CAPÍTULO 3

SENTENCIAS DE CONTROL DE FLUJO

Las sentencias de un programa se ejecutan por el ordenador de arriba hacia abajo y de forma consecutiva a no ser que, de alguna manera, se altere este orden. Los comandos que realizan esta función se llaman comandos de control de flujo y las sentencias a las que pertenecen sentencias de control de flujo.

3.1 Comandos condicionales

3.1.1 if simple.

Tiene la siguiente sintaxis:

if (expresión) sentencia

En donde expresión es cualquier expresión válida de C. Si expresión se evalúa como verdad se ejecutará la sentencia; si se evalúa como falso, se pasa por alto la sentencia y se ejecuta la línea de código que sigue al comando if.

Se recuerda que en C una expresión es falsa si su evaluación aporta el valor 0; en cualquier otro caso es verdadera. En la expresión que sigue a un if suelen aparecer operadores relacionales, de igualdad y lógicos.

Ejemplos:

El siguiente programa lee un entero por teclado y mediante dos *if simples* estudia si el número es múltiplo de 2 y/o de 3.
Estas dos condiciones no son excluyentes, es decir, se pueden cumplir a la vez. Cuando ocurra ésto se deben utilizar varios *if simples*.

Nótese la forma de escribir la condición número par:

num/2*2== num

(se utiliza la división entera que produce la pérdida de los decimales).

También se podría utilizar el operador resto (%)

num%2==0 ó !(num%2)

```
#include <stdio.h>
void main( )
{
 int num;
 printf("Introduce un entero\n");
 scanf("%d",&num);
 if (num/2*2 ==num)printf ("%d es multiplo de 2\n",num);
 if (num/3*3 ==num)printf ("%d es multiplo de 3\n",num);
}
```

Si se desea que se ejecuten varias sentencias si la expresión que sigue al if se evalúa como verdad, éstas se deben agrupar entre llaves:

```
if (expresión)
{
 sentencia_1;
 sentencia_2;
 sentencia_3;
    ..........
 sentencia_n;
 }
```

Volviendo al ejemplo anterior con alguna variación:

```
#include <stdio.h>
void main(  )
{
    int num;
    printf("Introduce un entero\n");
    scanf("%d",&num);
    if (num%2 ==0){
    printf ("%d es multiplo de 2\n",num);
    printf("es par\n");
    }
 if (num/3*3 ==num)printf ("%d es multiplo de 3\n",num);
}
```

3.1.2 Bloque **if** unicondicional.

También denominado *if dos ramas*, tiene por sintaxis:

```
if (expresión) sentencia_1;
else sentencia_2;
```

Si la expresión es evaluada como verdad se ejecuta la sentencia_1, si es falsa, se ejecuta la sentencia_2. En ningún caso se ejecutan las dos sentencias..

Si en cada bifurcación aparece un grupo de sentencias, éste debe ser englobado entre llaves.

El siguiente programa nos indica si el valor de un entero leído por teclado es par o impar:

```
#include <stdio.h>

void main( )
{
      int num;
      printf("Introduce un entero\n");
      scanf("%d",&num);
    if (num%2 ==0) printf ("%d es par\n",num);
      else printf ("%d es impar\n", num);
}
```

3.1.3 Bloque if multicondicional

También conocido como *if multirrama*, es un bloque de gran utilidad en el caso de problemas que dependen de muchas condiciones. Su sintaxis es la siguiente:

```
if (expresión_1) sentencia_1;
else if (expresión _2) sentencia_2;
else if (expresión _3) sentencia_3;
                        ……….
else sentencia_n;
```

En el esquema sintáctico anterior cada sentencia puede ser sustituida por un grupo de sentencias que deberán englobarse entre llaves. Su ejecución es como sigue:
el programa ejecutará la sentencia/s correspondiente a la primera expresión que se evalúe como verdadera y saldrá del if multicondicional. La sentencia **else** es opcional y se ejecutará por defecto cuando todas las condiciones anteriores sean falsas.

Ejemplo:

```
# include <stdio.h>
void main (  )
{
    int nota;
    printf("Introduzca la calificacion del alumno:\n");
    scanf("%d",&nota);
if(nota >= 0 && nota < 5) printf ("suspenso");
else if (nota>=5 && nota <=6) printf ("aprobado");
    else if (nota> 6 && nota <= 8) printf ("notable");
else if (nota > 8 && nota <= 10) printf ("sobresaliente");
else printf ("calificacion erronea");
}
```

3.2 La sentencia **switch**

Se utiliza para elegir un camino entre varios alternativos de forma semejante al *if multicondicional*. Su sintaxis es la siguiente:

```
switch (variable){
case valor_1:
 bloque de sentencias;
break;
case valor_2:
bloque de sentencias;
break;
                .
                .
                .
default:
 bloque de sentencias;
break;
}
```

La variable debe ser de tipo entero o carácter.

Se ejecuta de la siguiente forma: se compara el valor de la variable con los que aparecen en cada case; cuando concuerda con alguno, se ejecutan las sentencias que aparecen por debajo hasta encontrarse con un break o el fin del switch. Una vez que se entra en un case, no se sale del switch hasta que se encuentre con un break o el fin del switch, luego se puede pasar por varios case seguidos.

La porción default es opcional, se ejecuta cuando no hay ninguna coincidencia anterior.

El siguiente ejemplo utiliza la sentencia switch para imprimir en pantalla el nombre del número introducido por el usuario si está en el intervalo de 1 a 4.

```
# include <stdio.h>
void main ( )
{
   int i;
  printf("Introduzca un numero entre 1 y 4:\n");
   scanf("%d",&i);
   switch (i) {
   case 1:
       printf ("uno");
     break;
   case 2:
       printf ("dos");
     break;
   case 3:
       printf ("tres");
     break;
   case 4:
       printf ("cuatro");
     break;
   default:
       printf ("numero no reconocido");
    }
}
```

La secuencia de sentencias asociadas a cada case no son bloques, luego no están encerradas entre llaves.

Si se necesita que se ejecuten el mismo conjunto de sentencias para diferentes valores de la variable de estudio basta con actuar como en el siguiente ejemplo (semejante al que se vio con el if multicondicional):

```
# include <stdio.h>
void main (  )
{
 int nota;
 printf("Introduzca la calificacion del alumno: \n");
 scanf("%d",&nota);

 switch (nota) {
 case 0:
 case 1:
 case 2:
 case 3:
 case 4:
          printf ("suspenso");
          break;
 case 5:
 case 6:
          printf ("aprobado");
          break;
 case 7:
 case 8:
          printf ("notable");
          break;
 case 9:
      case 10:
          printf ("sobresaliente");
          break;
 default:
          printf ("calificacion erronea");
 }
}
```

Una vez que se encuentra un punto de entrada se ejecutarán todas las sentencias hasta el primer break o el fin del switch.

Es posible encontrar un switch como parte de la secuencia de sentencias de un switch más externo. Se denomina switch anidado. Ejemplo:

```
switch(a){
        case 1:
                switch (b){
                        case 0: printf("b es falso y a es cierto");
                                break;
                    case 1: printf("b es cierto y a es cierto");
                        }
                        break;
        case 2:
        ..........
.....................
```

3.3 Comandos repetititivos

3.3.1 El bucle while

Además del bucle for, visto en el capítulo anterior, C tiene otros comandos repetitivos. El primero que se va a estudiar es el bucle while. Sintaxis:

while (expresión) sentencia

Funciona ejecutando la sentencia mientras la expresión sea evaluada como verdad, cuando ésta es falsa el bucle se detiene. El valor de la expresión se comprueba al principio del bucle, por lo que si es falsa al iniciarse el bucle, éste no se ejecutará ni siquiera una vez.

El bucle puede repetir un grupo de sentencias; en ese caso se deben englobar entre llaves.

Ejemplo:

```
# include <stdio.h>
void main ( )
{
  int contador;
  contador=0;
  while (contador<6){
 printf ("el valor del contador es %d\n", contador);
    contador=contador+1;
  }
 }
```

Cuando el valor de la variable contador (que comienza en 0 y se va incrementando en 1) sea 6, la expresión es falsa y el bucle deja de ejecutarse continuando con la línea de código inmediatamente posterior a la finalización del bucle.

El bucle while del ejemplo anterior es equivalente al siguiente for:

```
for (contador=0; contador<6; contador++)
        printf ("el valor del contador es %d\n", contador);
```

En el siguiente ejemplo se utiliza el bucle while para validar un dato de entrada. Mientras el dato aportado por el usuario no sea válido se volverá a repetir la petición del dato.

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

void main ( )
{
        int a,b;
        char ch;
        printf ("¿Qué operación quiere realizar?: ");
       printf ("\nSuma \nResta \nMultiplicación \n División \n");
        printf ("Introduzca la primera letra (S, R, M o D)\n");
        ch=getche( );
```

```
        while (ch!='S' && ch!= 'R' && ch != 'M' && ch != 'D'){
                printf ("Letra errónea. Introduzca de nuevo:\n");
                ch=getche();
        }
        printf ("Introduzca los dos números:\n");
        scanf("%d %d",&a,&b);
        printf("El resultado de la operación es:  ");
        if (ch == 'S')
                printf("%d", a+b);
        else if (ch == 'R')
                printf("%d", a-b);
        else if (ch == 'M')
                printf("%d", a*b);
        else if (ch == 'D')
                printf("%d", a/b);
}
```

3.3.2 El bucle do while

Su sintaxis es como sigue:

```
do {
    sentencias;
}while (expresión);
```

Si sólo se va a repetir una sentencia no es necesario encerrarla entre llaves; pero es práctica habitual hacerlo para distinguir claramente que el while que aparece es parte del bucle do y no el principio de un bucle while.

El bucle do repite la sentencia o sentencias mientras la expresión sea verdadera; se detiene cuando la expresión se hace falsa.
La condición se comprueba al final del bucle, luego, el grupo de sentencias se ejecutan al menos una vez. Esta característica hace que el bucle do sea adecuado cuando se quiera validar datos introducidos por el usuario, ya que estos datos se deben introducir al menos una vez.
Veamos como cambia el ejemplo del apartado anterior utilizando el bucle do en lugar del while.

```
# include <stdio.h>
# include <conio.h>
void main ( )
{
        int a,b;
        char ch;
        printf ("¿Qué operación quiere realizar?:\n Suma\n Resta\n
Multiplicación\n División\n");

        do {
                printf ("\nIntroduzca la primera letra (S, R, M o D)");
                ch=getche();}
        while (ch!='S' && ch!= 'R' && ch != 'M' && ch != 'D');

        printf ("\nIntroduzca los dos números:\n");
        scanf("%d %d",&a,&b);
        printf("El resultado de la operación es:  ");
        if (ch == 'S')
                printf("%d", a+b);
        else if (ch == 'R')
                printf("%d", a-b);
        else if (ch == 'M')
                printf("%d", a*b);
        else if (ch == 'D')
                printf("%d", a/b);
}
```

En este caso, no hay que pedir el dato al usuario antes de entrar en el bucle ya que nos aseguramos una primera lectura al ejecutarse do al menos una vez.

3.4 La sentencia break

La sentencia break se puede utilizar con cualquiera de los tres bucles de C. Permite salir del bucle desde cualquier punto de su cuerpo. Cuando se ejecute el comando break, el bucle termina inmediatamente y el programa continúa en la sentencia que sigue al bucle. Por ejemplo, este programa imprime los números del 1 al 10.

```
#include <stdio.h>
void main(  )
{
   int i;
   for(i=1;i<100;i++)
       {
            if (i==11)break;
            printf("%d\n",i);
   }
}
```

Cuando i toma el valor 11, la condición del if es cierta y se ejecuta el break; en ese momento salimos del bucle (no se imprime el valor 11). Si no se hubiera escrito la línea de comando break, se imprimirían los números del 1 al 99.

Se ha estudiado en un apartado anterior que el comando break también se puede utilizar en la sentencia switch para salir de ella una vez que se ha entrado en uno de sus casos.

3. 5 La sentencia continue

Esta sentencia también se utiliza dentro de cualquiera de los tres bucles de C.

La sentencia continue envía el flujo del programa a la expresión de prueba del bucle en el que se encuentre, dejándose de ejecutar, en esa iteración, las líneas de comando que se encuentren entre continue y el final del bucle.

En el siguiente ejemplo, se ha realizado un programa semejante al del apartado anterior empleando el comando continue en lugar del break.

```
#include <stdio.h>
 void main( )
  {
          int i;
          for(i=1;i<100;i++){
                  if (i==10)continue;
                  printf("%d\n",i);
           }
  }
```

El bucle comienza con el valor 1 de la variable de control, que se va incrementando en una unidad en cada iteración imprimiéndose en pantalla. Cuando i vale 10 se cumple la condición del if y, por tanto, se ejecuta el continue, que envía el flujo del programa al inicio del bucle sin ejecutarse el printf correspondiente (el incremento de la variable de control sí se ejecuta, antes que la comprobación de la condición del bucle). Debido a esto, el valor 10 no aparece en pantalla. El siguiente valor que toma i es 11, que se imprimirá, y así sigue el proceso hasta el valor 99.

Tal como se ha dicho, en los bucles while y do while, una sentencia continue también hará que el control vaya directamente a la expresión y que continúe después el proceso del bucle.

En el siguiente programa se introducen las notas de 10 alumnos, si alguna no es correcta (menor que 0 o mayor que 10) se volverá a pedir esa nota. Además se calcula la nota media.

```
#include <stdio.h>
 void main(  )
{
   float notas[10],media=0.;
   int i=0;
   do {
 printf("Introduce la nota del alumno %d\n", i+1);
   scanf("%f",&notas[i]);
   if (notas[i]>10 || notas[i]<0)continue;
   media=media+notas[i];
   i=i+1;
   }while(i<10);
   printf("La nota media es %f",media/10);
}
```

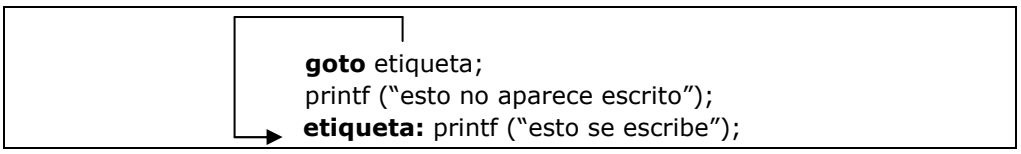
3.6 La sentencia goto

C aporta una sentencia de salto no condicional llamada **goto**.

La utilización de goto rompe la estructura del programa y, si se usa con mucha frecuencia, puede hacer que el programa sea muy difícil de entender. Por estas razones no se empleará fuera de este apartado del libro.

Se utiliza escribiendo la palabra reservada goto seguida de un nombre. El nombre seguido por dos puntos (**:**) se pone en el lugar del programa al que se quiera enviar el control. Al nombre que va delante de los dos puntos le denominamos **etiqueta**.

Ejemplo:



```
goto etiqueta;
printf ("esto no aparece escrito");
etiqueta: printf ("esto se escribe");
```

En el siguiente ejemplo se utiliza la sentencia goto para crear el equivalente a un bucle for que se ejecuta de 1 a 10:

```
# include <stdio.h>
void main ( )
{
   int i=1;
   repito:
   printf("%d ",i);
   i++;
   if(i<=10) goto repito;
}
```

No está permitido usar goto para saltar de una función a otra, ni al interior de un bucle. Sin embargo, se puede saltar fuera de los bucles.

3.7 La función exit( )

Su sintaxis es

void **exit** (int estado)

La utilización de la función exit(estado) da lugar a la terminación anticipada de un programa. Si el valor de estado es 0 se supone que se ha producido una terminación normal del programa, un valor distinto de 0 puede utilizarse para indicar un error. Se encuentra dentro de la librería <stdlib.h>.

El siguiente programa provoca la finalización anticipada cuando el dato introducido se considera no válido.

```
# include <stdio.h>
# include <stdlib.h>
void main ( )
{
 float nota1, nota2, nota3, media;
  printf("Introduce las tres notas del alumno\n");
  scanf("%f %f %f", &nota1, &nota2, &nota3);
 if (nota1 <0 || nota1 > 10 ||nota2 <0 || nota2 > 10 ||nota3 <0 || nota3 > 10)
        {
                printf("Datos erroneos");
                exit(0);}
        media=(nota1+nota2+nota3)/3;
        printf ("La nota media es: %f", media);
}
```

3.8 Enunciados de las prácticas

SENTENCIAS CONDICIONALES

Ejercicio 1. IF simple

A continuación aparecen dos programas C. Determinar en cada caso las posibles salidas a pantalla y las diferencias sustanciales que hay entre ellos:

1.a)

```
# include <stdio.h>
void main( )
{
char respuesta;
printf("Quieres saber quien soy?:afirmativo: S, negativo: cualquier caracter\n");
scanf("%c",&respuesta);
if (respuesta=='S')
        printf ("Soy tu ordenador y entiendo lenguaje C\n");
        printf("Hasta otra\n");
}
```

1.b)

```
# include <stdio.h>
void main( )
{
char respuesta;
printf("Quieres saber quien soy?: afirmativo: S, negativo: cualquier
caracter\n");
scanf("%c",&respuesta);
if (respuesta=='S')
        {
        printf ("Soy tu ordenador y entiendo lenguaje C\n");
        printf("Hasta otra\n");
        }

}
```

Ejercicio 2. IF de dos ramas. IF anidados

De igual forma que en el ejercicio anterior, estudiar los dos programas propuestos determinando la salida a pantalla y las diferencias entre ambos.

2.a)

```
#include <stdio.h>
void main( )
 {
  int lluvia,ventana;

  printf("Llueve? (0 = NO, 1 = SI): ");
  scanf("%d", &lluvia);
  printf("Esta la ventana abierta? (0 = NO, 1 = SI ): ");
  scanf("%d", &ventana);

  if (lluvia == 1)
{
        if (ventana == 1) puts("  Cierra la ventana y quedate en casa");
 }
  else
        puts("Puedes salir");
 }
```

2.b)

```
#include <stdio.h>
void main( )
{
  int lluvia,ventana;

  printf("Llueve? (0 = NO, 1 = SI): ");
  scanf("%d", &lluvia);
  printf("Esta la ventana abierta? (0 = NO, 1 = SI ): ");
  scanf("%d", &ventana);

  if (lluvia == 1)
          if (ventana == 1) puts("  Cierra la ventana y quedate en casa");
  else
          puts("Puedes salir");
 }
```

Ejercicio 3. IF de dos ramas

Utilizar la estructura condicional `if /else` para calcular el menor valor de una pareja de números enteros leídos por teclado.

Ejercicio 4. IF simple, IF de dos ramas, IF multirrama

Escribir un programa C que lea un número entero por teclado y haga lo siguiente:

Si la última cifra del número es impar, escriba en pantalla el mensaje
El número xxxxxxx es impar
Además, si esta cifra es 5 entonces escriba
El número xxxxxxx es múltiplo de 5
Y si no es 5 escriba
Me falta información
Si la última cifra del número es 0 escriba en pantalla
El número xxxxxxx es múltiplo de 2 y 5
Si además, la anteúltima cifra del número es 0 escriba
El número xxxxxxx es múltiplo de 4
Si la última cifra es par y distinta de 0 escriba
Sólo podemos asegurar que el número xxxxxxx es par

Indicación: es útil extraer la última y la anteúltima cifra del número y almacenarlas en las variables *nult* y *anteult*.

Ejercicio 5. Sentencia `switch`

Diseñar un programa C que pida al usuario dos números reales para realizar una operación entre ellos. A continuación le ofrezca el siguiente menú con las operaciones disponibles:
OPERACIONES DISPONIBLES:
+ : SUMA
- : RESTA
*: PRODUCTO
/: DIVISIÓN

El usuario debe introducir uno de los cuatro caracteres y el programa le ofrecerá como respuesta (ejemplo):

El resultado de la operación 3. + 5. es 8.

Introducir además las sentencias adecuadas para que si se va a realizar una operación no válida (división entre 0) se vuelvan a pedir los datos de los operandos todas las veces que sea necesario.

COMANDOS REPETITIVOS

Ejercicio 6. Bucle for

Sea el siguiente programa C, averiguar cuántas veces se ejecuta el bucle y determinar la salida a pantalla, en especial entender el valor de la variable *i*:

```
#include <stdio.h>
 void main(  )
      {
        int i,j=-2;
        for(i=5;i<=19;i+=3)
           j+=i;
        printf("\n j= %d  i= %d",j,i);
      }
```

Ejercicio 7.

Diseñar un programa C que mediante un bucle for calcule el sumatorio de 1 hasta n, y el factorial de n; siendo n un entero leído por teclado.

Ejercicio 8. Bucle for

Escribir la salida del siguiente programa C. ¿Qué significado tiene la variable *Pr*?

```
#include <stdio.h>
void main( )
{
      int j;
      float a[3]={1.,0.,-3.},b[ ]={3.,4,6.};
      float Pr=0.;
      for (j=0;j<3;j++) Pr+=a[j]*b[j];
      printf("El resultado de xxxxxxx es %f",Pr);
}
```

Ejercicio 9. Bucle for

Escribir un programa C que imprima en pantalla los números impares menores que 100.

Ejercicio 10. Bucle while / do ...while

Resolver el mismo problema del ejercicio anterior pero ahora utilizando la estructura repetitiva while y, por otro lado, la estructura do ... while.

Ejercicio 11. Bucle for y sentencia continue

Resolver el mismo problema empleando de nuevo el bucle for, pero evitando que se escriban los números pares mediante la utilización de la sentencia continue.

Ejercicio 12. Bucle for

Utilizar la estructura repetitiva for para imprimir en pantalla las potencias 1,2,3 y 4 de los números 1 al 10.

Ejercicio 13. Bucle for, while, do ... while

Diseñar un programa C que lea un número entero y calcule el número de dígitos del mismo; el programa debe producir la siguiente salida:

```
El número de dígitos del número xxxx  es zz
```

Indicación: en primer lugar lee el número entero, y hace un bucle desde que número de dígitos es 1 hasta que el número dividido sucesivamente entre diez sea 0, incrementándose el número de dígitos en cada pasada del bucle.

Resolver el mismo problema con un bucle while y do ... while

Ejercicio 14. Bucle `for` con varios índices

El comando `for` admite una sintaxis más general, de forma que es posible usar dos o más índices simultáneamente, separados por comas (operador coma). Veamos el siguiente ejemplo:

```
for(i=1,j=1;i<4&&j<4;i++,j++)
        printf(" %d  %d ",i,j);
```

Este bucle imprimiría:
1 1 2 2 3 3

Utilizar esta técnica para obtener una solución muy simple del siguiente problema:

Escribir los m primeros términos de la siguiente sucesión, siendo m un número entero leído del teclado.

1,1,4,2,7,3,10,4,13,5,16,6,19,7,22,8,25,9,28,10 .....

Ejercicio 15.

Escribir un programa C que imprima en pantalla la siguiente sucesión de números:

3,8,15,24,35,48.....

menores que 1000.

Nota: Resolver este problema utilizando distintas estructuras de control.

Ejercicio 16. Uso de las sentencias **break** y **continue**

Determinar la salida a pantalla del siguiente programa C

```
#include <stdio.h>
void main( )
{
   int m=0,l=0,i,j,k;
    for (i=1;i<=2;i++)
     {
         for (j=1;j<=4;j++)
         {
         k=4*j+10+m;
          if(k>25)continue;
          l=(m+3)/2;
          m++;
          printf("\n %d %d %d %d %d",i,j,k,l,m);
          }
      l-=2;
      if(l<0)break;
      printf("\n %d %d %d %d %d",i,j,k,l,m);
     }
}
```

Ejercicio 17.

Diseñar un programa C que lea por pantalla dos números enteros (i,j), de forma que si i no es menor igual que j, se intercambien los valores hasta determinar un intervalo, es decir, hasta que i<=j. A continuación aparece un mensaje en pantalla

Introduce un tercer número

y el programa debe leerlo de forma que si éste pertenece al intervalo el programa finaliza sin hacer nada, y en caso contrario vuelve a aparecer el mensaje y pedir otro número (así indefinidamente).

Se pueden probar varias soluciones a este problema.

Ejercicio 18. Cálculo de máximos y mínimos

Sea un vector *v* entero con un número indeterminado de componentes (menor que 100). Se pide diseñar un programa C que lea por teclado el número de componentes del vector y el valor de cada una de ellas y calcule el valor máximo y mínimo de las componentes del vector y en qué posición están situadas.

Nota: Cuando inicialmente no se conoce el número de componentes de una tabla, ya que ese dato lo proporcionará el usuario, con los conocimientos adquiridos hasta el momento, únicamente se puede sobredimensionar la tabla en la declaración. En el capítulo 6 (punteros) se estudiará una técnica (dimensionamiento dinámico) que permite dimensionar la tabla de forma exacta al tamaño real que tendrá en la ejecución del programa.

3.9 Soluciones a las prácticas del capítulo 3

SENTENCIAS CONDICIONALES

Ejercicio 1. if simple

Si el if *simple* afecta sólo a una sentencia no es necesario encerrarla entre llaves. Si se desea que dentro del if haya un grupo de sentencias obligatoriamente se deben disponer llaves.

En el primer programa no hay llaves que afecten al comando if, por tanto, sólo la primera sentencia de escritura está afectada por la condición. Si el usuario teclea un carácter distinto de ‘S’ aparece la frase "Hasta otra"; si el usuario teclea ‘S’ además aparece la frase "Soy tu ordenador y entiendo lenguaje C".

En el segundo programa, las dos sentencias de escritura están dentro del if. Por tanto, si el usuario no introduce ‘S’ no aparecerá nada escrito; y si introduce ‘S’ aparecen las dos frases.

Ejercicio 2. Bloque if unicondicional

El comando else va a pertenecer siempre al if inmediatamente anterior, a no ser que éste ya haya sido cerrado entre llaves.

2.a) En este caso, el segundo if ha sido cerrado antes de comenzar el else, luego este else pertenece al primer if.
Si la respuesta a la primera pregunta es 0, independientemente de la respuesta a la segunda cuestión, aparece el mensaje:
"Puedes salir"
Si la respuesta a la primera pregunta es 1, entonces:
Si la respuesta a la segunda pregunta es 0, no aparece ningún mensaje
Si la respuesta es 1, aparece el mensaje
"Cierra la ventana y quedate en casa"

2.b) En este caso, el segundo if no ha sido cerrado y por tanto, el else hace referencia a él.

Si la respuesta a la primera pregunta es 0, no aparece ningún mensaje.
Si la respuesta a la primera pregunta es 1, entonces:
Si la respuesta a la segunda pregunta es 1, aparece el mensaje
"Cierra la ventana y quedate en casa"
Si la respuesta es 0, aparece el mensaje
"Puedes salir"

Ejercicio 3. Bloque if unicondicional

Un posible programa es el siguiente:

```
#include <stdio.h>

void main(  )
{
   int elmenor;            /* el resultado */
   int n1, n2;          /* los valores de entrada */
   /*
     Pide al usuario dos números y los lee
    */
   printf("Introduce los numeros ");
   scanf("%d %d", &n1,&n2);

   if (n1 < n2)
      elmenor = n1;
   else
      elmenor = n2;
```

```
   printf("El menor de los numeros %d y %d es %d.\n",
      n1, n2, elmenor);
}
```

Ejercicio 4. if simple, bloque if unicondicional, bloque if multicondicional

Una posible solución es la siguiente:

```
#include <stdio.h>

void main(  )
{

        int n, nult,anteult;

        printf("Escribe un entero\n");
        scanf("%d",&n);
        nult=n-n/10*10;
        anteult=(n-n/100*100)/10;

        if (nult%2!=0)
                {
                printf("El numero %d es impar\n",n);
                if (nult==5) printf("El numero %d es multiplo de 5\n",n);
                else printf("Me falta informacion\n");
                }
        else if (nult ==0)
                {
                printf("El numero %d es multiplo de 2 y 5\n",n);
                if (anteult==0) printf("El numero %d es multiplo de 4\n",n);
                }
        else printf("Solo podemos asegurar que el numero %d es par\n",n);
}
```

Ejercicio 5. Sentencia **switch**

Una solución válida es la siguiente:

```
#include <stdio.h>

void main(  )
{
        float n1,n2,resul;
        char op;

        printf("OPERACIONES DISPONIBLES:\n");
        printf("+: SUMA \n -: RESTA \n *:PRODUCTO \n /: DIVISION\n");
        printf("Introduce simbolo del operador\n");
        scanf("%c",&op);
        printf("Introduce los operandos\n");
        scanf("%f %f", &n1,&n2);
        while (n2==0 && op == '/')
        {
             printf("ATENCION division entre 0. Reintroducir los operandos\n");
             scanf("%f %f", &n1,&n2);
        }

        switch (op)
        {
        case '+':
                        resul=n1 + n2;
                        break;
        case '-':
                        resul=n1 - n2;
                        break;
        case '*':
                        resul=n1 * n2;
                        break;
        case '/':
                        resul=n1 / n2;
                        break;
        }
        printf("El resultado de %f %c %f es %f\n",n1,op,n2,resul);
}
```

COMANDOS REPETITIVOS

Ejercicio 6. Bucle for

La salida a pantalla es la indicada a continuación:

j= 53 i= 20

El bucle se realiza cinco veces. En el inicio del ciclo número 6, i toma el valor 20, no se cumple la condición de repetición y por tanto se sale del bucle continuando el programa en la línea inferior a éste.

Ejercicio 7.

Un posible programa es el siguiente:

```
#include <stdio.h>
void main(  )
{

        int n,sum=0,fact=1,i;
        printf("Escribe el entero para calcular el sumatorio y el factorial\n");
        scanf("%d",&n);
        for(i=1;i<=n ;i++)
        {
                sum=sum+i;
                fact=fact*i;
        }
        printf("El sumatorio de %d es %d\n",n,sum);
        printf("El factorial de %d es %d\n",n,fact);
}
```

Ejercicio 8. Bucle for

La salida a pantalla es la siguiente:

El resultado de xxxxxxx es -15.000000

El bucle almacena en Pr el valor de a[0]*b[0]+ a[1]*b[1]+ a[2]*b[2] es decir el producto escalar de los dos vectores.

Ejercicio 9. Bucle for

Una solución válida es la siguiente:

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

      for (i=1;i<100;i+=2) printf("%d ",i);
}
```

Ejercicio 10. Bucle while / do ... while

A continuación se ofrece un programa válido:

a) Con el bucle while

```
#include <stdio.h>
void main(  )
{
      int i=1;
      while (i<100)
        {
              printf("%d ",i);
               i=i+2;
        }
}
```

b) Con el bucle do ... while

```
#include <stdio.h>
void main(  )
{
      int i=1;
      do {
             printf("%d ",i);
              i=i+2;}
      while (i<100);
}
```

Ejercicio 11. Bucle for y sentencia continue

Un posible programa es el siguiente:

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

```
void main(  )
{
      int i;

      for(i=1;i<100;i++)
                {
                        if(i%2==0)continue;
                        printf("%d ",i);
                }
}
```

Ejercicio 12. Bucle for

Una solución válida es la siguiente:

```
#include<stdio.h>

void main(  )
{
 int n;

 printf(" N\tN2\tN3\tN4 \n");
 for (n = 1; n <= 10; n++)
    printf(" %d \t %d \t %d \t %d\n",  n, n * n,  n * n * n,  n * n * n * n);
}
```

Ejercicio 13. Bucle for, while y do ... while

Un posible programa es el siguiente:

```
#include<stdio.h>

void main(  )
{
    int n, numdig,n2;
    printf("Introduce un entero\n");
    scanf("%d",&n);
    n2=n;

    n=n/10;
    for (numdig = 1;n!=0; numdig++) n=n/10 ;

/* las dos sentencias anteriores son equivalentes a
```

```
            for (numdig = 1;(n=n/10)!=0; numdig++)  ;
*/
    printf("El numero de digitos del numero %d es %d",n2, numdig);
}
```

El mismo programa utilizando un bucle `while`

```
#include<stdio.h>
void main(  )
{
     int n, numdig=1,n2;
     printf("Introduce un entero\n");
     scanf("%d",&n);
     n2=n;

     while ((n=n/10)!=0) numdig=numdig+1;
     printf("El numero de digitos del numero %d es %d",n2, numdig);
}
```

Y con un bucle `do ... while`:

```
#include<stdio.h>

void main(  )
{
     int n, numdig=0,n2;
     printf("Introduce un entero\n");
     scanf("%d",&n);
     n2=n;
     do {
                n=n/10;
                numdig=numdig+1;
         }
     while (n!=0) ;
     printf("El numero de digitos del numero %d es %d",n2, numdig);
}
```

Ejercicio 14. Bucle `for` con varios índices

Una posible solución es la siguiente:

```
#include <stdio.h>

void main(  )
{
     int i,j,m,mitad;
```

```
    printf("Introducir numero de terminos ");
    scanf("%d",&m);
    mitad=m/2;

    for(i=1,j=1;mitad-->0;i+=3,j++)
                        printf(" %d  %d ",i,j);
    if(m/2*2!=m)printf(" %d ",i);
}
```

En este ejemplo se ha usado el operador coma para emplear i y j como índices del bucle, mientras que en la condición del bucle no se emplean dichos índices, sino el valor de mitad (número de términos dividido por 2); se ha hecho uso, un vez más, del operador decremento, de forma que mitad tras cada iteración del bucle almacena un número menor.

Observar que la única instrucción del bucle (ya que no hay llaves), imprime dos términos de la sucesión en cada ciclo; se ha tenido que utilizar una variable auxiliar mitad que va a almacenar la mitad (división entera) de m (número de términos). Si el número de términos es impar, se debe añadir otro término más a la sucesión, de ahí la sentencia:

```
if(m/2*2!=m)printf(" %d ",i);
```

En este caso, puesto que en cada ciclo del bucle se realizaba la operación i+=3, el valor de i quedaba incrementado, por lo que para el último término impar basta con escribir i.

Ejercicio 15.

Una solución válida es la siguiente:

```
#include <stdio.h>

void main(  )
{
    int n=3,i=5;

    while (n<1000)
        {
                printf(" %d",n);
                n=n+i;
                i=i+2;
        }
}
```

Ejercicio 16. Uso de las sentencias `break` y `continue`

La salida a pantalla es la indicada a continuación:

```
1  1  14  1  1
1  2  19  2  2
1  3  24  2  3
1  5  29  0  3
2  1  17  3  4
2  2  22  3  5
2  5  31  1  5
```

Ejercicio 17.

Una posible solución es la siguiente:

```
#include <stdio.h>
void main(  )
{
        int i,j,k,aux;
        printf("Escribe dos enteros\n");
        scanf("%d %d",&i, &j);
        if(i>j)
        {
                aux=i;
                i=j;
                j=aux;
        }
        do {
                printf("Introduce un tercer numero\n");
                scanf("%d",&k);
        }while(k<i ||k>j);
        printf("El numero %d pertenece al intervalo [%d,%d]",k,i,j);
}
```

Se puede llegar a otra solución utilizando la sentencia `break`:

```
while (1) {
                printf("Introduce un tercer numero\n");
                scanf("%d",&k);
                if(k>=i && k<=j)break;
        }
```

En este caso, el bucle while es infinito ya que su condición siempre es verdad (valor 1). Se interrumpe el bucle ejecutando el comando break cuando el número k pertenezca al intervalo.

Ejercicio 18. Cálculo de máximos y mínimos

Un programa válido es el siguiente:

```
#include <stdio.h>

void main( )
{
   int v[100],i,n,maxi,mini,k,t;

   printf("Introduce el numero de componentes del vector\n");
   scanf("%d", &n);
   for (i=0;i<n;i++)
   {
      printf("Introduce la componente %d ",i+1);
      scanf("%d", &v[i]);
  }
  maxi=mini=v[0];

  for(i=1;i<n;i++)
  {
        if (v[i]>maxi)
        {
                maxi=v[i];
                k=i;
        }
        if (v[i]<mini)
        {
                mini=v[i];
                t=i;
        }
 }
 printf("El maximo es %d y esta en la posicion %d\n",maxi,k +1);
 printf("El minimo es %d y esta en la posicion %d\n",mini,t +1);

}
```