GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: INSTRUCCIÓN BUCLE FOR

1. Introducción

En el diseño e implementación de un algoritmo el mismo no siempre va ser de forma lineal (que se ejecuten cada instrucción una detrás de la otra) sino que llegado determinado paso del algoritmo el mismo debe ser capaz repetir una o un cojunto de instrucciones mientras se cumpla una determinada condición y cuando esta deje de cumplirse seguir con el resto de las instrucciones del programa. Para poder realizar es necesario el uso de estructura de control las cuales permiten modificar el flujo de ejecución de las instrucciones de un algoritmo.

Vamos analizar dentros de la estructura de control las sentencias de bucle aquella que nos permite iterar sobre un intervalo o rango de valores númericos previamente conocido y con determinado paso. Hablamos de la estructura **for**

2. Conocimientos previos

La instrucción **break** interrumpe la ejecución del bucle donde se ha incluido, haciendo al programa salir de él aunque la expresion de control correspondiente a ese bucle sea verdadera. La sentencia **continue** hace que el programa comience el siguiente ciclo del bucle donde se halla, aunque no haya llegado al final de las sentencia compuesta o bloque.

Un **bucle** se utiliza para realizar un proceso repetidas veces. Se denomina también **lazo** o **loop**. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalizar el bucle (booleanExpression) no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores poco experimentados.

3. Desarrollo

El **for** es quizás el tipo de bucle mas versátil y utilizado en la programación. Su forma general es la siguiente:

```
for (inicializacion; expresion_de_control; actualizacion){
   sentencia;
}
```

Posiblemente la forma más sencilla de explicar la sentencia **for** sea utilizando la construcción **while** que sería equivalente. Dicha construcción es la siguiente:

```
inicializacion;
while (expresion_de_control){
  sentencia;
  actualizacion;
}
```

donde **sentencia** puede ser una única sentencia terminada con (;), otra sentencia de control ocupando varias líneas (**if, while, for**, ...), o una sentencia compuesta o un bloque encerrado entre llaves {...}. Antes de iniciarse el bucle se ejecuta **inicializacion**, que es una o más sentencias que asignan valores iniciales a ciertas variables o contadores. A continuación se evalúa **expresion_de_control** y si es **false** se prosigue en la sentencia siguiente a la construcción **for**; si es **true** se ejecutan sentencia y actualizacion, y se vuelve a evaluar **expresion_de_control**. El proceso prosigue hasta que **expresion_de_control** sea **false**. La parte de **actualizacion** sirve para actualizar variables o incrementar contadores. Un ejemplo típico puede ser el producto escalar de dos vectores a y b de dimensión n:

```
for(int pe =0, i=1; i<=n; i++){
   pe+=a[i]*b[i];
}
```

Primeramente se inicializa la variable **pe** a cero y la variable **i** a 1; el ciclo se repetirá mientras que **i** sea menor o igual que **n**, y al final de cada ciclo el valor de **i** se incrementará en una unidad. En total, el bucle se repetirá **n** veces. Obsérvese que la **inicializacion** consta de dos sentencias separadas por el operador (,). Cualquiera de las tres partes puede estar vacía. La **inicializacion** y la **actualizacion** pueden tener varias expresiones separadas por comas.

4. Implementación

La implementación de esta estructura es similar en ambos lenguajes no existe ninguna diferencia.

4.1. C++

```
for ( int i =1; i <=10; i++){
   cout<<" Hola Mundo " ;
}
```

Esto indica que el contador **i** inicia desde **1** y continuará iterando mientras **i** sea menor o igual a **10** (en este caso llegará hasta 10) e **i++** realiza la suma por unidad lo que hace que el **for** y el contador se sumen repitiendo 10 veces *Hola Mundo* en pantalla.

```
for( int i=10; i>=0; i--){
   cout<<" Hola Mundo " ;
}
```

Este ejemplo hace lo mismo que el primero, salvo que el contador se inicializa a 10 en lugar de 1; y por ello cambia la condición que se evalúa así como que el contador se decrementa en lugar de ser incrementado.

La condición también puede ser independiente del contador:

```
int j=20;
for( int i=0;j>0;i++){
   cout<<" Hola "<<i<<" - "<<j<<endl;
   j--;
}
```

En este ejemplo las iteraciones continuaran mientras i sea mayor que 0, sin tener en cuenta el valor que pueda tener i.

4.2. Java

Ahora vamos a ver los mismos ejemplos en Java

```
for ( int i =1; i <=10; i++){
   System.out.print(" Hola Mundo ") ;
}
```

```
for( int i=10; i>=0; i--){
   System.out.print(" Hola Mundo ") ;
}
```

```
int j=20;
for( int i=0;j>0;i++){
   Syste.out.println(" Hola "+i+" - "+j);
   j--;
}
```

5. Complejidad

Para calcular el tiempo de ejecución del resto de sentencias iterativas (FOR, REPEAT, LOOP) basta expresarlas como un bucle WHILE. Por tanto una vez expresado el for como while para calcular la complejidad basta con recordar que la misma se calcula como:

El tiempo de ejecución de un bucle de sentencias WHILE C DO S END; es T = T(C) + (no iteraciones)*(T(S) + T(C)). Obsérvese que tanto T(C) como T(S) pueden variar en cada iteración, y por tanto habrá que tenerlo en cuenta para su cálculo. Donde:

1. **T(C) :** Es la complejidad de la expresión de control que se define dentro de los parentisís del while.
2. **T(S) :** Es la complejidad del bloque de instrucciones que conforman las instrcciones que serán ejecutadas si la expresión de control del while es verdadera.

6. Aplicaciones

La ventaja de la construcción **for** sobre la construcción **while** equivalente está en que en la cabecera de la construcción **for** se tiene toda la información sobre como se inicializan, controlan y actualizan las variables del bucle.

Esta estructura es muy utilizada cuando se necesita iterar sobre un rango o los valores de una estructura lineal iterativa como vectores, arreglos unidimensionales o bidimensionales pero para esta caso tendría que anidar una estructura **for** dentro de otra estructura **for**.

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utilizando esta instrucción:

- Bellotas.
- A*B*C
- A Plus B
- ¿Cuántas fichas de dominó? (I)
- Calculando áreas
- A - A + B
- B - A - B