Ciencias computacionales

Propedeutico: Programación

Dra. Hayde Peregrina

Contents

1 Estructuras de Control **2**
1.1 La estructura **if** . . . 2
1.1.1 If-else . . . 3
1.1.2 Break . . . 5
1.1.3 Links videos . . . 5
1.2 Ciclos (loops) . . . 6
1.2.1 Ciclo for . . . 7
1.2.2 Ciclo while . . . 8
1.2.3 Ciclo do-while . . . 9
1.2.4 Break y continue . . . 10

1 Estructuras de Control

Los programas pueden ser provistos de la habilidad de tomar desiciones mediante estructuras de selección. C++ proporciona tres tipos de estructuras de selección: `if`, `if-else` y `switch`. La estructura `if` es la más simple y selecciona si ejecuta una acción o no dependiendo de si la la condición que evalúa resulta verdadera o no. La estructura `if-else` realiza una acción si la condición evaluada resulta verdadera o bien ejecuta otra acción si resulta falsa. La estructura `switch` contempla muchas acciones diferentes dependiendo del valor de una expresión entera. A continuación se describen con más detalle estas estructuras.

1.1 La estructura if

La estructura `if` permite que un programa decida si ejecuta una sentencia o un bloque de sentencias, en caso de que la condición sea verdadera, o bien si salta su ejecución, en caso de que la condición sea falsa.

La figura 1.1 muestra el diagrama de flujo de una estructura condicional. El rombo es un simbolo de desicion que indica que el flujo del programa continuará a lo largo de una ruta determinada de acuerdo a la evaluación, verdadera o falsa, de la expresión contenida en el. Si la expresion resulta verdadera, el flujo del programa se desvia temporalmente para ejecutar la sentencia o el bloque de sentencias asociadas al `if`. Una vez terminada la ejecución de dicho bloque, el programa retoma su secuencia.

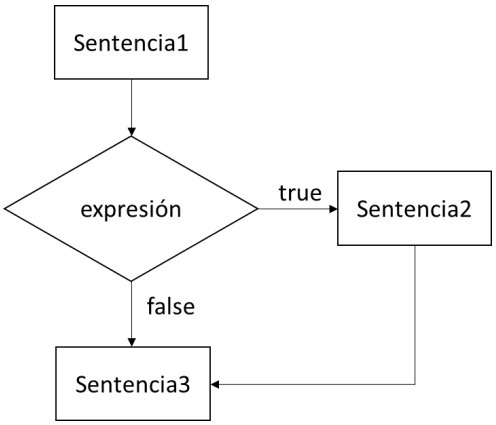

Figure 1: Estructura condicional if.

La sintaxis de la estructura if es la siguiente:

```
1
2  if (expresión){
3    sentencia_1;
4    sentencia_2;
5    .
6    .
7    .
8    sentencia_n;
9  }
```

Aquí `expresión` es el criterio condicional que, en caso de resultar verdadero, permite que se ejecuten las sentencias correspondientes al `if`. Observe que en este caso el `if` comprende un bloque de instrucciones, indicado dentro de las llaves { y }; si el `if` tiene solo una sentencia, entonces las llaves pueden omitirse. Por ejemplo, la expresión:

```
1  if (x>0.0)
2      y=sqrt(x);
```

Aquí el criterio condicional es la expresión relacional `x>0.0`, la cual se evalua a un valor de 1 (verdadero) si `x` es mayor que cero, y se evalúa con 0 (falso) si `x` es menor o igual a 0. Si se quiere incluir más de una sentencia asociada al `if`, entonces este bloque debe estar comprendido entre llaves formando un bloque:

```
if (x>0.0){
    y=sqrt(x);
    cout<<"La raiz cuadrada de ":<<x<<" es: "<<y<<endl;
}
```

1.1.1 If-else

También es posible especificar qué sentencias deben ser ejecutadas en caso de que la expresión sea evaluada como falsa. Para ello, existe la estructura `if-else`. La sintaxis es:

```
if (expresión){
  sentencia1;
  sentencia2;
  .
  .
  .
}else{
  sentenciaA;
  sentenciaB;
  .
  .
  .
}
```

Si `expresión` es evaluada como 1, se ejecutaran las sentencias del bloque `if`; por el contrario, si la evaluación fue 0 entonces se ejecutaran las sentencias del bloque `else` (ver Fig. 1.1.1). Por ejemplo, el siguiente código evalua si un numero dado por el usuario es par o impar. Dependiendo de la evaluación de la expresión se puede ejecutar un bloque de sentencias u otro y, una vez terminado, se continúa la ejecución del programa.

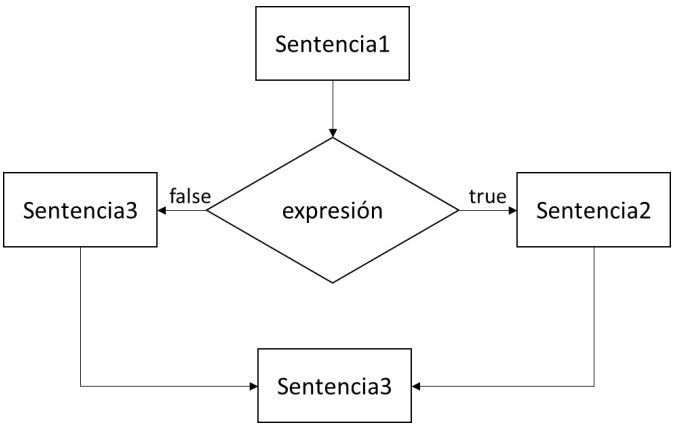

Figure 2: Estructura condicional if.

```
if (numero%2 == 0){
  cout<<numero<<":"<<;
  cout<<"Numero par.";
}else{
```

```
5     cout<<numero<<":"<<;
6     cout<<"Numero␣impar.";
7   }
```

Se pueden tener varias estructuras `if-else` concatenadas a fin de abarcar varios casos. Por ejemplo, el siguiente código subre tres casos posibles dado un numero `x` este puede ser: positivo, negativo o cero. Si la primer expresión resulta verdadera, el numero `x` es positivo; en caso contrario, no se tiene un resultado definido sino que es necesario volver a evaluar por lo que se utiliza otro `if`. El segundo `if` evalúa a `x` con otra expresion que al resultar verdadera informa que el numero es negativo o, en caso contrario que el numero es un cero.

```
1   if (x > 0){
2     cout<<x<<"␣es␣positivo"<<;
3   }else if (x < 0){
4     cout<<x<<"␣es␣negativo"<<;
5   }else{
6     cout<<x<<"␣es␣0"<<;
7   }
```

Así, la estructura `if-else` puede extenderse cuando es necesario tomar decisiones sucesivas relacionadas. Sin embargo, cuando se tiene un numero mayor de casos el uso de `if-else` puede resultar poco legible y compleja de seguir. La estructura `switch` se enfoca en evaluar multiples casos y presentarlos en forma clara. La estructura `switch` consiste en una serie de etiquetas asociadas a cada caso (`case`) y un caso opcional (`default`). De forma general, `switch` tiene la siguiente sintaxis:

```
1   swtich (expresión){
2     case Expresion_constante1:
3          sentencia1;
4     case Expresion_constante2:
5          sentencia2;
6     .
7     .
8     .
9     default:
10         sentencia_valor_por_default;
11  }
```

`Switch` actua como un enrutador que indica en el código cuál línea de código debe ejecutar entre multiples opciones. La expresión que evaluada debe ser de tipo entero y, dependiendo de su valor, se salta a la etiqueta correspondiente y se ejecuta el bloque de sentencias que tenga asociado. Si `expresión` produce un valor igual a `Expresion_constante1`, entonces se ejecuta `sentencia1`. Si expresión produce un valor igual a `Expresion_constante2`, entonces `sentencia2` se ejecuta primero. En caso de que el valor de expresión no coincida con el valor de alguna etiqueta, entonces se ejecutará `sentencia_valor_por_default` correspondiente a la etiqueta `default`. Por ejemplo, considere el siguiente código:

```
1
2   switch (dia){
3     case 1:
4          cout<<"Lunes";
5     case 2:
6          cout<<"Martes";
7     case 3:
8          cout<<"Miercoles";
9     case 4:
10         cout<<"Jueves";
11    case 5:
12         cout<<"Viernes";
13    default:
```

```
14          cout<<"Fin de semana!!!";
15  }
```

La estructura switch evalúa la variable entera día con valor dado por el usuario dentro del rango [1-7]. Si el usuario introduce un 6 o un 7, al evaluar la variable día esta no coincidirá con ninguna de las etiquetas case, por lo tanto se ejecutará la sentencia asociada a la etiqueta default. ¿Cuál sería el resultado si el usuario ingresa dia=4?. La variable dia coincidirá con la cuarta etiqueta del switch y ejecutará la impresión "Jueves" pero también se ejecutarán las impresiónes de los casos que le siguen. ¿Por qué ocurre esto?. Esta es una característica importante de la instrucción switch: la computadora continua ejecutando las instrucciones que estan dspués del caso seleccionado hasta el final de la instrucción.

1.1.2 Break

Comunmente se requiere que solo las sentencias asociadas a un caso sean las que se ejecuten. Para ello se utiliza la sentencia break al final de la lista de sentencias que tiene cada caso. De esta forma, break finaliza la instrucción switch y continua con la ejecución de sentencias fuera de su bloque de instrucciones. El código anterior puede modificarse como:

```
1   swtich (dia){
2     case 1:
3          cout<<"Lunes";
4          break;
5     case 2:
6          cout<<"Martes";
7          break;
8     case 3:
9          cout<<"Miercoles";
10         break;
11    case 4:
12         cout<<"Jueves";
13         break;
14    case 5:
15         cout<<"Viernes";
16         break;
17    default:
18         cout<<"Fin de semana!!!";
19         break;
20  }
```

De esta forma si el usuario da un valor dia=4, solo se ejecutara la impresión "Jueves". El uso de break en la etiqueta default es opcional puesto que si este es el caso no hay mas bloques de sentencias por ejecutar. Podemos simplificar el flujo de una sentencia switch como se muestra en la figura 1.1.2.

1.1.3 Links videos

- If, If-else en C++:

  `https://youtu.be/mn554EZ_K-k?list=PLw8RQJQ8K1ySN6bVHYEpDoh-CKVkL_uOF`

  `https://youtu.be/i179tR35cPw?list=PL58A7476AD725E577`

- Switch en C++

  `https://youtu.be/ZECPkOmA5Yk?list=PL58A7476AD725E577`

  `https://youtu.be/b_O2nRnkiUw?list=PL58A7476AD725E577`

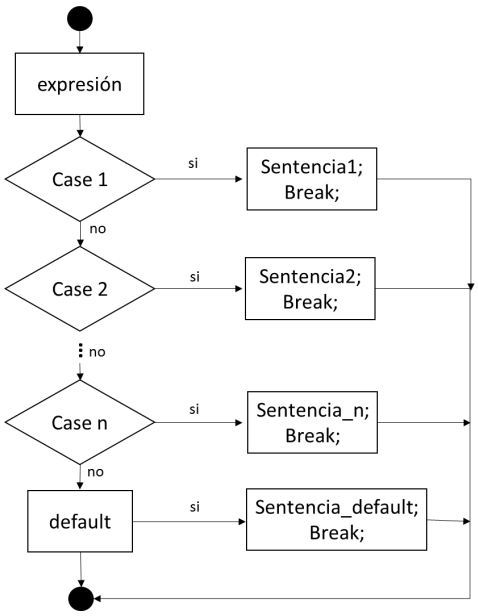

Figure 3: Estructura condicional if.

1.2 Ciclos (loops)

Material multimedia:

- Video ciclo for: `https://www.youtube.com/watch?v=pqA5pnXA4To`
- Video ciclo while y do-while: `https://www.youtube.com/watch?v=nuJsajtZBMw`
- Video sentencias brake y continue: `https://www.youtube.com/watch?v=MWaI1pRYqtk`

Expresiones booleanas

Los operadores de relación, así como los operadores lógicos se emplean para formar *expresiones booleanas.* Por expresión booleana debemos entender a una expresión que solo puede dar como resultado los valores verdadero (*true*) o falso (*false).* Los lenguajes de programación C y C++ usan el valor 0 para indicar un valor falso, y un valor distinto de 0 para indicar que se tiene una expresión verdadera. Ademas, en C++ se tiene el tipo de dato bool que puede tener los valores *true* o *false.* Por ejemplo, en C++ podemos declaran una variable de la siguiente forma:

```
1 bool esPalindromo;
```

y posteriormente asignarle un valor de verdad:

```
1 esPalindromo = false;
```

Los operadores de relación, así como las expresiones booleanas han sido explicadas en la sección 1.

Proposiciones y bloques

Una proposición es cualquier expresión seguida de un punto y coma (;). Un bloque, es una expresión compuesta o conjunto de expresiones agrupadas entre llaves { y }. Un ejemplo de bloque puede ser el siguiente:

```
1 {//Inicio de bloque
2
3         int a, b, c;
4
```

```
5           a = 5;
6           b = 7;
7           c = a+b;
8           printf("\%d", c);
9
10  }//Final de bloque
```

1.2.1 Ciclo for

El ciclo **for** es una instrucción de repetición, la cual es utilizada para ejecutar una proposición o bloque varias veces. La instrucción **for** es comúnmente utilizada para implementar un ciclo controlado por un contador.

El formato general de una instrucción **for** es:

```
1   for( <inicialización> ; <expresión booleana> ; <progresión> )
2           <proposición o bloque>
```

La inicialización solo se ejecuta una vez antes de iniciar el ciclo, y por lo general se encarga de inicializar variables. Al inicio de cada iteración se evalúa la expresión booleana y si el resultado es verdadera pasara a ejecutarse la proposición o bloque. Al final de cada iteración se evalúan las instrucciones que se encuentran en la sección de progresión, lo cual nos permite incrementar o decrementar contadores, por lo general encargados de las iteraciones del ciclo.

A continuación se muestra un ejemplo de ciclo **for**:

```
1   #include <iostream>
2   #include <stdio.h>
3   using namespace std;
4
5   int main(int argc, const char *argv[]) {
6
7           for (int i = 0; i < 5 ; i++)
8                   cout << "Propocición␣" << i << endl;
9   }
```

El ciclo anterior realiza una inicialización de la variable i con el valor 0. Posteriormente inician las iteraciones del ciclo, para lo cual es evaluada la condición $i < 5$, y siempre que sea verdadera se ejecutara la sección de proposición o bloque, en caso de que la expresión sea falsa terminará la ejecución del ciclo **for**. Por ultimo, al final de cada iteración se ejecutará la sección de progresión, la cual es comúnmente utilizada para incrementar o decrementar el valor de variables, por lo general la que se utiliza en la expresión booleana. Sin embargo, en esta sección pueden ir varias instrucciones seguidas por una coma.

El siguiente ejemplo manda mensajes dependiendo de que sección se esta ejecutando (se recomienda al lector probar este código y realizar modificaciones de su interés):

```
1   #include <iostream>
2   #include <stdio.h>
3   using namespace std;
4
5   int main(int argc, const char *argv[]) {
6
7   for( int i=!printf("Iicialización\n"); i<5 && printf("\tExp.␣booleana\n"
        ); i++,printf("\t\t\tProgresión\n")) {
8                   cout << "\t\tProposición␣" << i << endl;
9           }
10  }
```

La instrucción !printf("...") retorna un valor 0, por lo cual la variable i es inicializada con el valor 0. La instrucción printf("...") retorna un valor distinto de cero, lo cual es interpretado como un valor verdadero en la sección de expresión booleana. El bloque de código anterior imprime en pantalla lo siguiente:

```
$ Inicialización
$               Exp. booleana
$                             Proposición 0
$                                           Progresión
$               Exp. booleana
$                             Proposición 1
$                                           Progresión
$               Exp. booleana
$                             Proposición 2
$                                           Progresión
$               Exp. booleana
$                             Proposición 3
$                                           Progresión
$               Exp. booleana
$                             Proposición 4
$                                           Progresión
```

Como se puede observar el ciclo entra 5 veces. La variable i es inicializada con 0, y en cada iteración se incrementa en 1. Una vez que la variable i llega al valor 5 la expresión booleana $i < 5$ retorna un valor falso y el ciclo termina.

Todas las secciones del ciclo **for** son opcionales y en caso de que alguna no se requiera puede omitirse, y dejar el espacio vació. Por otro lado, aunque no se haya especificado alguna sección los puntos y comas son necesarios.

En la figura 4 se muestra la secuencia de ejecución de una instrucción for.

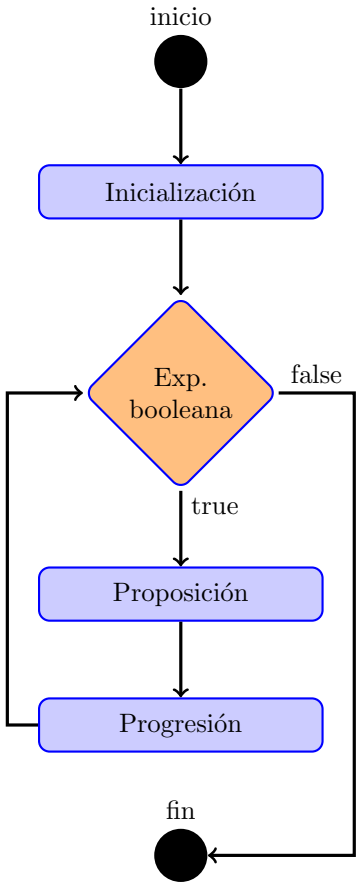

Figure 4: Diagrama de flujo para la instrucción **for**.

1.2.2 Ciclo while

El formato general de la instrucción **while** es el siguiente:

```
1 while( <expresión booleana> )
```

```
2         <proposición o bloque>
```

En esta estructura la proposición o bloque se ejecutará cada vez que la expresión booleana sea verdadera. Esta estructura carece de un bloque de inicialización, por lo cual es común hacer inicializaciones antes de la instrucción **while**. También carece de una sección especifica de progresión, por lo cual es común realizar cambios dentro del ciclo que posteriormente permitan obtener un resultado falso en la expresión booleana y así terminar con el ciclo y continuar con las siguientes instrucciones del programa. Es común utilizar un ciclo **while** infinito en algunas aplicaciones que solo estén a la espera de alguna entrada o evento.

Acontinuación se muestra un ejemplo de ciclo **while**:

```
1 int i = 0;
2 ...
3 while (i < 10)
4         printf("i=%d\n", i++);
```

En este ejemplo la inicialización de la variable i se realizó antes de la instrucción **while**. La expresión booleana es la encargada de continuar la ejecución de la proposición o de terminar el ciclo. El incremento de la variable i, que en algún momento permitirá terminar el ciclo y continuar con le ejecución de las demás instrucciones del programa se realiza dentro la sección de proposición o bloque de la instrucción **while**.

En la figura 5 se muestra la secuencia de ejecución de una instrucción while.

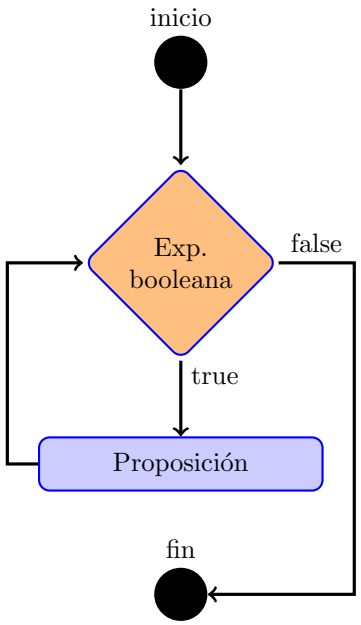

Figure 5: Diagrama de flujo para la instrucción **while**.

1.2.3 Ciclo do-while

La instrucción de repetición **do-while** es muy similar a la instrucción **while**. En una instrucción **while** el programa evalúa la expresión booleana (también conocida como condición) en cada iteración antes de ejecutar la proposición o bloque; en caso de que esta sea falsa el ciclo **while** termina. Por otra parte, en caso de que sea verdadera se ejecutara la sección de proposición o bloque. El ciclo **do-while** realiza la condición de terminación al final de cada ejecución de la sección proposición(es); por lo tanto esta sección siempre se ejecuta por lo menos una vez. Por su parte el bloque de proposición en un ciclo **while** puede nunca ejecutarse, en cambio en un ciclo **while** se ejecutará por lo menos la primera vez.

El formato general de la instrucción **do-while** es el siguiente:

```
do{
        <proposición(es)>;
}while( <expresión booleana> );
```

En el siguiente ejemplo se muestra un ciclo **do-while**:

```
int i=0;
...
do{
        printf("i␣=␣%d\n", i++);
}while( i < 10);
```

En la figura 6 se muestra la secuencia de ejecución de una instrucción do-while.

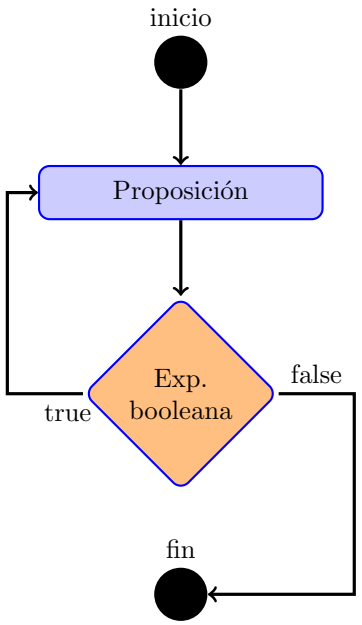

Figure 6: Diagrama de flujo para la instrucción **do-while**.

1.2.4 Break y continue

La proposición **break** produce una salida anticipada de un ciclo **for, while o do-while**, y también produce la salida de la sentencia **switch**. En el siguiente ejemplo la instrucción **break** produce que el ciclo solo se ejecute 20 veces, a pesar de que la expresión booleana dentro del ciclo siga siendo verdadera.

```
for(int i=0; i < 100; i++){
        printf("i␣=␣%d\n", i);
        if(i == 20)
                break;
}
```

La instrucción **continue** permite que se inicie la siguiente iteración de un ciclo **for, while o do-while** que la contiene. Una vez que se ejecuta la instrucción **continue** las instrucciones que siguen dentro del bloque de proposiciones después de la linea **continue** no se ejecutarán. En el siguiente ejemplo el uso de **continue** permite que se impriman sólo los números múltiplos de 5:

```
int i=0;
...
while(i<100){
        if(i++ % 5)
```

```
5                   continue;
6           printf("%d\n", i - 1);
7   }
```