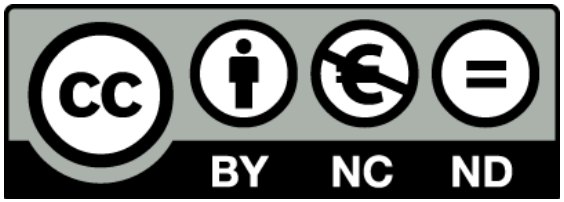
Laboratorio di Programmazione di Sistema

Controllo dell'Esecuzione in Linguaggio C

Luca Forlizzi, Ph.D.

Versione 20.1



Luca Forlizzi, 2020

© 2020 by Luca Forlizzi. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.

I Costrutti per il Controllo dell'Esecuzione

- Il linguaggio C offre i seguenti costrutti per il controllo dell'esecuzione dei programmi
  - Il *compound statement* `{ ... }`, che permette di definire una *sequenza* di istruzioni
  - Le istruzioni `if` e `switch` per effettuare la *selezione* di istruzioni
  - Le istruzioni `while`, `for` e `do-while` per effettuare l'*iterazione* di istruzioni
  - Le istruzioni `break`, `continue` e `goto` per *saltare* ad una prossima istruzione da eseguire
  - La chiamata di funzione e l'istruzione `return` per *saltare* ad un'altra funzione da eseguire
- La chiamata di funzione e l'istruzione `return` verranno esaminate approfonditamente in future lezioni
- Per i dettagli sulle rimanenti istruzioni, si rimanda ai capitoli 5 e 6 di **[Ki]**

Controllo dell'Esecuzione in Unstructured-C

- Nello stile *UC* le espressioni logiche ammesse sono solo quelle che hanno una delle forme seguenti
  - `A`, dove `A` è una variabile oppure una costante
  - `!A`, dove `A` è una variabile oppure una costante
  - `A op B`, dove `A` e `B` sono ciascuno una variabile oppure una costante e `op` è uno degli operatori relazionali binari `<`, `<=`, `>`, `>=`, `==`, `!=`, oppure uno degli operatori logici binari `&&`, `||`
- Gli operatori di concatenazione e condizionale non sono ammessi
- Sono possibili assegnamenti nella forma `V = E`, dove `E` è una espressione logica in una delle forme sopra elencate, ma non assegnamenti in cui l'operando destro è un'espressione logica non ammessa

Controllo dell'Esecuzione in Unstructured-C

- Lo stile *UC* prevede che ogni istruzione `if` abbia la forma
  `if ( A op B ) goto L`
  dove
  - `A` e `B` sono ciascuno una variabile oppure una costante
  - `op` è uno degli operatori relazionali binari <, <=, >, >=, ==, !=
  - `L` è una label
- Si osservi, quindi, che in *UC* non sono ammesse istruzioni `if`
  - che hanno la keyword `else`
  - che hanno un'espressione di controllo formata da una singola variabile o costante
  - che hanno un'espressione di controllo con operatori aritmetici
  - che hanno un'espressione di controllo con operatori logici

Controllo dell'Esecuzione in Unstructured-C

- In *UC* è ammessa l'istruzione `goto`
- In *UC* non sono ammesse le istruzioni `switch`, `while`, `for`, `do-while`, `break`, `continue`
- Si osservi, quindi, che in *UC* i soli compound statement usati sono i corpi delle funzioni
- Dunque, lo stile *UC* impone vincoli molto rigidi ai costrutti per il controllo dell'esecuzione
- Nonostante ciò, è possibile trasformare qualunque programma C in un programma in stile *UC* che, magari in modo meno elegante e conciso, esegue le stesse istruzioni

Esempi di trasformazione in Unstructured-C

- Code1_pc mostra un'istruzione `if` in una forma non ammessa in stile UC

Code1_pc

```
a = 4;
if ( x > 10 )
  a = a - b + 1;
else a += b;
c = a + 1;
```

Esempi di trasformazione in Unstructured-C

- Code1_pc mostra un'istruzione `if` in una forma non ammessa in stile UC
- Code1_uc mostra una possibile trasformazione in stile UC

Code1_pc

```
a = 4;
if ( x > 10 )
  a = a - b + 1;
else a += b;
c = a + 1;
```

Code1_uc

```
a = 4;
if ( x > 10 ) goto if_true;
  a += b;
  goto if_end;
if_true:
  int t;
  t = a - b;
  a = t + 1;
if_end:
c = a + 1;
```

Esempi di trasformazione in Unstructured-C

- Code2_pc è un altro esempio di istruzione `if` in una forma non ammessa in stile UC

Code2_pc

```
if ( x > 0 && y <= 10 ) {
  a = 10;
  b = 3;
} else a = b + 5;
c = 8;
```

Esempi di trasformazione in Unstructured-C

- Code2_pc è un altro esempio di istruzione `if` in una forma non ammessa in stile UC
- Code2_uc_1 mostra una possibile trasformazione in stile UC

Code2_pc

```
if ( x > 0 && y <= 10 ) {
  a = 10;
  b = 3;
} else a = b + 5;
c = 8;
```

Code2_uc_1

```
int t0, t1, t2;
t1 = x > 0;
t2 = y <= 10;
t0 = t1 && t2;
if ( t0 == 0 ) goto if_else;
  a = 10;
  b = 3;
  goto if_end;
if_else:
  a = b + 5;
if_end:
c = 8;
```

Esempi di trasformazione in Unstructured-C

- Ragionando con le leggi di De Morgan, è possibile trasformare Code2_pc in altro modo

Code2_pc

```
if ( x > 0 && y <= 10 ) {
  a = 10;
  b = 3;
} else a = b + 5;
c = 8;
```

Esempi di trasformazione in Unstructured-C

- Ragionando con le leggi di De Morgan, è possibile trasformare Code2_pc in altro modo
- Code2_uc_2 richiede meno istruzioni e meno variabili ausiliarie rispetto a Code2_uc_1

Code2_pc

```
if ( x > 0 && y <= 10 ) {
  a = 10;
  b = 3;
} else a = b + 5;
c = 8;
```

Code2_uc_2

```
if ( x <= 0 ) goto if_else;
if ( y > 10 ) goto if_else;
  a = 10;
  b = 3;
  goto if_end;
if_else:
  a = b + 5;
if_end:
c = 8;
```

Esempi di trasformazione in Unstructured-C

- Code3_pc contiene un'istruzione `do-while`

Code3_pc

```
do {
   a = v + 1;
   b++;
} while ( k );
c = 9;
```

Esempi di trasformazione in Unstructured-C

- Code3_pc contiene un'istruzione `do-while`
- Code3_uc mostra una possibile trasformazione in stile UC

Code3_pc

```
do {
  a = v + 1;
  b++;
} while ( k );
c = 9;
```

Code3_uc

```
do_begin:
  a = v + 1;
  b += 1;
if ( k != 0 ) goto do_begin;
c = 9;
```

Controllo_Esecuzione_C: Indicazioni per Studio ed Esercizi

- Per studiare in dettaglio gli argomenti introdotti in questa presentazione
  - Sezioni 5.1 e 5.2 di **[Ki]**
  - Capitolo 6 di **[Ki]**
- Svolgere gli esercizi delle lezioni **Selezione in C** e **Iterazione in C** su *Edu99*