7. Programmazione strutturata Istruzioni cicliche

Andrea Marongiu
(andrea.marongiu@unimore.it)
Paolo Valente

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Semplice programma 1/2

- Scrivere un programma che, dato un numero naturale *N*, letto a tempo di esecuzione del programma stesso, stampi i primi *N* numeri naturali

Semplice programma 2/2

- Semplicissimo, **ma al momento non sappiamo scriverlo!**

Analisi del problema

- Analizziamo il problema
  - non sappiamo a priori il valore di *N*, per cui non possiamo semplicemente scrivere un programma con *N* istruzioni di stampa!

Idee

- Siccome non sappiamo a priori di quante istruzioni di stampa abbiamo bisogno,
  - un'idea sarebbe quella di far ripetere più volte la stessa istruzione di stampa
  - ma **ogni volta** l'istruzione deve stampare un valore **diverso**!
    - potremmo allora far stampare a tale istruzione il valore di una variabile
    - l'importante è che **dopo ogni stampa** tale valore venga **incrementato**!

Verso un algoritmo

- Ci serve una variabile inizializzata al valore 1 (o, se preferite, 0)
- Quindi il valore di tale variabile deve essere stampato e subito dopo **incrementato di 1**,quindi di nuovo stampato ...
- Come facciamo a sapere quando dobbiamo fermarci?
  - Ad ogni 'giro' dovremmo confrontare il valore corrente della variabile con *N* per capire se siamo o meno andati oltre
- In definitiva, per poter completare la definizione dell'algoritmo, i dati che ci occorrono sono:
  - Una variabile **N** che rappresenti il numero naturale *N* dato
  - Una variabile **i** che rappresenti un ausilio per “scorrere” tutti i valori naturali da 1 fino a *N*

Algoritmo e programma

- Inizialmente, **i** vale 1
- Finché **i<=N**, ripetere:
  - stampare il valore corrente di **i**
  - incrementare di 1 il valore corrente di **i**
- Proviamo a scrivere *parzialmente* il programma:

```
main()
{
    int i = 1, N;
    cin>>N;
   finché resta vero che (i<=N),
        ripetere il blocco { cout<<i<<endl; i++; }
}
```

Istruzioni iterative

- Come scriviamo in C/C++ la parte di programma mancante?
- Abbiamo bisogno dell'ultimo costrutto fondamentale della programmazione strutturata: le **istruzione iterative**

Istruzioni iterative

Istruzioni iterative

- Le istruzioni iterative (o **di iterazione**, o **cicliche**) forniscono costrutti di controllo che permettono di **ripetere una certa istruzione fintanto che una certa condizione è vera**
- Per il Teorema di Jacopini-Böhm, una struttura di controllo iterativa è sufficiente (insieme all'istruzione composta e di scelta) per implementare qualsiasi algoritmo
- Tuttavia, per migliorare l'espressività del linguaggio, il C/C++ fornisce vari tipi di istruzioni iterative (cicliche):
  - **`while ( ... )`**
  - **`do ... while ( ... )`**
  - **`for ( ... ; ... ; ...)`**

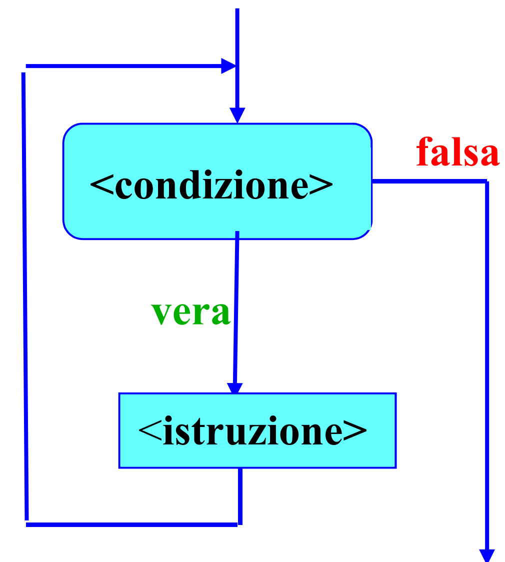
Corpo del ciclo ed iterazioni

- L'istruzione da ripetere fintanto che la condizione rimane vera viene tipicamente chiamata **corpo del ciclo**
  - A seconda dell'istruzione iterativa usata, si parla di corpo del **`while`**, del **`do … while`** o del **`for`**
- Ogni ripetizione dell'esecuzione del corpo del ciclo viene tipicamente chiamata **iterazione** (del ciclo)
- Incominciamo dall'istruzione iterativa **`while`**

Istruzione iterativa `while`

Istruzione iterativa `while`

`while` (<*condizione*>) <*istruzione*>



- <*istruzione*> costituisce il corpo del
- ciclo (**`while`**) e viene ripetuta per tutto il tempo in cui <*condizione*> rimane vera
- Se <*condizione*> è già inizialmente falsa, il ciclo non viene eseguito neppure una volta
- In generale, non è noto a priori quante volte <*istruzione*> verrà eseguita

Osservazione

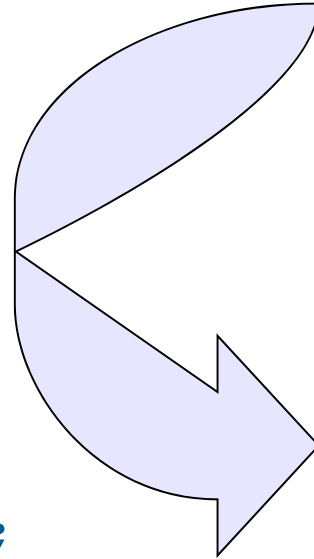
- Direttamente o indirettamente, *<istruzione>* deve modificare prima o poi la condizione, altrimenti si ha un **ciclo infinito**
- Per questo motivo, molto spesso *<istruzione>* è una istruzione composta, che contiene, tra le varie istruzioni, anche un'istruzione di modifica di qualcuna delle variabili che compaiono nella condizione
- Ci sono poi altri modi per uscire da un ciclo altrimenti infinito, che vedremo nelle prossime slide

Esercizio

- Completare il programma utilizzando l'istruzione **`while`**

Completamento programma

```
main()
{
    int i = 1, N;
    cin>>N;
```
finché resta vero che (i<=N),
ripetere il blocco `{ cout<<i<<endl; i++; }`
```
}
```



```
main()
{
    int i = 1, N;
    cin>>N;
    while (i<=N){
        cout<<i<<endl;
        i++;
    }
}
```

Modifica della variabile **i**, e quindi della condizione di ripetizione

Ciclo infinito 1/2

- Leggere la definizione nella prossima slide

Ciclo infinito 2/2

- Leggere la definizione nella slide precedente

Ciclo infinito

- Eseguire le istruzioni riportate nelle precedenti due slide porta ad un *ciclo infinito*
  - Sequenza di istruzioni ripetuta indefinitamente
- Cosa deve accadere affinché il corpo di un ciclo **`while`** sia ripetuto indefinitamente?

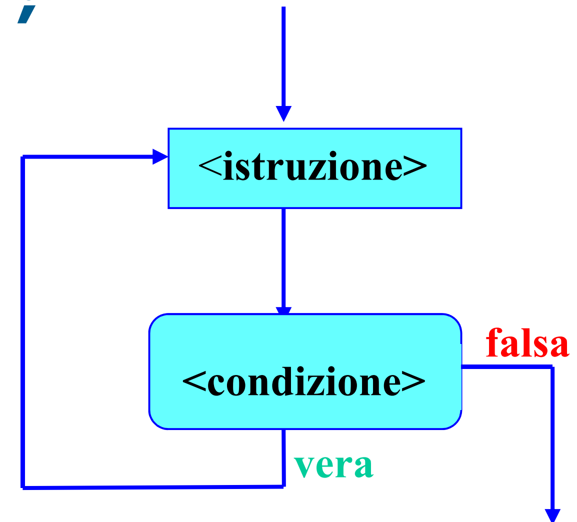
Condizione sempre vera

- Come si è visto, è necessario che la condizione sia sempre vera
- Come vedremo si può interrompere un ciclo infinito anche inserendo nel corpo del ciclo una istruzione speciale di uscita dal ciclo stesso

Istruzione iterativa
`do … while`

Istruzione `do ... while`

`do` *<istruzione>* **`while`** `(`*<condizione>*`) ;`



- È una “variazione sul tema” dell’istruzione **`while`**
- A differenza dell'istruzione **`while`**, la condizione è controllata **dopo** aver eseguito *<istruzione>*
- Quindi il (corpo del) ciclo viene sempre **eseguito almeno una volta**

Osservazioni

- Non dimenticate il `;` dopo il `while (...)`
- Analogamente al `while`, per evitare il ciclo infinito, <*istruzione*> **deve modificare prima o poi la condizione**
- Si noti che, come nel caso del `while`, si esce dal ciclo quando la condizione è falsa
- Non è adatta a quei casi in cui il ciclo può non dover essere mai eseguito
- È adatta a quei casi in cui, per valutare condizione, è necessario aver già eseguito <*istruzione*>Esempio tipico: **Controllo valori di input**

Controllo valori in input

Esempio 1: **n** deve essere positivo per andare avanti

```
do
     cin>>n;
while (n<=0);
```

Esempio 2: **n** deve essere compreso fra 3 e 15 (inclusi)

```
do
     cin>>n;
while ((n<3) || (n>15));
```

Esempio 3: **n** deve essere negativo o compreso fra 3 e 15

```
do
     cin>>n;
while ((n>=0) && ((n<3) || (n>15)));
```

Istruzione iterativa `for`

Visibilità della condizione

- Se dimenticassimo di inserire la condizione in un ciclo **`while`** o **`do … while`**, il programma non si compilerebbe affatto
- Invece, se la condizione è presente, siamo portati spontaneamente a leggerla prima di leggere il corpo del ciclo
- Quindi, nel caso in cui la condizione contenga errori, la probabilità che ce ne accorgiamo è molto alta

Domanda

- La correttezza della condizione di un ciclo è una condizione sufficiente ad assicurare che siano eseguite **tutte e sole** le iterazioni che devono effettivamente essere eseguite in accordo all'algoritmo da implementare?

Altre condizioni

- No
- Oltre alla correttezza della condizione del ciclo, sono fondamentali anche la correttezza
  - del valore iniziale e
  - delle istruzioni di modifica
- delle variabili che determinano la condizione del ciclo

Problema 1/2

- In merito possiamo evidenziare che, mentre la condizione del ciclo è esplicitata nelle intestazioni delle istruzioni `while` e `do … while`, mancano
  - sia un **punto esplicito in cui inizializzare le variabili**
  - che un **punto esplicito in cui inserire l'istruzione di modifica** della condizione del ciclo
- La mancanza dei precedenti punti espliciti fa sì che ogni programmatore inserisca le corrispondenti operazioni di inizializzazione e modifica dove meglio crede

Problema 2/2

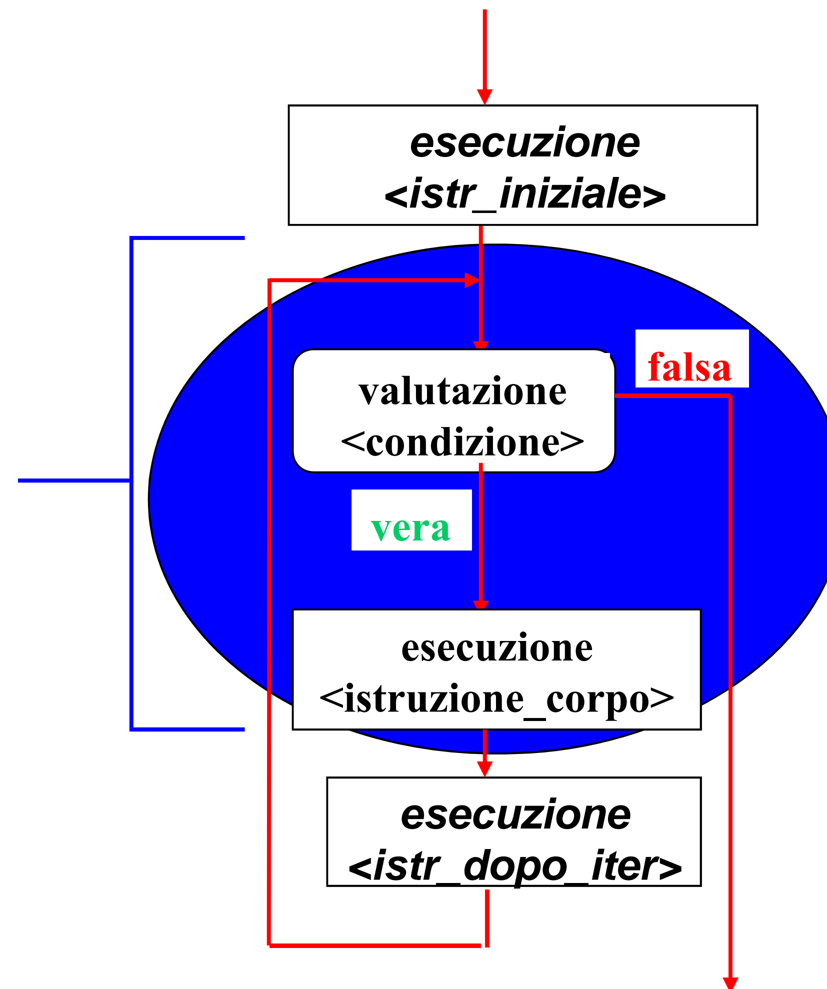
- Questo aumenta la difficoltà e la fatica di controllare la presenza/correttezza di tali operazioni
  - Quindi anche la probabilità di commettere errori
- Se invece prevedessimo dei punti espliciti in cui tali operazioni possano essere inserite, tali operazioni o la loro assenza salterebbero subito agli occhi (così come accadrebbe per la condizione del ciclo)

Istruzione iterativa `for`

- L'istruzione **`for`** è proprio una estensione dell'istruzione **`while`** in cui sono previsti, oltre ad un punto in cui inserire la condizione del ciclo, anche
  - un punto in cui inserire l'istruzione da eseguire **subito prima** della prima iterazione
  - un punto in cui inserire l'istruzione da eseguire **subito dopo** ciascuna iterazione

Sintassi e semantica

```
for (<istr_iniziale>;<condizione>;<istr_dopo_iter>)
    <istruzione_corpo>
```



Intestazione `for`

- Definiamo intestazione di un'istruzione (o ciclo) **`for`** la parte

  `for (`<*istr_iniziale*>**`;`**<*condizione*>**`;`**<*istr_dopo_iter*>**`)`**

  nella precedente definizione della sintassi dell'istruzione **`for`**

Soluzione problemi **`while`** 1/2

- Le istruzioni <*istr_iniziale*> ed <*istr_dopo_iter*> nell'intestazione del ciclo sono tipicamente utilizzate come punti espliciti per
  - inizializzare i valori delle variabili e per
  - modificare le variabili che determinano la condizione del ciclo
- Si risolvono così i problemi del **`while`** e del
- **`do … while`** precedentemente descritti

Soluzione problemi `while` 2/2

- Possiamo schematizzare la cosa nel modo seguente:

<uso-istruzione-for-per-esplicitare-inizializz_modifica> ::=

for (*<istr_inizializzazione>*;*<condizione>*;*<istr_modifica>*)
    <istruzione_corpo>

- Vediamone un esempio illuminante in linguaggio C …
  - Come vedremo in dettaglio in seguito, in linguaggio C non esiste l'oggetto **cout** e si può stampare una stringa su *stdout* con la funzione
  - **printf("Stringa da stampare") ;**
  - equivalente a:
  - **cout<<"Stringa da stampare" ;**

Esempio pratico in C

```c
#include <stdio.h>
int main(void)
{
  int count;

  for(count=1;count<=500;count++)
     printf("I will not Throw paper airplanes in class.");

  return 0;
}
```

Problema iniziale

- Risolviamo il nostro semplice problema iniziale utilizzando l'istruzione **`for`** al posto del **`while`**
- La traccia era: scrivere un programma che, dato un numero naturale *N*, letto a tempo di esecuzione del programma stesso, stampi i primi *N* numeri naturali
- Modifichiamo **opportunamente** e completiamo il programma parziale, che era:

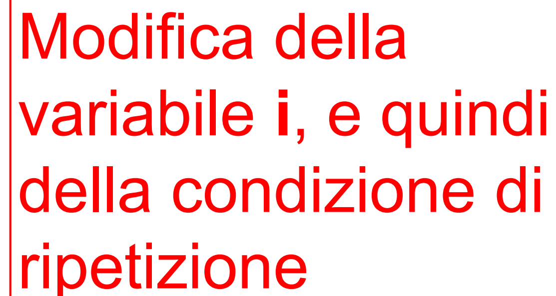
```
main()
{
    int i = 1, N;
    cin>>N;
   finché resta vero che (i<=N),
        ripetere il blocco { cout<<i<<endl; i++; }
}
```

Soluzione con `for`

```
main()
{
    int i, N;

    cin>>N;

    for(i = 1 ; i <= N ; i++)
      cout<<i<<endl;
}
```



Ulteriore vantaggio del C++

- Come abbiamo visto, col linguaggio C++ si possono inserire istruzioni qualsiasi, incluso le definizioni, in ogni punto del **main**
- In particolare si può inserire anche una **definizione come istruzione iniziale nell'intestazione** dell'istruzione **`for`**
  - In questo caso la variabile così definita si può utilizzare solo nell'intestazione e nel corpo del ciclo **`for`**
- Modifichiamo il programma precedente per sfruttare questa caratteristica

Esempio

```
main()
{
    int N;

    cin>>N;

    for(int i = 1 ; i <= N ; i++)
      cout<<i<<endl;
}
```

Commento

- Questa forma di definizione all'interno del ciclo da diversi vantaggi in termini di **leggibilità** e **riduzione del rischio di errori**
  - Tutte le **operazioni più importanti** relative alle variabili di controllo del ciclo (definizione, inizializzazione, controllo della condizione, modifica delle variabili) sono **raggruppate nell'intestazione** del ciclo
  - Variabili che devono essere utilizzate solo nel ciclo possono essere definite in maniera tale da *vivere* solo per la durata del ciclo, impedendo così di commettere l'errore di utilizzarle inavvertitamente quando non dovrebbero più essere utilizzate

Istruzioni multiple

- Si possono inizializzare più variabili nella istruzione iniziale dell'intestazione del **`for`**
- Allo stesso modo si possono effettuare più operazioni nell'istruzione da eseguire subito dopo la fine di ciascuna iterazione
- Basta utilizzare l'operatore virgola

Operatore virgola 1/2

- Date le generiche espressioni <*espr1*>, <*espr2*>, ..., <*esprN*> le si può concatenare mediante l'operatore virgola per ottenere la seguente espressione composta:
- <*espr1*>`,` <*espr2*>`, ...,` <*esprN*>

in cui

  - le espressioni <*espr1*>, <*espr2*>, ..., <*esprN*> saranno **valutate l'una dopo l'altra**
  - il valore dell'espressione composta sarà uguale a quello dell'ultima espressione valutata

Operatore virgola 2/2

Esempi:

```
int i, j ;
for(i = 1, j = 3 ; i < 5 ; i++, j--)
      ... ;
```

Definizione multipla

- Si possono inoltre definire ed inizializzare più variabili nella istruzione iniziale dell'intestazione del **`for`**
- Devono essere tutte dello stesso tipo

`for(`<tipo_variabili> *<nome_variabile1>* [**`=`** *<valore1>*] **`,`**

<nome_variabile2> [**`=`** *<valore2>*] **`, … ;`**

<condizione> **`;`** *<istruzione1>*, *<istruzione2>*, **`…)`**

- Esempio:
-
- **`for(int i = 1, j = 0 ; i <= N && j <= M; i++, j++)`**