C: Estructuras de control y tomas de decisión.

[Se supone que llegará un día en el que tendremos unos formatos fijos. Por favor, todo tipo de dudas o errores o imprecisiones o lo que sea, hacedlo saber]

Se tratarán aquí las estructuras de control y tomas de decisión, que nos permitirán establecer a nuestro antojo -o al antojo de nuestro algoritmo, más bien- el desarrollo de nuestro programa:

Tomas de decisión

- if...else
- switch ... case

Estructuras de control

- for ...
- while ...
- do ... while

Tomas de decisión

if ... then ... else

Esta estructura se adapta especialmente al caso más simple de una toma de decisión. Si se cumple una determinada condición, haz esto; en caso contrario, haz esto otro (o no hagas nada).

Ejemplo:
Imaginemos que queremos restringir la entrada a un aula a todo aquel que no sea de tercer curso, por ejemplo. Tendremos una variable que nos indique el curso al que pertenece el alumno. Una variable que, por ejemplo, podremos llamar **curso**.

```
if (curso == 3) {
  printf("El muchacho es de tercero. Puede pasar\n");
}
else {
  printf("Lo siento, no puede pasar porque su curso es %d\n",curso);
}
```

Vayamos por partes. En primer lugar vemos nuestro **if**, objeto de nuestro estudio. A continuación, y entre paréntesis, la expresión **curso == 3** (observad que uno de los errores más normales, y no necesariamente de principiantes, es poner un signo = en lugar de los dos ==; si se pone uno solo, se estará realizando una asignación (dándole a curso el valor 3), no una comparación. Esta expresión es evaluada y, en el caso de resultar cierta (lo que en C equivale a dar un valor distinto de cero) se entra en el cuerpo del if (el espacio entre llaves). En el caso de ser falsa, pasará a ejecutar el código entre las llaves que siguen al **else**.

Por supuesto, no es necesario que exista la cláusula **else**. Por ejemplo, podremos hacer algo como:

```
if (curso < 2) {
  print "Esto no es para bebés\n";
  print "Sal del programa ipso facto\n";
}

... continuación del programa.
```

Podemos crear expresiones más complejas. Por ejemplo, podemos dejar pasar a aquel cuyo curso sea mayor (**>**) que 3 y (**&&**) menor (**<**) que 6:

```
if ( (curso > 3) && (curso < 6) ) {
 ...
}
```

switch ... case ...

Existen ocasiones en las que queremos tomar diferentes acciones según el valor de una determinada expresión o variable. Por ejemplo, supongamos que queremos imprimir una determinada expresión según nuestra personalidad. Tenemos una función hipotética **Somos**, que devolverá lo que somos (unas constantes predefinidas, por ejemplo, en un **enum**).

```
somos=Somos();
if (somos==ELFO) {
 print "Que bien se esta en este bosquecillo\n";
}
else if (somos==GANDALF) {
 print "He de partir hacia el este\n";
}
else if (somos==SAM) {
 print "Tanta tranquilidad es sospechosa\n";
}
else if (somos==GOLLUM) {
 print "Mi tessssssssoro\n";
}
```

Bueno, un poco engorroso, ¿no es cierto?. Para esto tenemos la expresión que nos ocupa:

```
somos=Somos();
switch (somos) {
 case ELFO:
   print "Que bien se esta en este bosquecillo\n";
   break;
 case GANDALF:
   print "He de partir hacia el este\n";
   break;
 case SAM:
   print "Tanta tranquilidad es sospechosa\n";
   break;
 case GOLLUM:
   print "Mi tessssssssoro\n";
   break;
 default:
   print "Nada";
   break;
}
```

En los paréntesis que siguen al **switch** situamos la expresión a evaluar y, entre llaves, ponemos un conjunto de **case**, a continuación de los cuales va la constante correspondiente.
El **default** se utiliza si es necesario tener en cuenta la posibilidad de que se de algún valor no contemplado, y los **break** son necesarios para que el programa salga del **switch**; en otro caso, seguiría realizando comparaciones.

Estructuras de control

while ...

En ocasiones se necesita realizar algo mientras (**while**) se cumpla una determinada condición. Recordemos, una vez más, que para el lenguaje C, cumplirse significa ser distinto de cero. Vamos, que una expresión tan sencilla como **4** se cumple siempre, y la expresión **0** no se cumple nunca.

Por ejemplo, vamos a lanzar misiles hasta ganar la guerra.

```
fin=0; /* Inicializacion de la variable */

while (!fin) {  /* Mientras no acabamos */
  if (GuerraGanada()) {
    fin=1;
  }
}
```

El cierre de admiración antepuesto a cualquier expresión, la niega. Es decir, si **fin** es falso, **!fin** es verdadero.

do ... while

Esta expresión es semejante a la **while ...**. Hace algo mientras se cumpla una determinada condición pero, a diferencia de la expresión anterior, lo realiza al menos una vez. (Obsérvese que, en el ejemplo anterior, si **fin** es **0**, no entra en el bloque).

```
do {
... /* Este bloque se realiza al menos una vez */
} while (!fin); /* Mientras no acabemos */
```

for ...

Se emplea esta expresión generalmente -no siempre- cuando queremos realizar una determinada acción un número prefijado de veces. Su sintaxis es:

```
for ( [INICIALIZACION]; [CONDICION PARA SEGUIR EN EL BUCLE] ; [ACCIONES EN CADA
EJECUCION DEL BUCLE] ) { ... }
```

Por ejemplo, si queremos que un programa nos imprima la tabla del 9:

```
for (i=1; i<=10; i++) {
  print ("9 x %d es %d\n",i,9*i);
}
```

Aquí comenzamos el bucle dando a **i** el valor de inicialización **1**. No hay inconveniente en inicializar varias variables a la vez, separándolas por comas: **i=1, m=9**. Para seguir en el bucle se ha de cumplir la condición **i<=10** y, a cada paso del bucle, incrementamos en una unidad la variable **i**.

Conseguimos el mismo efecto haciendo lo siguiente:

```
for (i=1,m=9;i<=10;m+=9,i++)
 print ("9 x %d es %d\n",i,m);
```

Aquí, por un lado vemos que no son necesarias las llaves cuando solamente hay una línea en el cuerpo del bucle. Inicializamos dos variables (**i=1,m=9**), entramos en el bucle mientras **i<=10** y, a cada paso, incrementamos **m** en 9 e **i** en una unidad.