Estructuras algorítmicas

Las estructuras algorítmicas también denominadas sentencias son un conjunto de acciones u operaciones que pueden ser ejecutadas y permiten, mediante la manipulación de datos,realizar tareas específicas que nos lleven a la solución de problemas.

Las estructuras o sentencias se clasifican en:

Estructuras secuenciales o simples

Son sentencias que no contienen otra sentencia, y puede ser una única línea o un bloque de código que se ejecuta de manera secuencial, es decir que una acción o instrucción sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

Asignación

La sentencia de asignación es una operación que se utiliza para almacenar valores en una zona de la memoria, esta zona será reconocida con el nombre (Identificador) de la variable que recibe el valor.

La sintaxis o forma general de una asignación es la siguiente:

```
<Identificador>=<Expresión>
```

Identificador es un nombre válido declarado anteriormente en la zona de declaración de variables.
Expresión puede ser o bien una variable, una constante, una expresión o fórmula algorítmica, que debe ser del mismo tipo que el dato que la contiene (Identificador).

Ejemplos:

```
a=2;
a=b;
resultado=a+b;
cantidad=cantidad+1;
acum=acum+valor;
dato='S';
```

Según la tarea que realiza la asignación se puede clasificar de la siguiente forma:

Simples: consiste en asignar un valor constante o variable. Ejemplo:

```
a=15;
a=x;
b=0;
c=’a’;
```

Contador: consiste en usarla como un verificador del número de veces que se realiza un proceso. Ejemplo

```
a=a+1;
cont=cont+1;
```

En este ejemplos el contador va de uno en uno que es como generalmente se realiza un determinado conteo.

Acumulador: consiste en usarla como un sumador en un determinado proceso. Ejemplo:

```
a=a+valor;
acum=acum+ dato;
```

En este caso el valor que tomen tanto la variable “a” como la variable “acum”, va a depender de los valores que contengan “valor y “dato” respectivamente.

En ambos casos cuando trabajamos con contadores y acumuladores es necesario inicializar la variable ya que intervienen directamente en la operación.

De trabajo: en este caso puede recibir el resultado de una operación matemática más compleja que involucra muchas variables, constantes, y operaciones. Ejemplo:

```
a=c+b*2/4
res=(a>0) && (b!=a)
m=((a+2)>8) || (a==0)
```

En este caso hay que tomar mucha precaución en la declaración del tipo de dato que será en contenedor para evitar errores o pérdida de datos.

Entrada

Es la operación por medio de la cual se recibe un valor o un dato, a través de algún dispositivo de entrada.

Ejemplo:

```
...
scanf("%d",&dato);
scanf("%f",&valor);
scanf("%s",nombre);
...
```

Salida

Es la operación por medio de la cual se envía el resultado o mensaje, a través de algún dispositivo de salida.

Ejemplo:

```
...
printf("Bienvenidos");
printf("El resultado de la operación es: %d",res);
printf("El doble de %d es: %d",a,b);
...
```

Estas sentencias fueron abordadas en profundidad en un módulo anterior "Elementos de un programa en C".

Estructuras de control

Como lo indica su nombre son sentencias compuestas de otras sentencias que también se ejecutan en secuencia teniendo en cuenta que en general se basan en una prueba que determina la acción que se ha de llevar a cabo, controlando el flujo del programa, es por esto que también se denominan sentencias de control.

Condicionales

Son sentencias que se utilizan para tomar decisiones. Se evalúan una o varias condiciones cuyo resultado será una opción u otra, para lo cual es necesario establecer una condición para determinar si la acción se realiza o no.

El lenguaje C soporta dos tipos de sentencias condicionales a saber , if y switch.

La sentencia if

Simple

En una sentencia (if) simple, se ejecuta una determinada acción cuando se cumple la condición que necesariamente va encerrada entre paréntesis, o dicho de otra forma cuando la condición es verdadera, de lo contrario no se ejecuta nada o el programa sigue con su flujo normal.

La forma general de una sentencia if simple es:

```
...
if (condición)
   {
       instrucción/es;
   }
...
```

Ejemplo:

```
...
if (b > 0)
    {
        printf("%d es positivo", b);
    }
...
```

Doble

Son aquellos que permiten elegir entre dos opciones o alternativas posibles. Es decir, se evalúa la condición, si es cierta o "verdadera", se ejecutan las instrucciones o bloque de instrucciones que está inmediatamente a continuación del if, de lo contrario se ejecuta el bloque de instrucciones que está a continuación de la palabra reservada else.

La forma general de la sentencia if doble es:

```
...
if (condición)
    {
        instrucción/es;
    }
    else
    {
        instrucción/es;
    }
...
```

Ejemplo:

```
...
if (b > 6)
    {
        printf("Aprobado");
    }
    else
    {
        printf("Desaprobado");
    }
...
```

En el ejemplo citado las llaves de apertura y cierre son opcionales, ya que se trata de una sola instrucción por cada salida.

Otro ejemplo:

```
...
    if (importe > 5000)
    {
        desc = importe * 0.10;
        printf("El descuento es: $ %f",desc);
    }
    else
    {
        desc = 0;
        printf("El descuento es: $%f",desc);
    }
...
```

En este ejemplo las llaves de apertura y cierre, son obligatorias ya que se trata de un bloque de instrucciones o se ejecuta más de una instrucción.

Anidados

Cabe destacar que luego de evaluar la condición podemos ejecutar una instrucción simple, un bloque de instrucciones o incluso otro if, en este caso estamos frente a los que se denomina if anidados.

La sintaxis o forma general de un if anidado es:

```
...
if (condición)
{
    if (condición)
    {
       if (condición)
       {
           instrucción / es;
       }
       else
       {
           instrucción / es;
       }
    }
}
...
```

Ejemplo: en este ejemplo utilizamos dos if. El segundo está "anidado" con la condición falsa del primero.

```
...
if (n == 0)
    {
        printf("Es nulo");
    }
    else
    {
        if (n < 0)
        {
            printf("Es negativo");
        }
        else
        {
            printf("Es positivo");
        }
    }
...
```

Muchas veces un if anidado suele ser problemático ya que puede ser difícil saber qué se asocia con que if.

Supongamos el siguiente ejemplo:

```
...
if (x == 0)
        if (y == 0)
            printf("Dio en el centro");
        else
            printf("Fuera");
...
```

En C una sentencia else siempre se refiere al if precedente más próximo que no tenga ya asociada una sentencia else. En este caso el else está asociado con la sentencia if(y...).

Es muy útil aunque no sea obligatorio utilizar las llaves de cierre y apertura {}, de esta forma no solo se definen mejor los bloques sino que además se puede saltear la asociación normal de la condición.

Veamos el mismo ejemplo anterior utilizando llaves de apertura y cierre:

```
...
if (x == 0)
    {
        if (y == 0)
        {
            printf("Centro del cuadrante");
        }
        else
        {
            printf("Fuera del centro");
        }
    }
...
```

Como mencionamos anteriormente también podemos usar las llaves para alterar la asociación normal del else de la siguiente forma:

```
...
if (x == 0)
    {
        if (y == 0)
        {
            printf("Centro del cuadrante");
        }
    }
    else
    {
        printf("Fuera del centro");
    }
...
```

Debido a las reglas de alcance de C, ahora el else no tiene conocimiento de la sentencia if(y...), ya que no están en el mismo bloque de código.

La sentencia switch

Muchas veces aplicar muchos if-else-if, puede ser poco elegante, además suele ser bastante difícil de seguir y hasta puede confundir incluso a su autor. Por estas razones, C incorpora una sentencia de decisión de ramificación múltiple denominada switch.

La forma general de la sentencia switch es:

```
...
switch (expresión)//Variable a comparar
    {
    case expresión-constante-1:
                              instrucción/es;
                              break;
    case expresión-constante-2:
                              instrucción/es;
              break;
    case expresión-constante-3:
              instrucción/es;
              break;
    case expresión-constante-n:
              instrucción/es;
              break;
default:
              instrucción/es;
    }
...
```

Características propias de un sentencia switch en C

La variable (expresión) a comparar debe ser un tipo ordinal (int o char), pero no puede ser ni double ni float ni cadena de caracteres.

La sentencia compara sucesivamente una variable con una correspondencia y a continuación se ejecuta una instrucción o un bloque de instrucciones.

Cuando el switch se ejecuta, una (y solo una) de las expresiones-constantes se selecciona y ejecuta.

Evalúa cada constante si no encuentra ninguna correspondencia se ejecuta la sentencia "default", que es opcional, si no aparece entonces no se lleva a cabo ninguna sentencia.

La sentencia switch se diferencia de la sentencia if en que switch solo puede comprobar la igualdad, mientras que if puede comparar expresiones lógicas y relacionales.

No puede haber dos constantes case con el mismo valor en el mismo switch.

La instrucción break en cada case es opcional. Se utiliza para finalizar la secuencia asociada a cada constante.

Si se omite la instrucción break, la ejecución continúa en la siguiente sentencia case hasta alcanzar otra instrucción break o el final del switch.

Ejemplo: Ingresar un numero del 1 al 7 que corresponde a un dia de la semana, luego imprimir el nombre correspondiente al numero del dia.

```
...
switch (dia)
    {
    case 1:
        printf("Lunes");
        break;
    case 2:
        printf("Martes");
        break;
    case 3:
        printf("Miercoles");
        break;
    case 4:
        printf("Jueves");
        break;
    case 5:
        printf("Viernes");
        break;
    case 6:
        printf("Sabado");
        break;
    case 7:
        printf("Domingo");
        break;
    default:
        printf("Numero de dia no valido");
    }
...
```

Cíclicas o repetitivas

También se pueden denominar sentencias de iteración, este tipo de sentencias permiten que un conjunto de instrucciones sea ejecutado una cierta cantidad de veces o hasta que se cumpla una determinada condición. Esta condición puede estar predefinida o no tener un final determinado de esta forma nos encontramos con dos tipos de bucles o ciclos repetitivos.

El bucle for

En C y en todos los lenguajes de programación modernos, las sentencias de iteración permiten que un conjunto de instrucciones sea ejecutado hasta que se alcance una cierta condición.

Este bucle se utiliza para realizar una acción un número determinado de veces es por esto que también se lo denomina estructura iterativa definida.

Este tipo de ciclo está compuesto de tres expresiones separadas por punto y coma (;) que indican cómo inicia el mismo, como controla su finalización y como se incrementa o decrementa.

La forma general o sintaxis de la sentencia for es:

```
...
for (<inicialización>;<condición>;<incremento>)
    {
        instrucción/es;
    }
...
```

El primer término dentro del paréntesis del for inicializa una variable (id) declarada previamente, a un valor inicial y es la variable de control del bucle, como pueden observar es una expresión de asignación.

El segundo término indica la condición de finalización del bucle, se evalúa dicha condición comparando la variable de control con un posible valor final utilizando los operadores relacionales ya vistos . Si esta condición es verdadera se ejecuta la instrucción o instrucciones. Se repite el bucle que finaliza cuando la expresión de control es falsa.

El tercer término indica el incremento o decremento que sufre la variable de control en cada paso del bucle. El incremento o decremento puede ser en una unidad o en más unidades.

La modificación de dicho valor de incremento tiene la forma id = id + numero pudiendo utilizar directamente los operadores de asignación id++ o id-- y sus variantes. Si se trata de un decremento, hay que tener la precaución de que el valor de inicio necesariamente sea mayor que el valor final, de lo contrario no se no se producirá la entrada al bucle.

Como también se mencionó en el uso del if las llaves pueden no ser obligatorias pero resultan útiles para marcar el bloque del ciclo que debe repetirse.

Ejemplo que muestra 10 veces la palabra "HOLA" por pantalla.

```
...
int i;//Variable de control del bucle
    for (i = 0; i < 10; i++)
    {
        printf("HOLA\n");
    }
...
```

El siguiente ejemplo utiliza un printf y un scanf para leer un dato desde el teclado. Note que en este caso se declara la variable en la estructura del ciclo. Esta variable funciona como variable local, es decir, nace y muere dentro del bucle, no es visible para el resto de la función.

```
...
//Variable de control del bucle declarada directamente en el
cuerpo del ciclo
int dato;
    for (int i = 0; i < 10; i++)
    {
        printf("Leer un valor\n");
        scanf("%d",&dato);
    }
...
```

Las llaves de cierre y apertura en este caso no son opcionales porque dentro del cuerpo del ciclo se ejecutan dos instrucciones, es decir que se trata de un bloque de instrucciones.

Variantes de ciclo for

Las formas anteriores mostraron una descripción de la forma más común del bucle for. Sin embargo, el lenguaje ofrece varias alternativas que aumentan su potencia, su flexibilidad y su aplicación en algunas situaciones particulares.

Una de las variantes es utilizar el operador coma (,) para permitir dos o más variables de control del bucle.

Veamos esta situación con un ejemplo en donde las variables de control en este caso son dos, x e y, ambas se inicializan en la estructura del for.

```
...
int x, y;
    for (x = 0, y = 0; x + y < 10; x++)
    {
        printf("HOLA\n");
        y = y + 2;
    }
...
```

Examinemos el mismo ejemplo en otra variante utilizando el separador coma(,) en la misma estructura:

```
...
int x, y;
    for (x = 0, y = 0; x + y < 10; x++, y += 2)
    {
        printf("HOLA\n");
    }
...
```

Otra alternativa que tenemos es utilizar operadores lógicos para ampliar la condición de corte.

El siguiente ejemplo muestra esta aplicación:

```
...
int x, y;
    for (x = 0, y = 0; x < 10 && y != 10; x++)
    {
        printf("HOLA\n");
        scanf("%d", &y);
    }
...
```

En este caso el bucle termina cuando la variable de control (x) llega a 10 o cuando el usuario ingresa un 10 como valor de la segunda variable de control (y).

Otra variante interesante de un ciclo for es que puede no tener todas las secciones de definición de la estructura del bucle.

Veamos el siguiente ejemplo donde se omite la expresión de inicio, y la variable de control se declara en inicializa en la zona de declaración de variables:

```
...
int x = 0; //Declara e inicializa la variable de control del bucle
    for (; x < 10; x++)
    {
        printf("HOLA\n");
    }
...
```

También se puede omitir la expresión de incremento. Observe el siguiente ejemplo donde la tercera sección del bucle está en blanco en este caso la variable de control se modifica dentro del bucle.

```
...
int x;
    for (x = 0; x < 10;)
    {
        printf("HOLA\n");
        x++;
    }
...
```

Una última variante en donde se omiten tanto la inicialización como el incremento de la variable de control:

```
...
    int x;
    x = 0;
    for (; x < 10;)
    {
        printf("HOLA\n");
        x++;
    }
...
```

El bucle While

El bucle o ciclo while se ejecuta repetidamente hasta que se alcanza una condición de fin de ciclo, o condición de salida. Por lo general no sabemos exactamente la cantidad de veces que se va a repetir el ciclo.

Dicho de otro modo la estructura while es aquella en la que el número de iteraciones no se conoce por anticipado es por ello que también se la puede denominar estructura iterativa indefinida.

En este caso el cuerpo del bucle se ejecuta mientras se cumple una determinada condición es decir que el proceso se repite mientras que la condición sea verdadera.

La sintaxis o forma general es:

```
while (<condición>)
{
      instrucción/es
}
```

El bucle while comienza por evaluar la condición, si es cierta, se ejecuta la instrucción o instrucciones, luego se vuelve a evaluar la expresión, si es verdadera, se vuelven a ejecutar las instrucciones. Este proceso continúa hasta que el resultado de evaluar la condición es falso.

Ejemplo:

```
...
int valor = 0;
    while (valor < 10)
    {
        printf("HOLA\n");
        valor = valor + 1;
    }
...
```

Como el bucle for, el ciclo while comprueba la condición al principio, lo que supone que el código del bucle puede que no se ejecute.

Es importante recalcar que en el bloque de instrucciones debe ocurrir algo para que la condición deje de ser verdadera. En la mayoría de los casos, la condición se hace falsa al cambiar el valor de una variable que fue inicializada fuera del ciclo o haciendo una lectura fuera del ciclo y rompiendola dentro del mismo.

Veamos otro ejemplo:

```
...
int a;
printf("Entrar al ciclo presione 1 para entrar, 0 para salir\n");
scanf("%d", &a)
while (a == 1)
{
      printf("HOLA\n");
      printf("Presione 1 seguir, 0 para salir\n");
      scanf("%d", &a);
}
...
```

El bucle do/while

Se comporta similar al bucle while, solo que la condición se evalúa al final del bucle. Por lo tanto, el bucle do-while se ejecuta al menos una vez.

La sintaxis o forma general del bucle do/while es:

```
do{
     instrucción/es;
}while (condición);
```

Aquí al igual que en las estructuras anteriores, aunque las llaves no son necesarias cuando hay una sola instrucción, se utilizan normalmente para evitar confusiones.

En este caso las instrucciones del cuerpo del bucle se repiten hasta que la condición se haga falsa, o dicho de otra forma, se ejecutan las sentencias mientras la condición sea cierta.

Ejemplo:

```
int num;
do
{
    printf("Ingrese un numero");
    scanf("%d", &num);
} while (num > 100);
```

En este ejemplo el usuario ingresa números desde el teclado hasta que encuentra un número menor o igual que 100.

El uso más común del ciclo do/while es generalmente en rutinas de menú de opciones o para la validación de datos.

Ejemplo:

```
 //validar que el usuario solo ingrese numeros del 0 al 9
    int digito;
    do
    {
        printf("Ingrese un numero\n");
        scanf("%d", &digito);
    } while (digito > 9);
```

Otro ejemplo:

```
...
int n, op;
do
{
      printf("\n   1. Calcular el doble de un numero.");
      printf("\n   2. Calcular la mitad de un numero.");
      printf("\n   3. Calcular el cuadrado de un numero.");
      printf("\n   4. Salir del menu.\n");
      printf("\n   Introduzca la opcion deseada (1-4): ");
      scanf("%d", &op);
} while (op != 1 && op != 2 && op != 3 && op != 4);
...
```