

Lenguaje C, segundo bloque: Sentencias de control

José Otero

¹Departamento de informática
Universidad de Oviedo

28 de octubre de 2008

Índice

- 1 Sentencia if
- 2 Sentencia if-else
- 3 if e if-else anidados
- 4 Sentencia switch
 - Ejemplo: calculadora sencilla
- 5 Bucle while
 - Ejemplo: suma de las cifras de un número
- 6 Bucle do-while
 - Ejemplo: aproximación de una raíz por una sucesión
- 7 Bucle for
 - Ejemplo: cálculo del factorial

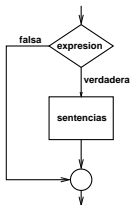
Permite ejecutar o no una sentencia o bloque, en función de si una expresión es cierta o no.

■ Una sentencia:

```
if (expresion)
    sentencia;
```

■ Un bloque:

```
if (expresion)
{
    //bloque
    ...
}
```



"expresion" se construye con operadores lógicos y relacionales.

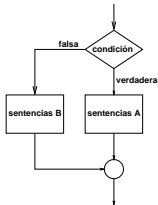
Ejemplo: ordenar dos números de mayor a menor.

- Si están desordenados hay que intercambiarlos.
- Si están ordenados no hay que hacer nada.

```
#include<stdio.h>
int main()
{
    int a,b,tmp;
    scanf("%d%d",&a,&b);
    //ordenar dos numeros
    //de mayor a menor
    if (a<b)
    {
        tmp=a;
        a=b;
        b=tmp;
    }
    printf("\n%d %d",a,b);
}
```

Permite ejecutar una sentencia/bloque u otra sentencia/bloque, en función de si una expresión es cierta o no.

- `if (expresion)`
 sentencia;
- `else`
 sentencia;
- `if (expresion)`
 {
 //bloque
 ...
 }
 `else`
 sentencia;
- Etc.

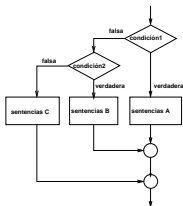


Ejemplo: mostrar el mayor de dos números.

```
#include<stdio.h>
int main()
{
  int a,b,mayor;
  printf("\nIntroduce dos numeros:");
  scanf("%d%d",&a,&b);
  if (a>b)
    mayor=a;
  else
    mayor=b;
  printf("\nEl mayor es %d",mayor);
}
```

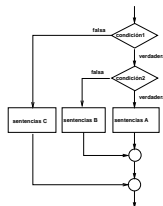
Entre las sentencias después de `if` o `else` puede haber más `if` e `if-else`. Por el lado del `else`:

```
if (condicion1)
{
  ...
}
else
  if (condicion2)
  {
    ...
  }
  else
  {
    ...
  }
}
```



Entre las sentencias después de `if` o `else` pueden haber más `if` e `if-else`. Por el lado del `if`:

```
if (condicion1)
{
  if (condicion2)
  {
    ...
  }
  else
  {
    ...
  }
}
else
  ...
```



Entre las sentencias después de `if` o `else` pueden haber más `if` e `if-else`. Por los dos lados:

```
if (condicion1)
{
    if (condicion2)
    {
        ...
    }
else
{
    ...
}
}

else
{
    if (condicion3)
    {
        ...
    }
else
{
    ...
}
}
```

Es equivalente a varios `if-else` restringiendo la condición a la comparación a igualdad entre `expresion` y `cte1...cte2`.

```
switch(expresion)
{
    case cte1:...
        break;
    case cte2:...
        break;
    ...
    default:...
}

if (expresion==cte1)
{
    ...
}
else
    if (expresion==cte2)
    {
        ...
    }
    else
        ...
```

IMPORTANTE:

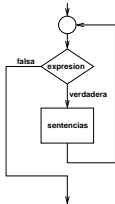
- Si se omite `break` se ejecuta todo el código que siga hasta encontrar el siguiente.
- Expresión es de tipo entero o carácter.
- Después de `case` solo pueden ir constantes de esos tipos.
- La condición es, implícitamente, la comparación a igualdad entre `expresion` y las constantes. No se puede hacer otro tipo de comparación.

```
#include<stdio.h>
int main()
{
    float a,b; char op;
    printf("\nIntroduce num op num:");
    scanf("%f%c%f",&a,&op,&b);
    switch(op)
    {
        case '+':printf("\n%f+%f=%f",a,b,a+b);
                break;
        case '-':printf("\n%f-%f=%f",a,b,a-b);
                break;
        case '*':printf("\n%f*%f=%f",a,b,a*b);
                break;
        case '/':printf("\n%f/%f=%f",a,b,a/b);
                break;
        default:printf("\nerror");
    }
}
```

Repite una sentencia o bloque mientras sea cierta una expresión. *Primero* se evalúa la expresión, si es cierta se ejecutan las sentencias.

```
while(expresion)
    sentencia;
```

```
while(expresion)
{
    //bloque
    ...
}
```



IMPORTANTE:

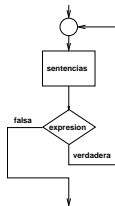
- La primera vez que se evalúe *expresion* tienen que tener valor definido todas las variables que intervengan en ella. De momento, tener valor definido es:
 - Que se haya pedido por el teclado.
 - Que haya estado a la izquierda de una asignación.
 - Esto tiene que hacerse *antes* del bucle.
- Si la primera vez que se evalúa *expresion* es falsa, no se realiza ninguna iteración.
- De entre las sentencias que se repiten, algunas de ellas tienen que modificar alguna variable de las que conforman *expresion*, haciéndola falsa en algún momento.
 - De lo contrario el bucle no termina.

```
#include<stdio.h>
int main()
{
    //declaracion de variables, inicializar suma
    int cifra,n,suma=0;
    printf("\nIntroduce un entero:");
    scanf("%d",&n);
    //mientras n!=0 quedan cifras por extraer
    while(n!=0)
    {
        //se extrae la cifra
        cifra=n%10;
        //se suma la cifra
        suma=suma+cifra;
        //se elimina la cifra del numero
        n=n/10;
    }
    printf("\nSuma de las cifras:%d",suma);
}
```

Repite una sentencia o bloque mientras sea cierta una expresión. *Primero* se ejecutan las sentencias, después se evalúa la expresión.

```
do
    sentencia;
while(expresion);

do
{
    //bloque
    ...
}while(expresion);
```



IMPORTANTE:

- La primera vez que se evalúe *expresion* tienen que tener valor definido todas las variables que intervengan en ella. De momento, estar definida es:
 - Que se haya pedido por el teclado.
 - Que haya estado a la izquierda de una asignación.
 - Esto *puede* hacerse durante la primera iteración.
- Como primero se ejecutan las sentencias, *al menos* se realiza una iteración.
- De entre las sentencias que se repiten, algunas de ellas tienen que modificar alguna variable de las que conforman *expresion*, haciéndola falsa en algún momento.
 - De lo contrario el bucle no termina.

```
#include<stdio.h>
#include<math.h>
int main()
{
    float x_i,x_i_1=.5,a,dif,precision;
    printf("\nIntroduce un num y precision");
    scanf("%f%f",&a,&precision);
    do{
        //siguiente termino
        x_i=.5*(x_i_1+a/x_i_1);
        //diferencia entre dos consecutivos
        dif=fabs(x_i_1-x_i);
        //actualizacion del anterior
        x_i_1=x_i;
        //mientras la dif mayor que precision
    }while(dif>precision);
    printf("\nRaiz cuadrada=%f",x_i);
}
```

Es una forma compacta de escribir el bucle `while` para ciertos usos, por ejemplo para realizar un número fijo de iteraciones.

```
for(inicializacion;expresion;actualizacion)
    sentencia;
```

```
for(inicializacion;expresion;actualizacion)
{
    //bloque
    ...
}
```

- La inicializacion se hace una sola vez.
 - Consiste en al menos una inicialización de las variables que intervienen en *expresion*.
- Después se evalúa *expresion*, si es cierta, se ejecutan las sentencias.
- Finalmente se realiza la actualizacion.
 - Consiste en al menos una asignación en las que se cambia el valor de alguna variable de las que intervienen en *expresion*.

Equivale a un `while` con la siguiente estructura.

```

inicializacion;
while(expresion)
{
    sentencias;
    actualizacion;
}
    
```



- Si hay varias inicializaciones se separan por comas.
- Todas las variables de `expresion` deben de tener valor definido la primera vez que se evalúe.
- Si hay varias actualizaciones se separan por comas.
 - Algunas deben hacer que `expresion` se haga falsa. De lo contrario el bucle no termina.
- En este curso casi siempre será suficiente una inicialización y una actualización.

```

#include<stdio.h>
int main()
{
    int n,fact=1,i;
    printf("\nIntroduce un numero:");
    scanf("%d",&n);
    for (i=2;i<=n;i++)
        fact=fact*i;
    printf("\n%d!=%d",n,fact);
}
    
```