UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño
y programación de Videojuegos

UNLVIRTUAL

Introducción a la programación

Unidad Temática Número 3

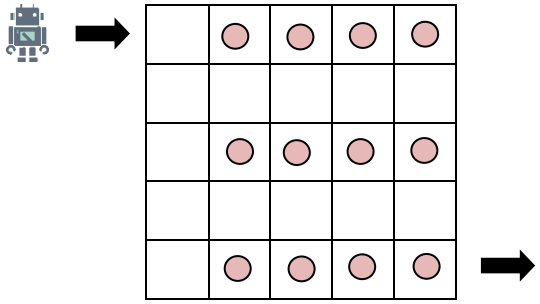
Estructuras de Control

Objetivo: Implementar Estructuras de Control en C++

Temas: Programación Estructurada, Secuencia, Expresiones, Operadores, Estructuras de Control Repetitivas y Condicionales

INTRODUCCIÓN

Ya vimos que las estructuras de control son bloques de instrucciones que nos permiten tomar decisiones con nuestros datos y a partir de ahí obrar según necesitemos. Vimos la estructura de control **Si condicional** y la estructura de control **Case**, que presta varias similitudes al Si, pero tiene ciertas particularidades que lo aventajan en algunas situaciones. A continuación veremos tres estructuras más que nos permiten iterar (repetir) sobre el código. Esto es necesario cuando tenemos situaciones dónde nos encontramos con ciertos patrones en el algoritmo, por ejemplo el caso de autómata, dónde las instrucciones para juntar cada fila eran prácticamente idénticas.



La primera estructura se llama FOR o LOOP y es una estructura iterativa fija, ya que nos permite realizar un número fijo de repeticiones, las otras dos son muy similares, se llaman While y Do-While y son estructuras iterativas condicionales, es decir que van a iterar hasta que una condición se cumpla (o deje de cumplirse)

ESTRUCTURA DE CONTROL: FOR

La estructura FOR es muy simple de usar, básicamente nos permite repetir una cierta cantidad de veces lo que tengamos dentro de su ámbito. Su sintaxis puede parecer un poco compleja al principio, pero es fácil de manejar una vez que la recordamos. Partamos del caso común:

```
for(int i=0; i<10; i++){
    //se repite 10 veces lo que
    // este aqui dentro
}
```

Este es el más caso común, y es posiblemente el que utilicemos la mayoría de las veces, vamos a analizarlo por partes.
La estructura FOR utiliza 3 expresiones que se separan por punto y coma (;).

```
for (INICIALIZACIÓN; CONDICIÓN; ALTERACIÓN)
```

`INICIALIZACIÓN`: En la `INICIALIZACIÓN` se inicializan los índices que servirán de condición. En el ejemplo se declara la variable entera *i* que variará su valor y servirá de referencia para saber cuándo se cumple la condición. Aquí la haremos recorrer de 0 a 9 (10 veces), que serán la cantidad de ciclos que tiene el ejemplo.

`CONDICIÓN`: En la condición se ponen los términos que harán que la estructura finalice su iteración. En el ejemplo el fin de la iteración se dará cuando la variable (índice) *i* llegue a 10. La iteración del FOR se ejecutará mientras la condición se cumpla. Cuando *i* vale 10, la condición *i < 10* deja de ser cierta y el ciclo finaliza.

`ALTERACIÓN`: En la alteración se modifican las variables que sirven de índice. En el ejemplo se incrementa en 1 el valor de *i*. Si la variable índice nunca incrementa su valor o se aleja de la condición de corte, entonces jamás se cumple la condición y la estructura FOR se repite por siempre dando un error lógico.

Ejecución: La ejecución del For es de arriba hacia abajo como veníamos trabajando en la programación estructurada tradicional. En primera instancia se ejecutan por única vez las

instrucciones que tengamos en la Inicialización y se verifica la condición. A continuación inicia la serie de instrucciones hasta llegar a la última llave (la llave de cierre de la estructura). Una vez que se llega a la última llave, el hilo de ejecución vuelve a la instrucción For, realiza la alteración de la variable índice y luego verifica la condición. Si esta es verdadera, vuelve a realizar todas las instrucciones hasta la llave final y así hasta que la condición deje de cumplirse (sea falsa).

Las variables (índices) declaradas en la Inicialización del For existen dentro del ámbito de la estructura (no fuera), persisten y pueden ser accedidas. El resto del ámbito del For se reinicia en cada ciclo. Esto quiere decir que en cada ciclo lo que se encuentre dentro de las llaves se ejecutará nuevamente, si se declara alguna variable ésta se volverá a declarar y al fin del ciclo perderá su valor y se destruirá. Las únicas variables que permanecen son aquellas que pertenecen a ámbitos superiores y también las variables índices.

Por ejemplo, el siguiente código muestra los números del 1 al 10:

```
#include<iostream>
using namespace std;

int main (int argc, char *argv[]) {

    //Numeros del 1 al 10
    for(int i=1;i<11;i++){

        cout<<i<<endl;
    }

    return 0;
}
```

La variable i existe dentro del For y se actualiza en cada ciclo, pero fuera de la estructura no existe. Cualquier variable declarada dentro del For se volverá a iniciar en cada ciclo.

Los siguientes son ejemplos de FOR:

```
    //for de 10 ciclos: de 0 a 9
    for(int i=0; i<10; i++){}

    //for de 5 ciclos: de 1 a 6
    for(int k=1; k<6; k++){}

    //for de 10 ciclos: de 0 a 9
    for(int j=0; j=<9; j++){}

    //for de 10 ciclos: de 10 a 20
    for(int i=10; i<20; i++){}
```

De ser necesario, puede cambiarse el paso del índice, en vez de ir de 1 en 1, puede avanzar 2, 3, o lo que sea necesario:

```
    //for de 100 ciclos: de 10 en 10,
    for(int i=0; i<100; i+=10){}
```

Incluso puede ir en retroceso:

```
    //Numeros del 10 al 1
    for(int i=10; i>0; i--){}
```

En el siguiente ejemplo calculamos el promedio de 5 números ingresados por teclado:

```
#include<iostream>
using namespace std;

int main (int argc, char *argv[]) {

    int sumatoria = 0;

    for(int i=0; i<5; i++){

        int numero;
        cout<<"Ingrese un numero: ";
        cin>>numero;
        sumatoria += numero;

        cout<<endl;
    }

    cout<<"El promedio es: "<<sumatoria/5 <<endl;

    return 0;
}
```

ESTRUCTURAS DE CONTROL: WHILE Y DO-WHILE

Las estructuras de control While y Do-While son estructuras muy similares, y si bien parecen que funcionan de la misma manera, su aplicación es significativamente distinta.

El For nos permitía iterar sobre un conjunto de instrucciones un determinado número de veces, pero qué ocurre si queremos iterar sobre una acción de la cual no sabemos cuándo va a finalizar, lo único que sabemos es que esperamos que algo ocurra o bien una condición se cumpla. Este es el ejemplo de la mayoría de las situaciones. Un juego mismo, no termina hasta que completamos el último nivel y cumplimos con los objetivos, sin saber si nos va a llevar 1 o 100 horas, o si vamos a ocupar las 3 vidas. Un juego no inicia mientras no coloquemos una moneda o presionemos *start*.

La estructura While posee la siguiente sintaxis:

```
    while (expresion){
        //este codigo se ejecutara
        //mientras expresion sea cierta
    }
```

Por ejemplo, podríamos ingresar un valor por teclado hasta que éste sea mayor a 10.

```
    int valor = 0;

    while (valor<10){
        cout<<"Ingrese un numero mayor a 10"<<endl;
        cin>>valor;
    }
```

Lo que se encuentra dentro de las llaves del while se ejecutará de arriba abajo hasta que la validación del valor<10 sea cierta.

Aclaración. La variable valor debe existir y tener un valor para que la expresión del while pueda utilizarla. Una solución podría haber sido ingresar el valor una vez por fuera de la estructura while y luego continuar con la validación dentro de la estructura.

Ejecución: El hilo de ejecución del While se inicia en la evaluación de la expresión, si ésta es falsa el ciclo termina (si es falsa antes de ingresar, nunca ingresa). A continuación se ejecutan todas las instrucciones dentro de las llaves de while, de arriba hacia abajo, al llegar a la última llave, el hilo de ejecución vuelve a evaluar la expresión y el ciclo vuelve a empezar. Todo lo que se declare dentro del ámbito del while se reiniciará en cada ciclo y no existirá fuera de la estructura.

La estructura Do-While es similar, su sintaxis es la siguiente:

```
do{
    //este codigo se ejecuta
    //mientras que la expresion sea cierta
}
while (expresion);
```

El comportamiento del Do-While es igual al del While, el código dentro de las llaves se ejecuta mientras que la condición de la expresión no sea cierta. La gran diferencia que existe es que el Do-While ejecuta al menos una vez la secuencia dentro de las llaves.
Si quisiéramos hacer el mismo ejemplo de insertar un número hasta que sea mayor a 10, con el Do-While no tendríamos problemas de que la variable esté inicializada antes de empezar:

```
int valor;
do{
    cout<<"ingrese un numero mayor a 10"<<endl;
    cin>> valor;
}
while (valor<10);
```

Aquí la variable valor puede no estar inicializada (aunque sí tiene que estar declarada ya que el ámbito de la consulta de la expresión va por fuera de la estructura). Dentro de la estructura le asignamos un valor mediante el ingreso por teclado, recién al finalizar el conjunto de instrucciones es que se evalúa la expresión.

Ejecución: El hilo de ejecución del Do-While se inicia en el Do, luego se ejecutan todas las instrucciones dentro de la llave, finalmente se evalúa la expresión, si es cierta el hilo de ejecución vuelve al Do y la estructura vuelve a empezar. Todo lo que se declare dentro del ámbito del Do-While se reiniciará en cada ciclo y no existirá fuera de la estructura.