50
4	7	4	25
		5	20
		10	10

La idea es buscar los divisores que aparecen en la columna de la izquierda y calcular los de la columna de la derecha. De esta manera podremos reducir aún más la cantidad de vueltas que da nuestro for.
La pregunta es entonces ¿Cuál es el límite superior para i?.
Notar que a medida que i crece N / i decrece. El límite está dado por el valor para el cual i se hace igual a N / i, es decir

$$i = N / i$$
$$i * i = N$$
$$i = \sqrt{N}$$

es decir que nuestros posibles divisores van a ser los naturales menores a la raíz cuadrada de N. Notar que esto disminuye aún más la cantidad de iteraciones ya que si N es 100000 solo debemos buscar entre los naturales menores a 10000 (en la primer mejora este tope era 500000).

```
#include <stdio.h>
#include <math.h>
int main () {
        int i, N ;
        printf("Ingrese un número natural  ");
        scanf ("%d", N);
        printf ("1\n%d\n", N);
        for ( i=2 ; i < sqrt(N) ; i++)
                if ( ! (N % i))
                        printf("%d\n%d\n", i, N / i);
        if (! N % sqrt(N))
                printf("%d\n", sqrt(N));
        return 0;
}
```

La función **sqrt** (square root) devuelve la raíz cuadrada de N. Se encuentra en la biblioteca math.h que incluimos al principio del programa. El ciclo for se ejecuta hasta la raíz de N y cada vez que encuentra un divisor lo muestra junto al que obtenemos por cálculo (N / i). Una vez terminado el **for** nos queda el caso de los cuadrados perfectos como el 100 que pusimos en el ejemplo. Para estos números la raíz cuadrada también resulta divisor

2) **Determinar si un número es o no primo**

Recordamos que un número es primo si solo tiene a 1 y a sí mismo como divisores. El 1 no es primo por definición.

Luego, podemos utilizar el programa anterior solo que en vez de mostrar los divisores de N debemos comprobar que no los tenga.

```
#include <stdio.h>
#include <math.h>
int main () {
        int i, N, b = 1;

        printf("Ingrese un número natural  ");
        scanf ("%d", N);
        for ( i=2 ; i <= sqrt(N) ; i++)
                if ( ! (N % i)) {
                        b = 0;
                        break;
                }
        if (b)
                printf("%d es primo\n", (N);
        else
                printf("%d NO es primo\n", (N);
        return 0;
}
```

El programa solicita un número natural y busca posibles divisores. Si encuentra uno cambia el valor de b a 0 e interrumpe el ciclo (no tiene sentido buscar más divisores, si hay uno el número NO es primo). Por el contrario, si no encuentra divisores el valor de b permanece inalterado en 1.

Luego debemos saber por qué motivo terminó el for. Si terminó por el break significa que se encontró un divisor. En ese caso b vale 0 y el if colocará el cartel de número NO primo. Por el contrario, si el for concluyó porque la condición del mismo dejó de ser válida la variable b tendrá el valor 1 y el cartel a mostrar será el de número primo.

Es común utilizar variables para determinar si el programa ejecutó alguna instrucción en particular. A estas variables las denominamos bandera y se les asigna uno de dos valores distintos (en este caso 0 y 1) para cumplir con dicho fin.

3) **Mostrar los primeros 4 números perfectos**

Se dice que un número es perfecto si la suma de sus divisores, exceptuando al mismo número es igual al número.

Ejemplo, 6 es un número perfecto ya que sus divisores son 1, 2, 3 y 6. Luego, la suma de sus divisores (excepto el 6) es 1 + 2 + 3 = 6

Para resolver este problema debemos buscar los divisores de los números e ir acumulando su suma. Luego comparamos con el número en cuestión y sabremos si es o no perfecto.

```
#include <stdio.h>
#include <math.h>
#define TOPE 4
int main () {
        int i, acum, perfecto = 0, num = 2 ;
```

```
        whiile (perfecto < TOPE) {
                acum = 1;   /* 1 siempre es divisor */
                for ( i=2 ; i < sqrt(num) ; i++)
                        if ( ! (num % i))
                                acum += i + num / i;
                if (! num % sqrt(num))
                        acum += sqrt(num);

                if (acum == num) {
                        printf("%d es perfecto\n", num);   /* Si es perfecto lo mostramos */
                        perfecto++;                        /* aumentamos el contador de perfectos */
                }
                num++;  /* sea o no perfecto probamos con el siguiente */
        }
        return 0;
}
```

Notar que los ciclos for y while se pueden colocar uno dentro de otro. También es válido colocar un for dentro de un for y un while dentro de un while. A esto se lo denomina anidar y la regla es que para cada iteración del ciclo exterior se producen todas las iteraciones del ciclo interior. En principio no hay límites para anidar, vale decir que podemos anidar N ciclos uno dentro de otro.
En este programa tenemos un ciclo exterior. Para cada una de las iteraciones de este se ejecutarán todas las del ciclo interior que encuentra y acumula los divisores, es decir, vamos probando para cada entero mayor que 1 si se cumple la definición de número perfecto. El ciclo exterior (while) concluye cuando encontramos 4 números perfectos.

Apéndice A

Problemas con tomas de decisión

1. Dados dos números enteros, mostrar en pantalla si son iguales o caso contrario al mayor de ellos

2. Dados tres números enteros distintos, mostrar en pantalla al mayor y al menor valor entre ellos

3. Dados dos números determinar si uno es el cuadrado del otro

4. Dados cinco números enteros determinar si el promedio de los que sean mayores al promedio es o no par

5. Dadas las tres notas que obtuvo un alumno a lo largo del cuatrimestre determinar su promedio y si aprobó o no la materia. Para aprobar necesita un promedio mayor o igual a 6 y ninguna nota menor a 4.

6. Dados tres ángulos, determinar si pueden formar un triángulo

7. Dados los lados de un triángulo determinar si es rectángulo

8. Dados los lados de un triángulo determinar si es equilátero, isósceles o escaleno

9. Determinar la categoría que le corresponde al socio de un club conociendo su edad y antigüedad. Si es menor de 8 años es NIÑO. Si no es NIÑO y es menor de 18 años es CADETE. Si es mayor de 18 años y tiene menos de 20 años de antigüedad es ACTIVO. Con 20 años de antigüedad es PASIVO.

10. Dado el consumo de energía eléctrica de una casa durante el último bimestre en Kwh, calcular y mostrar en pantalla el importe en pesos de la factura de acuerdo al siguiente cuadro tarifario:

Kwh	**$ (pesos)**
Primeros 120	0,19
Segundos 120	0,23
Consumo restante	0,38

Además se deberá agregar una multa de 100 pesos por exceso de consumo si se superan los 1000 Kwh.

Apéndice B

1. Mostrar la suma de los números pares entre 1 y 101

2. Mostrar la suma de los múltiplos de C en el intervalo [A, B]. A, B y C son nùmeros naturales

3. Sabiendo que la temperatura en la escala Fahrenheit se puede calcular a partir de la correspondiente en la escala Celsius con la siguiente fórmula

   $F = (C * 9 / 5) + 32$

   mostrar en pantalla una tabla a dos columnas, una para la temperatura Fahrenheit y otra para la Celsius, con esta última variando de 40 a 100 y creciendo de a 10 grados

4. Mostrar en pantalla una tabla a dos columnas donde se vea la velocidad y la posición cada t segundos de un cuerpo en caída libre que parte desde una altura de h metros con una velocidad inicial v_0. Las fórmulas para el cálculo de posición y velocidad son las siguientes:

   $$x = x_0 + v_0 t + \frac{1}{2} g t^2$$

   $$v = v_0 + g t$$

   el último renglón de la tabla deberá mostrar los datos correspondientes a la altura de 0 metros

5. Dada una lista de N números mostrar en pantalla la suma, el promedio, el mayor y el menor

6. Mostrar en pantalla los N primeros números primos.

7. Mostrar en pantalla los números primos menores a N