Sentencias y bloques

- Una sentencia es una acción a realizar, se indica el final de una sentencia con un punto y coma. Ejemplos:

```
x = 0;
i++;
printf("hola");
```

- Un bloque es un conjunto de sentencias que se agrupan. Se lo conoce también como **sentencia compuesta**. El modo de agrupar es con llaves. Ejemplo:

```
{
        x = 0;
        y = 2 * x;
}
```

- Puede haber una sentencia nula, se expresa con solamente un punto y coma.

Sentencias y expresiones

- En general dada una expresión, agregando ; al final la convierto en un sentencia

```
a = b + 2;


/* la asignación es una expresión cuyo valor es
 * el asignado. Al agregar ; es una sentencia */
```

- El operador , permite incluir más de una expresión en una misma sentencia

```
a = b = 2, c++;

/* 2 es asignado a b y su resultado (2)
 * es el asignado a, luego se incrementa c
 * ambas expresiones forman una sola sentencia */
```

Estructuras de control

- Sirven para modificar la ejecución secuencial
- Cuando indiquemos que en lugar de la estructura de control va una sentencia, la misma puede ser simple o compuesta (agrupada por llaves)

Selección

```
if (condición)
        sentencia por verdadero
else
        sentencia por falso
```

- El else es optativo
- Cada else se “acopla” al if más cercano

```
if (n > 0)
        if (a > b)
                z = a;
        else
                z = b;
```

- Si queremos asignar b a z solo si n no es positivo, agrupamos

```
if (n > 0) {
        if (a > b)
                z = a;
} else {
         z = b;
}
```

If-else if

Se puede encadenar if – else if de modo tal de lograr una selección por varios casos

```
if (condición1)
        sentencia1
else if (condición2)
        sentencia2
else if (condición3)
        sentencia3
else
        sentencia por casos restantes
```

Ejemplo

```
if (n < 0)
        z = a;
else if (n < 100)
        z = b;
else
        z = c;
```

Operador ? :

Es un operador ternario y el valor que devuelve depende de la condición:

```
condición ? valor-por-verdadero : valor-por-falso
```

Ejemplos:

```
x = n > 100 ? y : z;
```

Equivale a :

```
if (n > 100)
        x = y;
else
        x = z;
```

Selección múltiple

```
switch (expresión) {
case etiqueta1:
        sentencias1
       break;
case etiqueta2:
        sentencias2
        /* Sin break continúa la ejecución con el código
           a continuación de la siguiente etiqueta*/
default:
        sentencias caso contrario
}
```

Ejemplo

```
switch (n) {
case 1:
        z = a;
       break;
case 2:
        z = b;
       break;
default:
        z = c;
}
```

Ciclo mientras

Itera cero o más veces, eventualmente infinitas

```
while (condicion sea verdadera)
        Sentencias
```

Ejemplo

```
while (n > 0) {
        printf("n = %d\n", n);
        n--;
}
```

Ciclo hacer mientras

Itera una o más veces, eventualmente infinitas

```
do {
        sentencias
} while (condicion sea verdadera);
```

Ejemplo

```
do {
        printf("n = %d\n", n);
        n--;
} while (n);   /* se detiene cuando n valga cero */
```

Ciclo para

Forma general

```
for (sentencias-antes; condición; sentencias-finales) {
        Sentencias;
}
```

Es equivalente a

```
sentencias-antes
while (condición) {
        Sentencias
        sentencias-finales
}
```

Ciclo para: ejemplos

```
for(i = 0; i < 10; i++)
        printf("i = %d\n", i);
//------>>
i = 0;
while (i < 10) {
        printf("i = %d\n", i);
        i++;
}
////////////////////////////////////////
for(n = 100,j = 2; j < n; j += 2) {
        printf("j = %d\tn = %d\n", j, n);
        n -= j;
}
//------>>
n = 100;
j = 2;
while (j < n) {
        printf("j = %d\tn = %d\n", j, n);
        n -= j;
        j += 2;
}
```

break, continue y goto

- **break** sirve para salir del bloque **switch**, **for** ,**while** o **do while** que lo encierra y seguir con la siguiente instrucción al bloque en que se encuentra
- **continue** se usa en ciclos **for** ,**while** y **do while**. Su efecto es saltear el resto del ciclo y volver a chequear la condición del ciclo para o bien comenzar una nueva iteración o bien darlo por terminado
- Ambos son goto “disfrazados” pero que “encajan” en las normas de programación estructurada.
- **goto** se utiliza como un salto incondicional. Está “mal visto” pero a veces (muy raras veces) es conveniente. Se utiliza con la forma goto etiqueta; y en algún lado del programa figura etiqueta: (similar a los case del switch)

Ejemplo break y continue

```
while (i < TOTAL) {
        /* codigo que procesa los datos */

        /*supongamos que -1 significa que frenar el proceso
          aunque i le falta para llegar a TOTAL */
        if (indicador == -1)
                break; // salta al printf de abajo

        /* supongamos que 10 significa que para esta iteracion
           no se requieren los procesos adicionales */
        if (indicador == 10)
                continue; // salta a while (i < TOTAL)

        /* otros procesos adicionales */
}
printf("proceso terminado\n");
```

Ejemplo goto

```
promedio = 0;
if (cantidad == 0)
        goto final; //mal uso, pero para mostrar como funciona

promedio = sumatoria / cantidad;

final:
printf("Promedio = %f\n", promedio);
```

Licencia