(Unidad 4) Programación estructurada en C

(4.1) expresiones lógicas

Hasta este momento nuestros programas en C apenas pueden realizar programas que simulen, como mucho, una calculadora. Lógicamente necesitamos poder elegir qué cosas se ejecutan según unas determinada circunstancias.

Todas las sentencias de control de flujo se basan en evaluar una expresión lógica. Una expresión lógica es cualquier expresión que pueda ser evaluada con verdadero o falso. En C (o C++) se considera verdadera cualquier expresión distinta de 0 (en especial el uno, valor **verdadero**) y falsa el cero (**falso**).

(4.2) sentencia *if*

(4.2.1) sentencia condicional simple

Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de sentencias en caso de que la expresión lógica sea verdadera. Su sintaxis es:

```
if(expresión lógica) {
   sentencias
}
```

Si sólo se va a ejecutar una sentencia, no hace falta usar las llaves:

```
if(expresión lógica) sentencia;
```

Ejemplo:

```
if(nota>=5){
   printf("Aprobado");
   aprobados++;
}
```

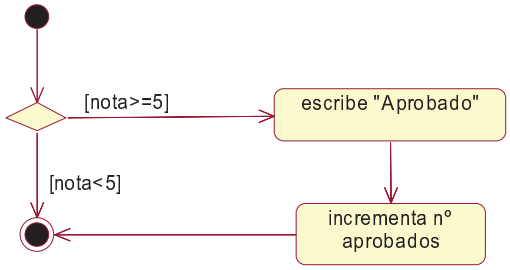


Ilustración 7, Diagrama de actividad del ejemplo anterior

(4.2.2) sentencia condicional compuesta

Es igual que la anterior, sólo que se añade un apartado **else** que contiene instrucciones que se ejecutarán si la expresión evaluada por el **if** es falsa. Sintaxis:

```
if(expresión lógica){
   sentencias
}
else {
   sentencias
}
```

Las llaves son necesarias sólo si se ejecuta más de una sentencia. Ejemplo:

```
if(nota>=5){
   printf("Aprobado");
   aprobados++;
}
else {
   printf("Suspensos");
   suspensos++;
}
```

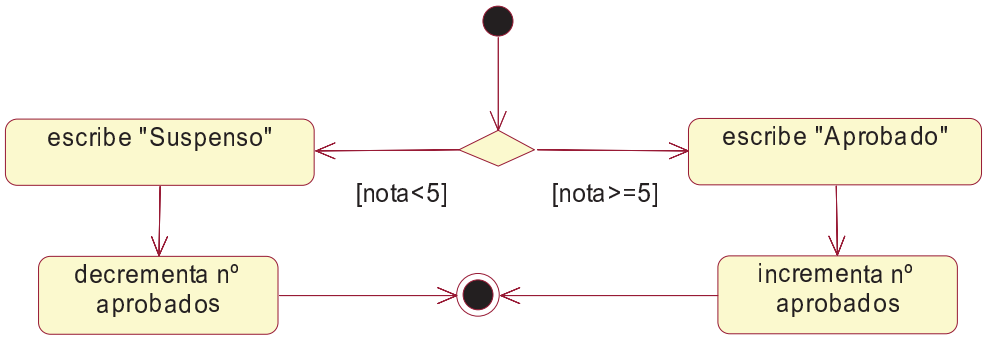


Ilustración 8, diagrama UML de actividad del ejemplo anterior

(4.2.3) anidación

Dentro de una sentencia **if** se puede colocar otra sentencia **if**. A esto se le llama **anidación** y permite crear programas donde se valoren expresiones complejas.

Por ejemplo en un programa donde se realice una determinada operación dependiendo de los valores de una variable, el código podría quedar:

```
if (x==1) {
   sentencias
   …
}
else {
   if(x==2) {
         sentencias
         …
   }
   else {
         if(x==3) {
               sentencias
               …
         }
   }
}
```

Pero si cada **else** tiene dentro sólo una instrucción **if** entonces se podría escribir de esta forma (que es más legible), llamada ***if-else-if***:

```
if (x==1) {
   instrucciones
   …
}
else if (x==2) {
   instrucciones
   …
}
else if (x==3) {
   instrucciones
   …
}
```

(4.3) sentencia *switch*

Se trata de una sentencia que permite construir alternativas múltiples. Pero que en el lenguaje C está muy limitada. Sólo sirve para evaluar el valor de una variable entera (o de carácter, ***char***).

Tras indicar la expresión entera que se evalúa, a continuación se compara con cada valor agrupado por una sentencia **case.** Cuando el programa encuentra un ***case*** que encaja con el valor de la expresión se ejecutan todos los ***case*** siguientes. Por eso se utiliza la sentencias **break** para hacer que el programa abandone el bloque ***switch.*** Sintaxis***:***

```
switch(expresión entera){
   case valor1:
        sentencias
        break; /*Para que programa salte fuera del switch
                 de otro modo atraviesa todos los demás
                 case */
   case valor2:
        sentencias
   ...
   default:
        sentencias
}
```

Ejemplo:

```
switch (diasemana) {
  case 1:
       printf(”Lunes”);
       break;
  case 2:
       printf(”Martes”);
       break;
  case 3:
       printf(”Miércoles”);
       break;
  case 4:
       printf(”Jueves”);
       break;
  case 5:
       printf(”Viernes”);
       break;

case 6:
       printf(”Sábado”);
       break;
  case 7:
       printf(”Domingo”);
       break;
  default:
       std::cout<<”Error”;
}
```

Sólo se pueden evaluar expresiones con valores concretos (no hay una ***case >3*** por ejemplo). Aunque sí se pueden agrupar varias expresiones aprovechando el hecho de que al entrar en un case se ejecutan las expresiones de los siguientes.

Ejemplo:

```
switch (diasemana) {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
       printf("Laborable");
       break;
  case 6:
  case 7:
       printf("Fin de semana");
       break;
  default:
       printf("Error");
}
```

(4.4) bucles

A continuación se presentan las instrucciones C que permiten realizar instrucciones repetitivas (bucles).

(4.4.1) sentencia *while*

Es una de las sentencias fundamentales para poder programar. Se trata de una serie de instrucciones que se ejecutan continuamente mientras una expresión lógica sea cierta.

Sintaxis:

```
while (expresión lógica) {
  sentencias
}
```

El programa se ejecuta siguiendo estos pasos:

(1) Se evalúa la expresión lógica

(2) Si la expresión es verdadera ejecuta las sentencias, sino el programa abandona la sentencia **while**

(3) Tras ejecutar las sentencias, volvemos al paso 1

Ejemplo (escribir números del 1 al 100):

```
int i=1;
while (i<=100){
  printf("%d",i);
  i++;
}
```

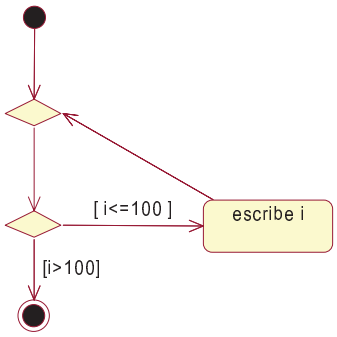


Ilustración 9, diagrama UML de actividad del bucle anterior

(4.4.2) sentencia do..while

La única diferencia respecto a la anterior está en que la expresión lógica se evalúa después de haber ejecutado las sentencias. Es decir el bucle al menos se ejecuta una vez. Es decir los pasos son:

(1) Ejecutar sentencias

(2) Evaluar expresión lógica

(3) Si la expresión es verdadera volver al paso 1, sino continuar fuera del while

Sintaxis:

```
do {
  sentencias
} while (expresión lógica)
```

Ejemplo (contar del 1 al 1000):

```
int i=0;
do {
  i++;
  printf("%d",i);
} while (i<=1000);
```

(4.4.3) sentencia *for*

Se trata de un bucle especialmente útil para utilizar contadores. Su formato es:

```
for(inicialización;condición;incremento){
   sentencias
}
```

Las sentencias se ejecutan mientras la condición sea verdadera. Además antes de entrar en el bucle se ejecuta la instrucción de inicialización y en cada vuelta se ejecuta el incremento. Es decir el funcionamiento es:

(1) Se ejecuta la instrucción de inicialización

(2) Se comprueba la condición

(3) Si la condición es cierta, entonces se ejecutan las sentencias. Si la condición es falsa, abandonamos el bloque ***for***

(4) Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y se vuelve al paso 2

Ejemplo (contar números del 1 al 1000):

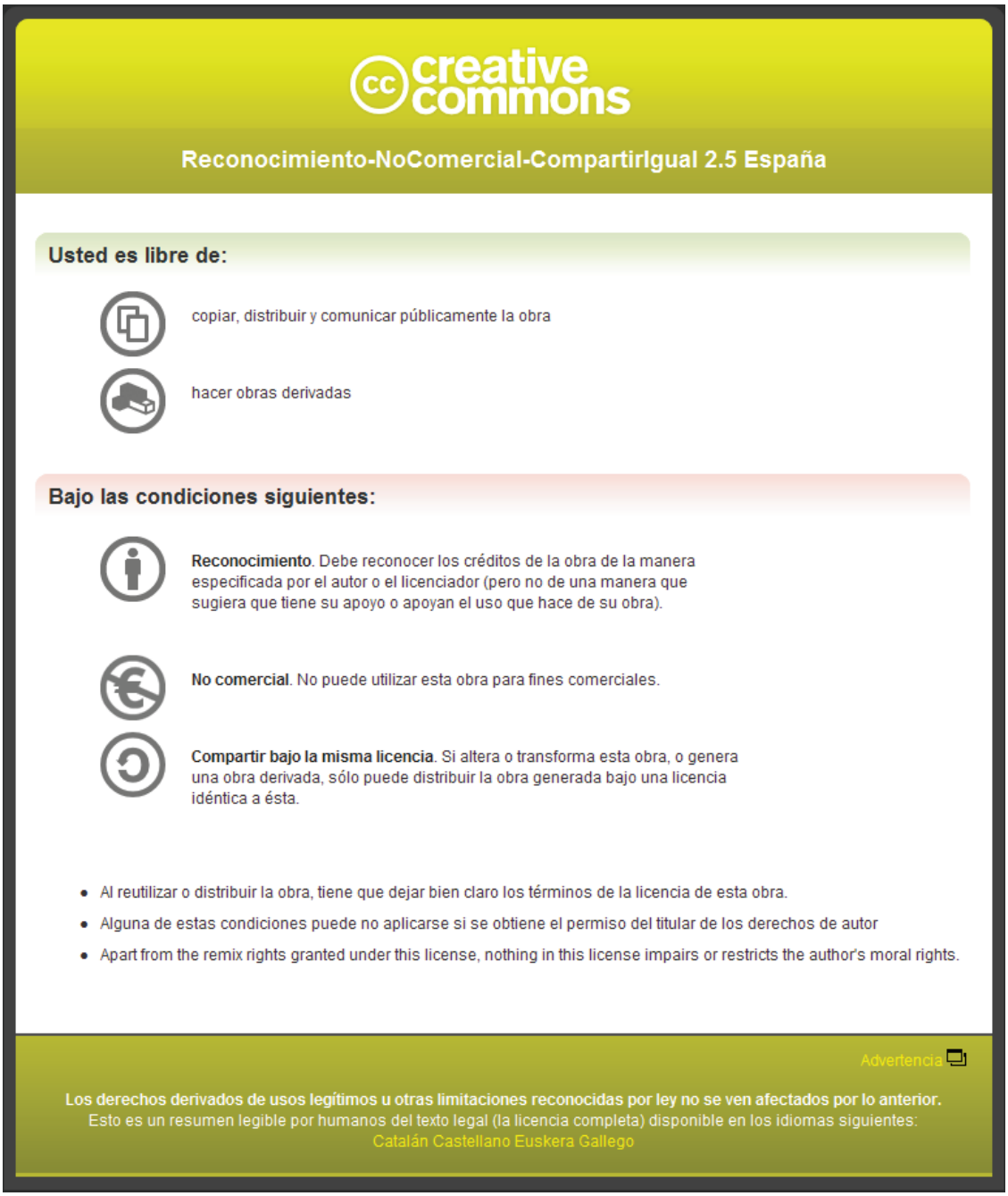
```
for(int i=1;i<=1000;i++){
   printf("%d",i);
}
```

La ventaja que tiene es que el código se reduce. La desventaja es que el código es menos comprensible. El bucle anterior es equivalente al siguiente bucle **while:**

```
i=1; /*sentencia de inicialización*/
while(i<=1000) { /*condición*/
   printf("%d",i);
   i++; /*incremento*/
}
```

(4.5) sentencias de ruptura de flujo

No es aconsejable su uso ya que son instrucciones que rompen el paradigma de la programación estructurada. Cualquier programa que las use ya no es estructurado. Se comentan aquí porque en algunos listados de código puede ser útil conocerlas.



Para ver una copia completa de la licencia, acudir a la dirección
http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es