

Costrutti di selezione

Stefano Ferrari

Università degli Studi di Milano
stefano.ferrari@unimi.it

Programmazione

anno accademico 2016–2017

Selezione

Selezione: date un'espressione logica e una o più attività di processo (istruzioni o blocchi)

- ▶ si valuta l'espressione logica
- ▶ si esegue al massimo una delle attività

Il caso più semplice si esprime con il **costrutto if**

if (*espressione*)
istruzione

Si valuta il valore dell'espressione, che deve essere di tipo logico

- ▶ se è *vero*, si esegue l'istruzione
- ▶ se è *falso*, si passa oltre

Valori logici

Vale la pena di ricordare che **in C non esistono valori logici**.

Gli operatori logici

- ▶ interpretano 0 come falso e ogni valore diverso da zero come vero
(se *i* è una variabile intera, ha senso scrivere `if (i)`)
- ▶ restituiscono 0 per falso, 1 per vero
(quindi `(3 < 6)` vale 1, mentre `(3 > 6)` vale 0)

Questo porta a **due fonti di possibili errori**

- ▶ **confondere uguaglianza (`==`) con assegnamento (`=`)**
`if (i = 0)` vale *falso* qualunque sia il valore iniziale di *i*
- ▶ **costruire espressioni con operatori relazionali multipli**
`if (10 < i < 6)` vale *vero* se *i* è > 10

Selezione di un blocco

Si può condizionare l'esecuzione di un intero blocco anziché un'istruzione

```
if (espressione)  
{  
    istruzione/i  
}
```

Se l'espressione ha valore *vero*, esegue il blocco; altrimenti no

Come sempre

- ▶ ogni istruzione termina con `;`
- ▶ il blocco nel suo complesso no

La clausola else

Se vi sono due blocchi alternativi

```
if (espressione)
{
    istruzione/i
}
else
{
    istruzione/i
}
```

Se l'espressione vale *vero*, esegue il primo blocco; altrimenti il secondo

Ogni istruzione termina con *;* mentre i due blocchi no

Operatore condizionale (1)

(espressione1 ? espressione2 : espressione3)

è l'*unico operatore ternario* del C:

è costituito dai due simboli *?* e *:* che separano i tre operandi

Produce un'espressione composta, che si valuta così:

1. si valuta *espressione1* che è di tipo logico
2. se *espressione1* è vera, si valuta *espressione2*
se *espressione1* è falsa, si valuta *espressione3*
3. si assegna all'espressione composta il valore dell'espressione valutata

Operatore condizionale (2)

Per conservare il valore, si può assegnarlo a una variabile:

variabile = (espressione1 ? espressione2 : espressione3);

significa

```
if (espressione1)
{
    variabile = espressione2;
}
else
{
    variabile = espressione3;
}
```

Che cosa significano le seguenti espressioni?

- ▶ *k = (i >= 0 ? i : 0);*
- ▶ *k = (i > j ? i : j);*

Selezioni annidate

I costrutti di selezione
si possono annidare

Chiudere fra parentesi graffe
anche le istruzioni singole
e indentare ciascuna condizione
rende la struttura più chiara

```
if (espressione1)
{
    if (espressione2)
    {
        istruzioni
    }
}
else
{
    if (espressione3)
    {
        istruzioni
    }
    else
    {
        istruzioni
    }
}
```

Selezioni in cascata

Se i blocchi alternativi sono più di due, se ne isola uno alla volta

```
if (caso1)
{
    istruzioni
}
else
{
    if (caso2)
    {
        istruzioni
    }
    else
    {
        if (caso3)
        {
            istruzioni
        }
        else
        {
            istruzioni
        }
    }
}
```

Per chiarezza, in questi casi
si usa indentare diversamente

```
if (caso1)
{
    istruzioni
}
else if (caso2)
{
    istruzioni
}
else if (caso3)
{
    istruzioni
}
else
{
    istruzioni
}
```

L'else pendente

Ogni **else** si riferisce all'**if** più vicino non ancora accoppiato
se le parentesi graffe non indicano altrimenti

Questo può facilmente indurre in errore: meglio usare sempre le
parentesi

```
if (y != 0)
    if (x != 0)
        r = x / y;
else
    printf("Errore!");
```

Se y vale 0, non stampa nulla!

```
if (y != 0)
{
    if (x != 0)
        r = x / y;
}
else
    printf("Errore!");
```

Versione corretta

Selezione multipla (1)

L'istruzione `switch` consente di separare **blocchi alternativi che corrispondono ai singoli valori possibili per un'espressione**

<pre>switch (espressione) { case costante1 : istruzione/i case costante2 : istruzione/i ... default: istruzione/i }</pre>	<pre>if (espressione == costante1) { istruzione/i } else if (espressione == costante2) { istruzione/i } else if else { istruzione/i }</pre>
---	---

Il blocco `default` corrisponde ai casi non elencati esplicitamente, cioè all'ultimo `else`

Il blocco `default` può mancare (come l'ultimo `else`)

Selezione multipla (2)

L'istruzione viene eseguita così

1. si valuta l'espressione
2. si cerca la costante di valore uguale all'espressione
 - ▶ se c'è, si eseguono le istruzioni associate alla costante e quelle associate a tutte le costanti successive
 - ▶ se non c'è, si eseguono le istruzioni associate a `default`

Per eseguire le istruzioni associate a una sola costante, occorre terminarle con l'istruzione `break`;

Il costrutto `switch` viola la **programmazione strutturata**, perché in generale non ha un solo punto di ingresso e un solo punto di uscita:

- ▶ ogni case è un diverso punto di entrata (così pure il `default`)
- ▶ ogni istruzione `break` è un diverso punto di uscita

In pratica, l'uso è talmente specifico e chiaro da risultare accettabile

Selezione multipla (3)

- ▶ L'**espressione** valutata deve essere di **tipo intero o carattere**
- ▶ Le **costanti** devono essere di **tipo intero o carattere**
- ▶ Le **costanti** devono avere tutte **valori diversi**
- ▶ Le costanti possono essere esplicite o simboliche, semplici o composte
- ▶ Le istruzioni associate a ogni valore non richiedono parentesi graffe

(non formano un blocco!)