Costrutti di Programmazione Strutturata

Luca Abeni

March 20, 2014

Costrutti di Base

- Ricordi? Un algoritmo è detto strutturato se usa solo
  - Sequenze di istruzioni
  - Selezioni (se *predicato* allora fai *istruzioni* altrimenti *istruzioni*)
  - Cicli (mentre *predicato* esegui *istruzioni*)
- Un algoritmo strutturato è scomponibile in blocchi di istruzioni
  - Annidati o in sequenza

Supporto in C

- Il linguaggio C permette di evidenziare blocchi di istruzioni
  - Simboli "{" e "}"
- Inoltre, fornisce i 3 costrutti base che ci servono
  - Sequenze di istruzioni
  - Selezioni (`if (...) { ... } else { ... }`)
  - Cicli (`while (...) { ... }`)
- Fornisce anche altri costrutti per cicli e salti...
- ...Ed altri costrutti non strutturati (`goto`, ...)!!!
  - Ma ignoriamo questa cosa...

Sequenze di Istruzioni in C

- Primo esempio di blocco di istruzioni: corpo di una funzione

```
int f(int i)
{
  int res;

  ...

  return res;
}
```

- Può contenere definizioni di variabili e sequenze di istruzioni
  - Istruzioni: assegnamenti, invocazioni di funzione
  - Terminano con ";"
- Nota: in C, ";" non è usato per comporre sequenzialmente 2 istruzioni, ma per terminare un'istruzione o dichiarazione/definizione di variabile
  - Ci vuole ";" anche dopo "`return res`"

Sequenze - 2

- È possibile inserire “;” anche dove non strettamente necessario...
  - Esempio: `a = 1;;` (notare il doppio “;”)
  - Che significato ha?
- Un “;” indica sempre la fine di un’istruzione...
  - ...Se l’istruzione non c’e’, istruzione vuota!
  - Equivalente a “non fare niente”
  - Equivalente a blocco vuoto “{}”
  - Vedremo poi come questo possa essere pericoloso!
- `a = 1;;` $\Leftrightarrow$ `a = 1; {};`

Selezioni in C

- Il costrutto "se *predicato* allora fai *istruzioni* altrimenti *istruzioni*" si mappa direttamente in C

```
if (predicato) {
    ...
    /* istruzioni */
    ...
} else {
    ...
    /* istruzioni */
    ...
}
```

- In inglese, "if" significa "se" ed "else" significa altrimenti
- Notare che il costrutto crea 2 sottoblocchi di istruzioni
  - In realtà la cosa sarebbe più complessa, ma semplifichiamo...

Il Costrutto if()

- La parola chiave `if` è seguita da un predicato fra parentesi...
- ...Il predicato viene valutato e se è vero viene eseguito il bloccho di istruzioni subito seguente alla parentesi chiusa
  - In questo caso, il blocco che segue `else` non viene eseguito
- Se il predicato è falso, viene eseguito il blocco di istruzioni che segue la keyword `else`
  - In questo caso, il blocco che segue il predicato non viene eseguito

```
if (predicato) {
    ...
    /* istruzioni */
    ...
} else {  /* ramo else vuoto */
}
```

→

```
if (predicato) {
    ...
    /* istruzioni */
    ...
}
```

Esempi di Selezione in C

```c
void positivo_negativo(int n)
{
  if (n >= 0) {
    printf("Numero positivo\n");
  } else {
    printf("Numero negativo\n");
  }
```

- Funzione che stampa se il parametro ricevuto è positivo...
- Ma... Stampa anche che 0 è positivo???

```c
void positivo_negativo_nullo(int n)
{
  if (n > 0) {
    printf("Numero positivo\n");
  } else {
    if (n == 0) {
      printf("Numero nullo\n");
    } else {
      printf("Numero negativo\n");
    }
  }
```

$\longrightarrow$

```c
void positivo_negativo_nullo(int n)
{
  if (n > 0) {
    printf("Numero positivo\n");
  } else if (n == 0) {
    printf("Numero nullo\n");
  } else {
    printf("Numero negativo\n");
  }
```

- Non sempre è chiaro a quale `if()` un `else` si riferisce (quello immediatamente precedente)

Attenzione!!!

- Ricordi? È possibile aggiungere “;” inutili nel codice C...
  - `a = 1;;` $\Leftrightarrow$ `a = 1; {};`
  - Aggiunge un blocco vuoto (o istruzione “non fare niente”)
- Fino ad ora questo sembrava innocuo, ma...
  - Che effetto ha un “;” subito dopo un `if()` (prima del primo blocco di istruzioni)?
- Un “;” sbagliato causa l'esecuzione incondizionata del blocco di istruzioni!

```
if (n > 0); {
    printf("Positivo\n");
}
```
$\Leftrightarrow$
```
if (n > 0) {};  {
    printf("Positivo\n");
}
```
$\Leftrightarrow$
```
if (n > 0) {
}
printf("Positivo\n");
```

Esempio: Valore Assoluto

```
unsigned int valore_assoluto(int n)
{
  if (n >= 0) {
    return n;
  }

  return -n;
}
```

```
unsigned int valore_assoluto(int n)
{
  unsigned int risultato;

  if (n >= 0) {
    risultato = n;
  } else {
    risultato = -n;
  }

  return risultato;
}
```

- L'esempio a sinistra ha due punti di uscita...
- ...Tecnicamente, non strutturato!!!

Esempio su Selezioni Multiple

- Verifica se un carattere c è una lettera
- Dalla tabella ASCII, 0x41 - 0x5a e 0x61 - 0x7a

```
int lettera(char c)
{
  int risultato = 0;

  if (c >= 0x41) {
    if (c <= 0x5a) {
      risultato = 1;
    }
  } else {
    if (c >= 0x61) {
      if (c <= 0x7a) {
        risultato = 1;
      }
    }
  }

  return risultato;
}
```

```
int lettera(char c)
{
  int risultato = 0;

  if (c >= 'A') {
    if (c <= 'Z') {
      risultato = 1;
    }
  } else {
    if (c >= 'a') {
      if (c <= 'z') {
        risultato = 1;
      }
    }
  }

  return risultato;
}
```

```
int lettera(char c)
{
  int risultato = 0;

  if (((c >= 'A') && (c <= 'Z')) ||
      ((c >= 'a') && (c <= 'z'))) {
    risultato = 1;
  }

  return risultato;
}
```

- Predicati composti invece di `if` annidati

- Più semplice...

Cicli in C

- Il costrutto "mentre *predicato* esegui *istruzioni*" si mappa direttamente in C

```
while (predicato) {
    ...
    /* istruzioni */
    ...
}
```

- In inglese, "while" significa "mentre"
- Notare che il costrutto crea un sottoblocco di istruzioni
  - Ancora, la cosa sarebbe più complessa, ma semplifichiamo...

Il Costrutto while()

- La parola chiave `while` è seguita da un predicato fra parentesi...
  - ...Il predicato viene valuato e se è vero viene eseguito il blocco di istruzioni subito seguente alla parentesi chiusa
  - Alla fine dell'esecuzione del blocco, il predicato viene valutato ancora e se è vero il blocco viene eseguito un'altra volta...
  - ...E così via
- Il blocco di istruzioni viene eseguito fino a che il predicato non diventa falso

Esempi di Cicli

Contatore

```
i = 0;
while (i < 10) {
   printf("%d\n", i);
   i = i + 1;
}
```

Ciclo Generico

```
r = a % b;
while (r != 0) {
   a = b;
   b = r;
   r = a % b;
}
```

- Particolare "tipologia" di ciclo: contatore
  - Termina quando una variabile raggiunge un valore prestabilito
  - Variabile incrementata nel corpo del ciclo

Cicli Infiniti

- Cicli con contatore
  - Garanzia che il predicato diventa falso in un numero noto di passi
  - Il ciclo termina in un tempo finito
- Cicli generici
  - No garanzia che il predicato diventi falso prima o poi
  - Il ciclo potrebbe non terminare mai...
- Per la correttezza dell'algoritmo è importante dimostrare che il ciclo termini

Predicati in C

- I costrutti `if()` e `while()` contengono dei predicati
  - `if(`*predicato*`)`: se il predicato è vero esegui un blocco, altrimenti eseguine un altro
  - `while(`*predicato*`)`: ripeti un blocco fino a che ill predicato è vero
- Predicato: proposizione il cui valore di verità dipende dal valore di una o più variabili
  - Il tipo più naturale per un predicato sarebbe booleano (vero/falso)...
  - ...Ma il linguaggio C usa espressioni intere come predicati!
- Un valore intero è valutato come "vero" se è $\neq 0$, viene valutato come "falso" se è $= 0$

Esempi di Predicati in C

- Espressioni di questo genere sono sintatticamente corrette:
  - **if** (5) {
    - $5 \neq 0$: esegue sempre il blocco che segue (mai il blocco else)
  - **while** (1) {
    - $1 \neq 0$: ciclo infinito
  - **while** (x) {
    - ripete il ciclo fino a che il valore di x è $\neq 0$
  - **if** (a + b) {
    - equivalente a **if** (a + b != 0) {
- A volte, effetti sorprendenti

Predicati in C: Sorprese

- **if** (delta = 0) {
  - Nota: questo è un errore di battitura!!!
  - Avrebbe dovuto essere **if** (delta == 0) {!
  - Perché viene compilato senza errori???
- delta = 0: assegnamento (con "==" sarebbe stato un confronto
  - Ma gli assegnamenti in C hanno un valore di ritorno...
  - ...Uguale al valore assegnato!
- delta = 0 viene valutato come 0 = "falso"
  - Inoltre, cambia il valore di `delta`

Predicati Composti

- Predicati "semplici" possono essere combinati usando gli operatori dell'algebra booleana
  - *and*, *or* e *not* (*e*, *o* e *non*)
  - In C, `&&`, `||` e `!`
- <espressione 1> && <espressione 2> vale 0 se almeno una delle espressioni viene valuata a 0, vale 1 altrimenti
- <espressione 1> || <espressione 2> vale 0 se entrambe le espressioni vengono valuate a 0, vale 1 altrimenti
- !<espressione> vale 1 se l'espressione viene valutata a 0, vale 1 altrimenti