Corso “Programmazione 1”
Capitolo 03: Istruzioni

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 27 settembre 2021

Terms of Use and Copyright

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

L'Istruzione Iterativa **`while`** (while-do)

- Sintassi: **`while`** `(exp) istruzione`
  - `exp` è un'espressione Booleana
  - `istruzione` può essere un'istruzione complessa
- L'esecuzione dell'istruzione **`while`** comporta
  1. il calcolo dell'espressione `exp`
  2. se `exp` è vera, l'esecuzione di `istruzione` e la ripetizione dell'esecuzione dell'istruzione **`while`**
- `istruzione` potrebbe non essere mai eseguita
- È possibile generare loop infiniti.

Nota

`exp` tipicamente contiene almeno una variabile (variabile di controllo del ciclo), che viene modificata in `istruzione` per far convergere `exp` verso uno stato in cui diventi falsa.

L'Istruzione Iterativa `while`: Esempi I

- ripetizione pedissequa di un'operazione (contatore crescente):
  { `ESEMPI_LOOP/stampaciao.cc` }
- ... (contatore decrescente):
  { `ESEMPI_LOOP/stampaciao2.cc` }
- ..., con loop infinito:
  { `ESEMPI_LOOP/stampaciao_infloop.cc` }
- somma con accumulatore:
  { `ESEMPI_LOOP/sommainteri_while.cc` }
- prodotto con accumulatore:
  { `ESEMPI_LOOP/fact_while.cc` }
- condizione di uscita diversa da conteggio:
  { `ESEMPI_LOOP/divisibile.cc` }

L'Istruzione Iterativa **`while`**: Esempi II

- ripetizione di comando a menu:
  { `ESEMPI_LOOP/conversione3_while.cc` }
- somma con accumulatore, con conteggio:
  { `ESEMPI_LOOP/serie_while.cc` }
- somma con accumulatore, con cond. uscita :
  { `ESEMPI_LOOP/serie_while1.cc` }
- uso di "cin loops":
  { `ESEMPI_LOOP/cin_loop.cc` }
- stessa cosa, ma con fail:
  { `ESEMPI_LOOP/cin_loop_equivalent.cc` }

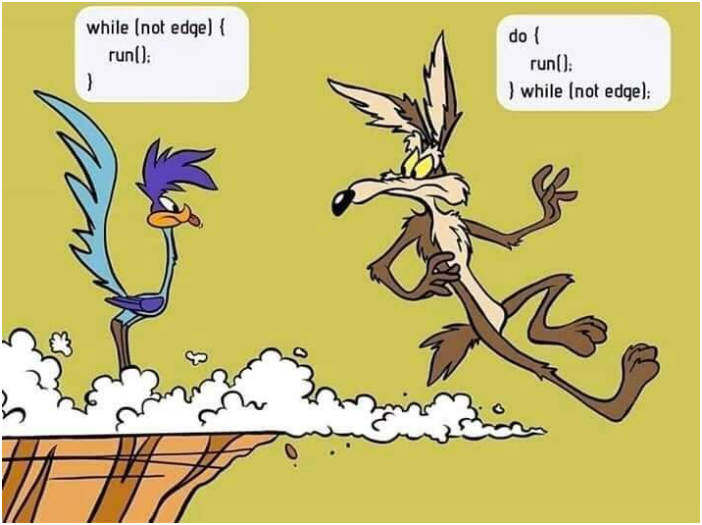
L'Istruzione Iterativa `do` (do-while)

- Sintassi: **do** `{ istruzione }` **while** `(exp);`
  - `exp` è un'espressione Booleana
  - `istruzione` può essere un'istruzione complessa
- L'esecuzione dell'istruzione **do** comporta
  1. l'esecuzione di `istruzione`
  2. il calcolo dell'espressione `exp`
  3. se `exp` è vera, la ripetizione dell'esecuzione dell'istruzione **do**
- `istruzione` viene sempre eseguita almeno una volta
- è la meno usata tra le istruzioni iterative.

L'Istruzione Iterativa `do`: Esempi

- somma con accumulatore (do):
  { `ESEMPI_LOOP/sommainteri_do.cc` }
- ripetizione di comando a menu (do):
  { `ESEMPI_LOOP/conversione3_do.cc` }
- conversione di base:
  { `ESEMPI_LOOP/base.cc` }

While-Do vs. Do-While



L'Istruzione Iterativa `for`

- Sintassi: **`for`** `( init; exp; agg) istruzione`
  - `init` è un'istruzione di inizializzazione delle variabili di controllo
  - `exp` è un'espressione Booleana
  - `istruzione` può essere un'istruzione complessa
  - `agg` è un'istruzione di aggiornamento delle variabili di controllo
- L'esecuzione dell'istruzione **`for`** comporta:
  1. l'esecuzione di `init`
  2. il calcolo dell'espressione `exp`
  3. se `exp` è vera, viene eseguita `istruzione`, poi `agg`, e si ricomincia dal passo 2.
- è la più usata tra le istruzioni iterative.
- si possono definire variabili di controllo interne al ciclo:
  `for` `(`**`int`** `i=0;i<MAXDIM;i++) {<i occorre solo qui>}`

Consente di separare le istruzioni di controllo del ciclo e concentrarle tutte in un'unica riga $\Longrightarrow$ miglior praticità e leggibilità del codice.

Cicli for e while

```
for ( init; exp; agg )
   istruzione
```

equivale a:

```
{ init;
  while ( exp ) {
    istruzione
    agg;
}; }
```

Esempio

```
for (int i=1; i<10; i++)
   x*=2;
```

$\Longleftrightarrow$

```
{ int i=1;
while (i<10) {
   x*=2;
   i++;
}; }
```

L'Istruzione Iterativa `for`: Esempi I

- prodotto con accumulatore (for):
  { `ESEMPI_LOOP/fact_for.cc` }
- somma con accumulatore (numero iterazioni) (for):
  { `ESEMPI_LOOP/serie_for.cc` }
- somma con accumulatore (cond. uscita) (for):
  { `ESEMPI_LOOP/serie_for1.cc` }
- for annidati:
  { `ESEMPI_LOOP/doublefor.cc` }

L'Istruzione Iterativa `for`: Esempi II

- condizione iniziale multipla con for:
  { `ESEMPI_LOOP/serie_for1_2init.cc` }
- cond. iniziale multipla & uscita multipla con for:
  { `ESEMPI_LOOP/serie_for1_2init2.cc` }
- incremento come input dato dall'utente:
  { `ESEMPI_LOOP/minmax.cc` }
- doppio incremento:
  { `ESEMPI_LOOP/doublecontrol.cc` }

Gli Invarianti di un Ciclo (Loop Invariant)

- Tecnica per la verifica di correttezza dei cicli (proprietà P)
- Idea: suddividere la proprietà desiderata P della correttezza del ciclo in una sequenza di affermazioni P_0, P_1, ... P_n, in modo che:
  (1) P_0 sia vera immediatamente prima che il ciclo inizi
  (dopo l'inizializzazione!)
  (2) per ogni indice di ciclo $i \in \{1, ..., n\}$:
  se P_{i-1} è vera prima dell'inizio del ciclo i-esimo (ed è verificata la condizione di permanenza del ciclo), allora P_i è vera alla fine del ciclo i-esimo
  (e quindi immediatamente prima dell'inizio del ciclo $(i+1)$-esimo)
  (3) Alla fine dell'ultimo ciclo (n-esimo), P_n (e la negazione della condizione di permanenza) implica la proprietà P
- Tipicamente (2) è il passo più critico
- P_i a volte ovvie, a volte molto complesse (or, if-then-else, ...)
  $\Longrightarrow$ problema indecidibile in generale
- Talvolta necessarie variabili ausiliarie addizionali
- Talvolta si adottano convenzioni per gestire il caso $i = 0$:
  (la somma di 0 elementi è 0, il prodotto di 0 elementi è 1, ...)

Esempio: fattoriale

```
i = 1;
fact = 1;
while (i<=n) {
  fact *= i;
  i++;
}
```

- Proprietà P: dopo il ciclo, `fact` vale il prodotto dei primi `n` numeri
- Invariante P_i: `fact` vale il prodotto dei primi `i` numeri
  - ✓(1) prima del ciclo, `fact` vale il prodotto dei primi `0` numeri (cioè 1)
  - ✓(2) prima dell'i-esimo ciclo `fact` vale il prodotto dei primi `i-1` numeri
    $\Longrightarrow$ dopo l'i-esimo ciclo `fact` vale il prodotto dei primi `i` numeri
  - ✓(3) Alla fine dell'ultimo ciclo (n-esimo), P_n (più la negazione della condizione di permanenza del ciclo) implica la proprietà P

Nota

"dopo l'i-esimo ciclo" i è incrementato di 1. (Ex: dopo il 3^o ciclo, $i = 4$).

Esempio: divisibilità per 2

```
ndiv2=0; tmp=num;  // "tmp" ausiliaria
while ( tmp%2 == 0 ) {
  ndiv2++;
  tmp/=2;
}
```

- Proprietà P: dopo il ciclo, `tmp%2!=0` e `tmp*(2^ndiv2)==num`
- Invariante P_i: `tmp*(2^ndiv2)==num`
  - ✓(1) prima del ciclo, `tmp*(2^0)==num`
  - ✓(2) prima dell'i-esimo ciclo `tmp` è divisibile per due e `tmp*(2^ndiv2)==num` $\Longrightarrow$ dopo l'i-esimo ciclo `tmp*(2^ndiv2)==num` (infatti `tmp/2*(2^(ndiv2+1))==num`)
  - ✓(3) Alla fine dell'ultimo ciclo (n-esimo), P_n (più la negazione della condizione del ciclo) implica la proprietà P : `tmp*(2^ndiv2)==num` e `tmp%2!=0`

Esercizi Proposti

Esercizi sui cicli:
{ `ESEMPI_LOOP/ESERCIZI_PROPOSTI.txt` }

Istruzione di Salto

Istruzioni di salto (**`break`**, **`continue`**, **`goto`**):
come **non** si deve programmare in C/C++ !!!

L'Istruzione di Salto **break**

L'istruzione **break** termina direttamente tutto il ciclo

```
while (...) {
   ...
   break;        // --+
   ...           //   |
}                //   |
       // <---------+
```

- Da evitare! $\Longrightarrow$ si può sempre fare modificando la condizione
- semplice break (while):
  { ESEMPI_LOOP/break_while.cc }
- come evitare un break (while):
  { ESEMPI_LOOP/nobreak_while.cc }

L'Istruzione `return` in un loop (salto implicito)

L'istruzione **`return`** termina direttamente il ciclo (e l'intera funzione)

```
int main () {
...
  while (...) {
    ...
    return 0;    // --+
    ...          //   |
  }              //   |
}        // <---------+
```

- Da evitare! $\Longrightarrow$ si può sempre fare modificando la condizione
- semplice return (while):
  { ESEMPI_LOOP/return_while.cc }
- come evitare un return (while):
  { ESEMPI_LOOP/noreturn_while.cc }

L'Istruzione di Salto **`continue`**

L'istruzione **`continue`** termina il ciclo attualmente in esecuzione e passa al successivo

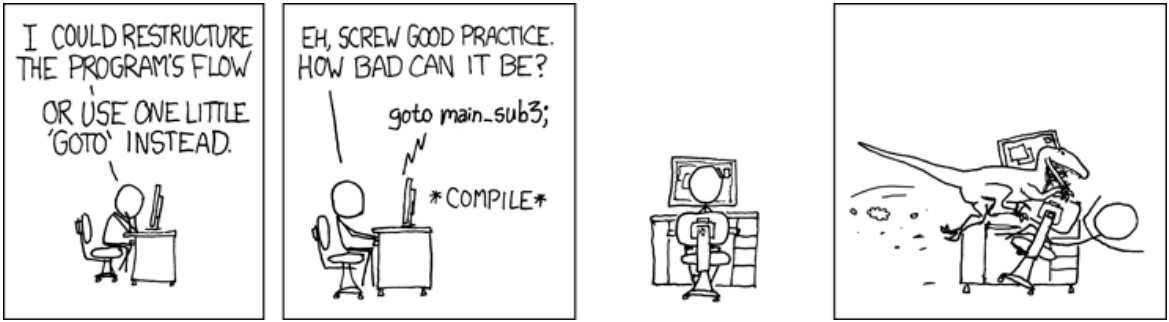
```
while (...) {
   ...
   continue;    // --+
   ...          //   |
       // <---------+
}
```

- nel caso di ciclo **`for`** viene saltata l'istruzione di aggiornamento
- Da evitare! $\Longrightarrow$ si può sempre fare lo stesso con un "if"
- semplice continue (while):
  { `ESEMPI_LOOP/continue.cc` }
- come evitare continue (while):
  { `ESEMPI_LOOP/nocontinue.cc` }

Esempio differenze istruzioni di Salto

- Semplice programmino per differenze **`break`**, **`continue`** e **`return`** :
  { `ESEMPI_LOOP/bcr.cc` }

L'Istruzione di Salto `goto` II



L'Istruzione di Salto **goto** II

- Esempio di **goto**:
  { ESEMPI_LOOP/goto.cc }

Nota di servizio:

Nella soluzione di un testo di esame, NON è ammesso l'uso di **break**, **continue**, o **goto** (con l'importante eccezione dell'uso di **break** all'interno del costrutto **switch**), o di **return** all'interno di loop, pena l'annullamento dell'esercizio stesso.