CFIGM SISTEMAS MICROINFORMÁTICOS Y REDES
SISTEMAS OPERATIVOS MONOPUESTO
TEMA 3 - PRÁCTICA 02

Conceptos

1. Estructura condicional if-else.

Las estructuras condicionales *if-else* nos permiten definir en los programas sentencias, o bloques de código, que se ejecutarán sólo si una determinada condición se cumple. Dichas estructuras tienen el siguiente formato:

```
if(<condición>){
        codigoIF
}
else{
        codigoELSE
}
```

Llegado el punto de ejecución a una estructura *if-else* se evaluará la <*condición*> para comprobar si ésta es verdadera o falsa. En el caso de que la condición sea verdadera se ejecutará la sentencia, o bloque de código, *codigoIF*. Si se diese el caso de que la condición evaluada fuese falsa se ejecutará la sentencia, o bloque de código, *codigoELSE*. Una vez terminada la ejecución de esta estructura se procederá a ejecutar el código a continuación de la misma.

Se ha de remarcar que la parte *else* de este tipo de estructuras es opcional, por lo que si no es necesario que se ejecute ningún código cuando la condición evaluada sea falsa, puede y debe evitar la inclusión de esa parte de la estructura. Por tanto, en estos casos la estructura resultante seguiría el siguiente formato:

```
if(<condición>){
        codigoIF
}
```

Cuando *codigoIF* o *codigoELSE* sea una sola sentencia, no es necesario que ésta vaya encerrada entre llaves. En el caso de que alguno de los dos, o los dos, sean bloques de código (varias sentencias) éstos irán precedidos por una apertura de llave y cerrados por un cierre de llave. Se ha de tener en cuenta que ni la línea de la expresión *if*, ni la línea de la expresión *else* han de llevar punto y coma en ningún caso. El siguiente ejemplo ilustra los casos indicados anteriormente en los que se emplean sentencias y bloques de código:

```
if(a==0) {
        printf("a es igual a cero");
        a++;
}
else
        printf("a es distinto de cero");
```

2. Estructuras de control repetitivas while.

La estructura de control *while* se emplea para crear en los programas secciones de código que se repiten mientras una determinada condición sea verdadera. Esta clase de estructuras tienen la siguiente forma:

```
while(<condición>){
        codigoWHILE
}
```

Cuando el punto de la ejecución llega al inicio de la estructura *while* se procede a evaluar la <*condición*>. Si ésta resulta falsa, no ejecutará *codigoWHILE* y la ejecución seguirá por el código existente a continuación de la estructura *while*. Si dicha condición resulta verdadera, entonces se ejecutará la sentencia o bloque de código *codigoWHILE*. La particularidad de este tipo de estructura es la repetición del código que encierra, por tanto una vez ejecutada la sentencia, o bloque de código *codigoWHILE*, se volverá a evaluar la condición que contiene la estructura. Si dicha condición resulta verdadera se volverá a ejecutar desde el principio *codigoWHILE*. En caso de resultar falsa terminarán las repeticiones y la ejecución del programa continuará con el código situado a continuación de la estructura *while*.

Se ha de tener en cuenta que *codigoWHILE* puede ser un bloque de código (varias sentencias) o una única sentencia. En este último caso, las llaves son opcionales.

3. Estructuras de control repetitivas for.

Al igual que las estructuras *while*, las estructuras *for* se emplean para provocar la repetición de un determinado fragmento de código en los programas. La principal diferencia entre ambas estructuras es que las estructuras *for*, están especializadas en bucles de código que se repiten un número determinado de veces, estando dicho número de repeticiones controladas mediante un contador. La sintaxis de las estructuras *for* es la siguiente:

```
for(<inicialización>;<condición>;<actualización>)
       codigoFOR
```

Como puede observar, la estructura *for* declara tres secciones diferentes separadas por punto y coma. La sección <*inicialización*> se emplea para inicializar la variable contador que controlará el número de iteraciones del bucle. La sección <*condición*> está dedicada a contener la expresión lógica que representa la condición que ha de cumplirse para que se siga ejecutando el bucle, por lo que típicamente ésta suele ser una condición sobre el valor de la variable contador. La última sección, <*actualización*>, contiene el código para actualizar el valor de la variable contador, ya sea incrementando o decrementando su valor. La ejecución de una estructura *for* se realiza según los siguientes pasos:

1. Ejecución del bloque de la sección <*inicialización*>.

2. Evaluación de la <*condición*>. Si dicha expresión es verdadera:
   a. Ejecución de la sentencia, o bloque de código, *codigoFOR*.
   b. Ejecución de la sección <*actualización*>.
   c. Volver al punto 2.

3. Fin del bucle.

Experimentos

E1. Analice el siguiente código y ejecútelo paso a paso varias veces introduciendo en cada ocasión un número distinto (pruebe también con números negativos). Estudie por qué, en cada ocasión, se ejecutan o no ciertas partes del código.

```
#include<stdio.h>
void main(void){
     int n;
     printf("Introduce un numero\n");
     scanf("%d",&n);
     if(n<0){
        printf("%d el un número es negativo\n",n);
     }
     if(n%2==0)
        printf("%d es par",n);
     else
        printf("%d es impar",n);

       }
```

E2. Analice el siguiente código y descubra cuál es su objetivo. Ejecute paso a paso el código para descubrir cuál es su funcionamiento dependiendo del número introducido.

```
#include<stdio.h>

void main(void){
     int n;
     int i;

     printf("Introduce un numero\n");
     scanf("%d",&n);

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

E3. Analice y ejecute paso a paso el siguiente código. Observe que realiza una tarea similar al programa del experimento anterior cambiando el sentido de la cuenta.

```
#include<stdio.h>

void main(void){
     int n;
     int i;

     printf("Introduce un numero\n");
     scanf("%d",&n);

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

Compare el presente código con el del experimento anterior. ¿Qué código resulta más compacto y sencillo?

E4. Escribir una estructura de control que examine el valor de una variable real llamada *temp* y escriba uno de los siguientes mensajes dependiendo de su valor:

a) HIELO, si el valor de *temp* es menor que 0.
b) AGUA, si el valor de *temp* se encuentra entre 0 y 100
c) VAPOR, si el valor de *temp* es mayor que 100

Ejercicios

EJ1. Escriba un programa que pida al usuario un número. El programa indicará si este número es o no múltiplo de 2, 3, 5, 7 y 10.

Problemas

P1. Escriba un programa que pida al usuario una hora determinada, que denominaremos h_1, (pidiendo primero las horas expresadas en formado de 0 a 24 horas, y a continuación los minutos). Posteriormente se pedirá al usuario otra hora h_2. El programa deberá indicar si h_1 es posterior o anterior a h_2. En el caso de ser posterior se mostrará por pantalla un mensaje y se indicará cuantas horas y minutos han pasado, en caso de ser anterior se mostrará un mensaje similar y se indicará las horas y minutos que faltan.

P2. Escriba un programa que pida al usuario números hasta que éste proporcione el número cero. Una vez proporcionado dicho número el programa deberá de devolver por pantalla la media de los números introducidos anteriormente excluyendo al cero.

P3. Escriba un programa que muestre por pantalla los múltiplos de 5 hasta el número 1000.

P4. Escriba un programa que, dado un número entero imprima por pantalla los divisores del mismo.

P5. Escriba un programa que pida un número al usuario. Si el número introducido no es divisible por 2 y 3 entonces el programa mostrará un mensaje de error y volverá a pedir un número al usuario. En caso de que el número sea divisible por dichos números se mostrará el resultado de dividirlo por ellos y se terminará el programa.