Laboratorio de Arquitectura de Redes

Sentencias de control en lenguaje C

Sentencias de control en lenguaje C

- Introducción
- Sentencia `if-else`
- Sentencia `switch`
- Sentencia `while`
- Sentencia `do-while`
- Sentencia `for`
- Sentencia `break`
- Sentencia `continue`
- Sentencia `return`
- Sentencia `goto`

Introducción

- Las **sentencias de control** determinan el *flujo de ejecución* del programa
- Pueden ser
  - De **selección** o **condicionales**: Permiten la toma de decisiones en tiempo de ejecución
    - `if else` y `switch`
  - De **iteración** o **bucles**: Permiten la repetición de operaciones
    - `for`, `while` y `do-while`
  - De **salto**: Permiten programar la ruptura de la secuencia del programa
    - `continue`, `break`, `return` y `goto`
  - De **etiquetado**: Permiten identificar puntos en el programa
    - `case`, `default` y `«etiqueta:»`

Sentencia `if-else` (I)

```
if (expresion) sentencia1;
else sentencia2;
```

- Si `expresion` resulta verdadera, se procesa `sentencia1`
- Si `expresion` es falsa, se procesa `sentencia2`
- `expresion` es verdadera si tiene un valor distinto de cero
- En cualquier caso se ejecuta una de las dos sentencias
- `else sentencia2;` es opcional

Sentencia `if-else` (II)

- `sentencia1` y `sentencia2` pueden sustituirse por bloques de sentencias entre llaves

```
if (expresion)
{
    /* Bloque de sentencias 1 */
}
else
{
    /* Bloque de sentencias 2 */
}
```

- Las llaves pueden permitir el agrupamiento de sentencias `if-else` de forma adaptada a las necesidades de cada caso

Sentencia if-else (III)

```
if (expresion1)
{
   if (expresion2)
      if (expresion3) sentencia31;
      else sentencia32;
}
else sentencia2;
```

- sentencia31 se procesa si son ciertas expresion1, expresion2 y expresion3
- sentencia32 se procesa si son ciertas expresion1 y expresion2 y falsa expresion3
- sentencia2 se procesa si expresion1 es falsa, sin tomar en consideración expresion2 ni expresion3

Sentencia `if-else` (IV)

- **Sentencias `if-else` anidadas**
  - Como bloque de sentencias aparece una nueva sentencia `if-else` completa o no
  - La presencia de llaves determina las agrupaciones de forma prioritaria entre los `if` y los `else`
  - Si no hay llaves
    - Cada `else` se asocia al `if` más próximo
    - Cada sentencia (o bloque de sentencias) se ejecuta independientemente de las demás

```
if (expresion1) sentencia1;
else if (expresion2) sentencia2;
else if (expresion3) sentencia3;
...
else if (expresionN) sentenciaN;
else sentenciaN+1;
```

  - `sentenciaN` representa a una sentencia o a un bloque de sentencias que se ejecuta sólo si `expresionN` es cierta
  - `sentenciaN+1` se procesa sólo si no se ha procesado ninguna de las anteriores

Sentencia switch (I)

```
switch (expresion)
{       /* Llave obligatoria */
    case expresionconstante1:
        /* Secuencia de sentencias 1 */
        break;
    case expresionconstante2:
        /* Secuencia de sentencias 2 */
        break;
    ...
    case expresionconstanteN:
        /* Secuencia de sentencias N */
        break;
    default:
        /* Secuencia de sentencias N+1 */
        break;
}       /* Fin de la sentencia swtich. Llave obligatoria */
```

Sentencia `switch` (II)

- La sentencia **`swtich`** es una sentencia de selección múltiple que busca la igualdad entre una expresión y una expresión *constante de tipo entero*
  - Las llaves son obligatorias, forman parte de la sentencia
  - El número de `case` no está limitado
  - El `default` es opcional
  - Las sentencias `break` aislan las sentencias de cada `case`
- Se evalúa `expresion` y se comprueba si es igual a alguna constante asociada a un `case`
  - Si se encuentra la igualdad, se ejecutan **todas** las sentencias hasta encontrar la llave de *fin del `switch`* o hasta encontrar un *`break`*
  - Si no se encuentra la igualdad, se ejecutan las sentencias del *`default`* (si existe, ya que es opcional) hasta la llave de *fin del `switch`* o hasta encontrar un *`break`*
  - Las *secuencias de sentencias* asociadas a cada case no son bloques de sentencias si no van entre llaves

Sentencia `while`

```
while (expresion) sentencia;

while (expresion)
{
   sentencia; /* Bloque de sentencias */
}
```

- Si `expresion` resulta verdadera, se procesa `sentencia`
- Tras la ejecución, vuelve a repetirse la comprobación de `expresion`
- Si `expresion` resulta falsa, continua la ejecución del programa en la siguiente sentencia tras este `while`
- Si `expresion` no cambia de valor como consecuencia de la ejecución sucesiva, puede crearse un bucle infinito

Sentencia do-while

```
do sentencia;
while (expresion);

do
{
   sentencia; /* Bloque de sentencias */
} while (expresion);
```

- Tras ejecutar `sentencia`, se evalúa `expresion` y, si resulta verdadera, se vuelve a procesar `sentencia`
- Si `expresion` resulta falsa, continua la ejecución del programa en la siguiente sentencia tras este `while`
- Si `expresion` no cambia de valor como consecuencia de la ejecución sucesiva, puede crearse un bucle infinito

Sentencia for

```
for (inicializacion ; condicion ; progresion)
   sentencia;

for (inicializacion ; condicion ; progresion)
{
   sentencia; /* Bloque de sentencias */
}
```

- inicialización es una sentencia (o conjunto de sentencias separadas por comas) de asignación de valor a una o más variables
- condición es la expresión que, si resulta cierta hace que se procese sentencia, si resulta falsa finaliza el bucle
- progresion representa una o más sentencias de asignación que se ejecutan siempre al finalizar sentencia (o el bloque de sentencias) y antes de evaluar nuevamente condicion

Sentencia `break`

- **`break`** es la sentencia que permite finalizar de inmediato las sentencias `switch`, `while`, `do-while` o `for`, independientemente de otras condiciones
- En caso de anidamiento de bucles o sentencias, la sentencia `break` detiene el bucle o sentencia más interno en el que se encuentra

Sentencia `continue`

- **`continue`** es la sentencia que fuerza una nueva iteración del bucle en el que se encuentre, saltando las sentencias que se encuentren entre su posición y el final del bucle
  - En los bucles `while` y `do-while`, salta a evaluar la condición del bucle
  - En los bucles `for`, salta a ejecutar la progresión y, posteriormente a comprobar la condición
- En caso de anidamiento de bucles o sentencias, la sentencia `break` detiene el bucle o sentencia más interno en el que se encuentra

Sentencia `return`

- **`return`** es la sentencia que finaliza una función, devolviendo el control del programa al punto en el que fue llamada la función
- Sintaxis

  `return expresion;`
  - El término `expresión` representa a una expresión válida en lenguaje C cuyo valor será *devuelto por la función al punto del programa en el que fue llamada*.
    - Debe ser del tipo declarado en la función
- La llave de cierre o finalización de una función «`}`» es equivalente a esta sentencia sin `expresion`
- La sentencia `return` sin el término `expresion` se utiliza en funciones que no devuelven ningún valor
- Puede aparecer en una función tantas veces como sea preciso

Sentencia goto

- **La sentencia goto** es la sentencia de salto incondicional
- Está totalmente desaconsejada en programación estructurada
- Sintaxis

```
...
etiqueta:
...
goto etiqueta;
...
```

  - «etiqueta:» es un identificador de línea.
  - Puede encontrarse en cualquier parte del programa
  - Será del destino del salto del flujo de ejecución del programa tras la ejecución de la sentencia goto