Control De Flujo De Programa

Sentencias Condicionales Y Bucles

Hasta ahora los programas que hemos visto eran lineales. O sea que las instrucciones se ejecutaban en orden y una sola vez. Pero resulta que muchas veces no es esto lo que queremos que ocurra. Lo que nos suele interesar es que dependiendo de los valores de los datos se ejecuten unas instrucciones y no otras. O también puede que queramos repetir unas instrucciones un número determinado de veces. Para esto están las sentencias de control de flujo.

Bucles

Los bucles nos ofrecen la solución cuando queremos repetir una tarea un número determinado de veces. Supongamos que queremos escribir 100 veces la palabra Bienvenido.
Con lo que sabemos, tendríamos que escribir 100 veces la expresión: `printf ("Bienvenido\n");` , sin embargo un ciclo for nos facilitaría en grande esta tarea. Estos bucles pueden ser: ***for*** (para), ***while*** (mientras), ***do while*** (hacer-mientras).

*El bucle **for***
Este bucle nos permite hacer un número finito de tareas, cuantas veces necesitemos. La sintaxis del **for** es la siguiente:

```
for ([valor inical] ; [condición] ; [incremento])
{
    // ...sentencias
    // ...sentencias
}
```

Ejemplo:

```
for (i = 0 ; i < 10 ; i++)
{
  printf ("Mostrando el uso de for... \n");
  printf ("Pasado nro %d", i + 1);
}
```

Recibe tres parámetros: El valor inicial con el que empieza la variable contadora, o la variable que nos va a decir cuando salir. La condición, es la que nos determinará el número de veces que se llevarán a cabo las instrucciones que siguen. Estas sentencias se realizarán mientras la condición estipulada se cumpla. El incremento, es el número en que vamos a incrementar la variable contadora.

*El bucle **while***

La sintaxis del bucle **while** es la siguiente:

```
while ( condición )
{
    // ... bloque de instrucciones a ejecutar
}
```

Ejemplo:

```
int i=0;
while (i < 8)
{
  printf ("Mostrando el uso de for... \n");
  printf ("Pasado nro %d", i + 1);
  i++;
}
```

La condición de este bucle, al igual que en el for, es la que indica cuantas veces se ejecutarán las instrucciones. Estas (las instrucciones) se ejecutarán mientras la condición estipulada se cumpla.

*El bucle **do - while***

La sintaxis de este bucle es:

```
do {
   // instrucciones a ejecutar
} while (condición) ;
```

Ejemplo:

```
char seguir;
do
{
   printf("Probando la ejecución del do while \n";
   printf("¿Desea continuar? Ingrese 's' o 'n' \n");
   fflush(stdin);
   scanf( "%c", &seguir);
} while ( seguir != 'n') ;
```

En este bucle, las instrucciones se llevan a cabo al menos una vez, puesto que la condición se evalúa al final del mismo; a diferencia del while o el for, en los que si la condición inicial no se cumple, entonces las instrucciones no se ejecutan ni una sola vez. La condición es similar a las antes descriptas.

Sentencias Condicionales

Hasta aquí hemos visto cómo podemos repetir un conjunto de instrucciones las veces que deseemos. Pero ahora vamos a ver cómo podemos controlar totalmente el flujo de un programa. Dependiendo de los valores de alguna variable se tomarán unas acciones u otras.

*Sentencia **if***
La palabra if significa si (condicional), su sintaxis es como sigue:

```
if ( condición )
{
   // instrucciones a ejecutar
}
```

Cuando se cumple la condición comprendida entre los paréntesis se ejecuta el bloque de sentencias inmediatamente siguiente al if.

Ejemplo:

```
int i = 0;
printf("Ingrese un valor entero: ");
scanf("%d",&i);
if (i < 10) {
   print("El valor ingresado es mayor es menor a 10");
}
```

*Sentencia **if-else***
En el if, si la condición no se cumplía, entonces las instrucciones no se ejecutaban y el programa finalizaba o seguía con las siguientes instrucciones. En el if-else, (si-sino (en pseudocódigo)) si no se cumple la condición entonces se realizarán otras instrucciones (las del else).

Su sintaxis es la siguiente:

```
if ( condición ) {
  // bloque que se ejecuta si se cumple la condición
} else {
  // bloque que se ejecuta si no se cumple la condición
}
```

Por ejemplo:

```c
int main( ) {
  int a ;
  printf ( "Introduzca un número: " ) ;
  scanf ("%d", &a) ;
  if ( a == 8 ) {
    printf ( "El número que introdujo es ocho.\n" ) ;
  } else {
    printf ( "El número que introdujo no es ocho\n" ) ;
  }
  return 0;
}
```

En este ejemplo se solicita un número, si el número introducido es ocho, entonces manda un mensaje positivo, mientras que si el número no es ocho manda un mensaje negativo.

Se pueden agrupar varios if, o sea que si se cumple uno, luego se pregunta otra condición y así sucesivamente. A estos se los llama “if anidados o en cascada”. A la vez es posible analizar más de una condición siempre y cuando estas vayan ligadas con su respectivo operador. Por ejemplo:

```c
int main( ) {
  int a ;
  printf ("Introduce un número: ") ;
  scanf ("%d", &a) ;
  if ( a < 10 ) {
    printf ("El número introducido es menor de 10.\n") ;
  } else if (a > 10 && a < 100) {
    printf ("El número está entre 10 y 100"\n") ;
  } else if (a > 100) {
    printf ("El número es mayor que 100\n") ;
  } else {
    printf ("No está dentro de intervalos especificados") ;
  }
  return 0;
}
```

En este ejemplo lo que se hace es pedir un número, se analiza una condición y si se cumple se manda un mensaje, si no, se analiza otra condición y se manda a imprimir un mensaje para cada caso, si no, se imprime un mensaje negativo.

*Sentencia **switch***

La sentencia switch, analiza una condición y dependiendo del valor escoge entre varias opciones para realizar distintas instrucciones y si la opción no está establecida, simplemente ejecuta una opción por defecto.

La sintaxis para el switch es:

```
switch ( variable ) {
  case [opción 1]:
    //código a ejecutar si la variable tiene el valor de la opción 1 ;
    break ;
  case [opción 2]:
    //código a ejecutar si la variable tiene el valor de la opción 2 ;
    break ;
  case [opción n]:
    //código a ejecutar si la variable tiene el valor de la opción n ;
    break ;
  default:
    //código que se ejecuta si la variable tiene un valor distinto a
    //los anteriores ;
}
```

Observemos, el switch analiza el valor de "`variable`" luego si variable tiene el valor "`opcion` 1" entonces se ejecutan las instrucciones que siguen, y así sucesivamente. Es importante poner al final de cada grupo de instrucciones la sentencia "`break ;`", la que indica al programa que ahí terminan las sentencias de cada opción. Se pueden poner cuantas opciones se desee y al final es recomendable (aunque no exigido) que se ponga un grupo de instrucciones que se ejecuten si la variable toma un valor que no se ha definido. Estas últimas se ponen después de la etiqueta "`default :`".

Sentencias de salto

Este tipo de sentencias permite romper la ejecución secuencial de un programa. Mas alla que aqui se detalla su funcionamiento y utilización, no es recomendable su uso, salvo en situaciones excepcionales.

*Sentencias **break** y **continue***

Para ver un ejemplo de estas dos instrucciones, estudiemos el siguiente código:

```
# include <stdio.h>
   int main( ) {
    int xx;
    for (xx = 5 ; xx < 11 ; xx ++) {
      if (xx == 8) break;
      printf ("Este bucle se ejecuta cuando xx es menor de 8, ahora xx es %d\n", xx);
    }
(9) for (xx = 5 ; xx < 11 ; xx ++) {
      if (xx == 8) continue;
      printf ("Ahora xx es diferente de 8, xx tiene el valor de %d\n", xx);
     }
    return 0;
   }
```

Resultado de la ejecución:

```
Este bucle se ejecuta cuando xx es menor de 8, ahora xx es 5
Este bucle se ejecuta cuando xx es menor de 8, ahora xx es 6
Este bucle se ejecuta cuando xx es menor de 8, ahora xx es 7

Ahora xx es diferente de 8, xx tiene el valor de 5
Ahora xx es diferente de 8, xx tiene el valor de 6
Ahora xx es diferente de 8, xx tiene el valor de 7
Ahora xx es diferente de 8, xx tiene el valor de 9
Ahora xx es diferente de 8, xx tiene el valor de 10
```

Observe que en el primer bucle for, existe una instrucción if que llama a un **break** si **xx** es igual a **8**. La instrucción **break** lleva la ejecución del programa a salir del bucle que se estaba ejecutando para continuar con las instrucciones inmediatas al bucle, terminando éste en forma efectiva. Se trata de una instrucción muy útil cuando se desea salir de un bucle dependiendo de los resultados que se obtengan dentro del mismo. En este caso, cuando **xx** alcanza el valor de **8**, el bucle termina imprimiendo el último valor válido: **7**. La instrucción "**break;**" provoca un salto de la ejecución a la sentencia que se encuentra inmediatamente después de la llave que cierra el bucle (otro for).

El siguiente bucle **for** que empieza en la línea 9, contiene una sentencia "**continue;**" el cual no finaliza el bucle pero ***suspende el presente ciclo***. Cuando el valor de **xx** alcanza como en este caso, el valor 8, el programa brincará al final del bucle para continuar la ejecución del mismo, eliminando así la instrucción **printf()** cuando en la variable **xx** alcanza el valor 8. El enunciado **"continue;"** siempre produce un salto al final del bucle, justo antes de la llave que indica el fin del bucle.

*Sentencia **goto***

> **Observación:** *Con el uso de esta sentencia el control se transfiere directamente al punto etiquetado con un identificador especificado. El "*goto*" es un mecanismo que está en guerra permanente, y sin cuartel, con la programación estructurada. El "*goto*" **NO SE USA**, se incluye aquí porque existe, pero siempre **puede y debe ser eludido.** Existen mecanismos suficientes para hacer todo aquello que pueda realizarse con un "*goto*". En resumen, los que nos consideramos programadores y valoramos el poder de programación que nos brinda* C/C++*, vemos la sentencia "*goto*" como una* **mala palabra.***.*

La sentencia **goto** (ir a) nos permite hacer un salto a la parte del programa que deseemos. En el programa podemos poner etiquetas, estas etiquetas no se ejecutan. Es como poner un nombre a una parte del programa. Estas etiquetas son las que nos sirven para indicar a la sentencia **goto** dónde tiene que saltar. Por ejemplo:

```
   int main( ) {

     printf ( "Línea 1\n" ) ;
(4)  goto linea3 ; /* Le decimos al goto que busque la etiqueta linea3 */
     printf ( "Línea 2\n" ) ;

  linea3: /* Esta es la etiqueta */
     printf ( "Línea 3\n" ) ;
     return 0;
}
```

Como resultado tenemos:

```
Línea 1
Línea 3
```

O sea que el printf() de la línea 4 no se ejecuta, puesto que lo saltamos con el goto....

*Sentencia **return***

Esta sentencia sale de la función donde se encuentra y devuelve el control a la instancia o rutina que la llamó, opcionalmente con un valor de retorno.

Sobre las sentencias de salto y la programación estructurada: *Lo dicho para "*goto*" es válido para todas las sentencias de salto, salvo "*return*" y "*break*", con este último se tiene un poco más de tolerancia, sobre todo en las sentencias "switch", donde resulta imprescindible. En general, es una buena norma huir de las sentencias de salto.* ***C*** *nos brinda los mecanismos de programación necesarios y suficientes para poder tomar distancia de estas sentencias.*