Strutture di controllo iterative

Andrea Marin

Università Ca' Foscari Venezia
Laurea in Informatica
Corso di Programmazione part-time

a.a. 2011/2012

Introduzione

Problema

Scrivere un programma che acquisisca da standard input un intero dopo l'altro fino a quando non ne trova uno pari (che sarà l'ultimo acquisito).

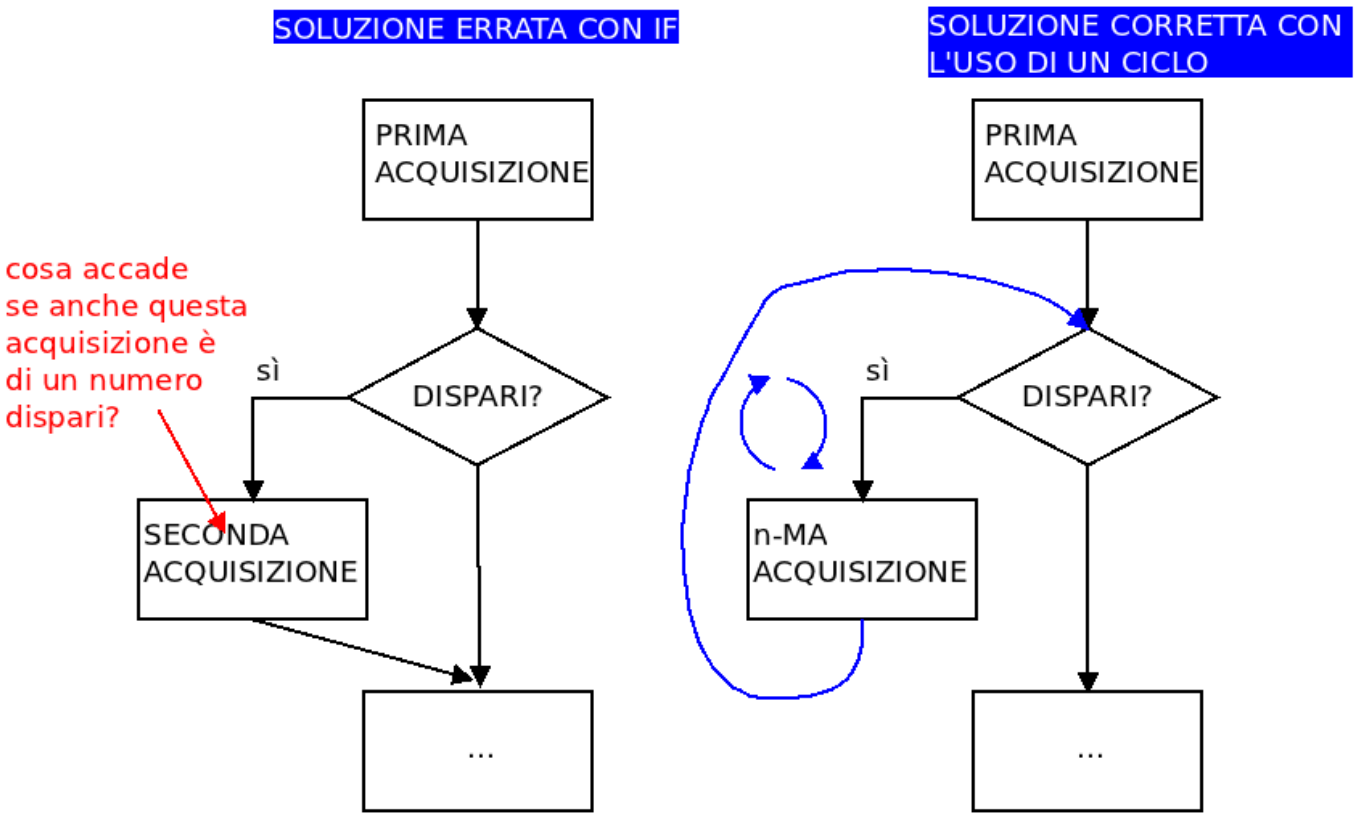
- Ricordiamo che l'istruzione `scanf` serve ad acquisire un dato da standard input
- Quante volte va eseguita l'istruzione `scanf`?
  - Non siamo in grado di deciderlo perchè lo scopriremo solo conoscendo la sequenza in input!

Una soluzione errata: l'uso del condizionale

```
#include <stdio.h>

int a;

int main() {
    scanf(''%d'', &a);
    if (a%2 != 0) {
        scanf(''%d'', &a);
    }
    ...
    return 0;
}
```

- Perchè non funziona?

Perchè serve un ciclo?

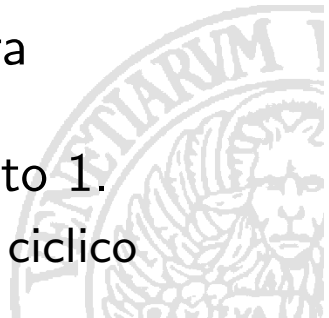


Introduzione al ciclo `while`

- Il ciclo while ha la seguente struttura sintattica:

```
while (exp)
    blocco;
```

- dove:
  - `exp`: è un'espressione di tipo `int` interpretata come boolean
  - `blocco`: è il blocco di istruzioni da eseguire (racchiuso tra graffe nel caso sia composto da più di un'istruzione)
- il ciclo `while` fa quanto segue:
  1. valuta l'espressione `exp`: se codifica il valore `true` allora esegue `blocco` altrimenti lo salta
  2. dopo aver eseguito le istruzioni di `blocco` ritora al punto 1.
     - Da questo ritorno al punto 1. si ha il comportamento ciclico



Soluzione (non migliore) dell'esempio

```
#include <stdio.h>

int a;

int main(){
    scanf(''%d'', &a);
    while (a%2 != 0) {
        scanf(''%d'', &a);
    }
    ...
    return 0;
}
```

- Siamo certi che il ciclo termini?
- Come possiamo essere certi che al termine del ciclo (nelle istruzioni ...) la variabile intera a abbia un valore pari?
- In pratica, come facciamo ad essere certi che il ciclo faccia quello che vogliamo?

Analisi dei cicli (motivazioni)

- Saper analizzare un ciclo -cioè essere certi che faccia quello che ci aspettiamo- è fondamentale
- Spesso nella programmazione i cicli rappresentano un ostacolo molto faticoso da superare
- Se non si comprendono a fondo si rischia di incorrere in molti errori

Introduciamo un esempio...

Problema

Sia data una cesta con palline bianche e nere (con almeno una pallina).

- Se nella cesta c'è più di una pallina, allora pescane due altrimenti termina
- Se le palline hanno lo stesso colore vanno buttate via e nella cesta si inserisce una pallina nera
- Se le palline hanno colore diverso, si butta la nera e si reinserisce la bianca

Vogliamo provare che il ciclo termina e alla fine del ciclo l'ultima pallina è:

- Bianca se il numero inizale di palline bianche è dispari
- Nera altrimenti

Nella prossima slide...

- `b, n`: input, numero di palline bianche e nere inizialmente nell'urna
- `p1, p2`: palline estratte, possono essere bianche o nere
- `bianca(p1)`: restituisce la codifica di `true` se `p1` è bianca
- `nera(p1)`: restituisce la codifica di `true` se `p1` è nera

Pseudo-codice C

```
int b, n; /*numero di palline bianche e nere*/
int p1, p2; /*palline pescate*/
int main() {
   scanf(''%d'', &b); scanf(''%d, &n);
   while (b+n > 1) {

      /* pesca le palline p1, p2 */
      ...

      if (bianca(p1) && bianca(p2)) {
         b = b - 2;
         n = n + 1;
      }
      else {
         if (nera(p1) && nera(p2)) {
            n = n - 1;
         }
         else { /*qual e' la condizione?? */
            n = n - 1;
         }
      }
   }
   if (b == 1) {
       printf(''E rimasta una pallina bianca'');
   } else {
       printf(''E rimasta una pallina nera'');
   }
}
```

Cosa sappiamo e cosa vogliamo ottenere dal ciclo

- Cosa sappiamo?
  - Inizialmente nella cesta c'è almeno una pallina, quindi $B \geq 0$, $N \geq 0$, $B + N \geq 1$, dove B ed n sono il numero iniziale di palline bianche e nere
  - La variabile `b` ha valore B e la variabile `n` ha valore N
- Cosa dovrebbe succedere alla fine del ciclo?
  - C'è una sola pallina nell'urna: `(b+n == 1)`
  - La pallina rimanente è bianca se il numero di palline bianche iniziali B è dispari
    - `(b == B%2)`
- Il ciclo fa proprio quanto richiesto?

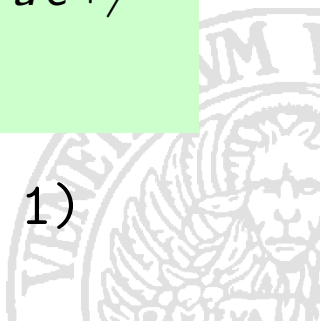
Un piccolo passo in avanti

- Possiamo osservare che se si arriva ad eseguire l'istruzione immediatamente dopo un ciclo, in quel punto possiamo assumere che la condizione del ciclo sia falsa (altrimenti si starebbe ancora ciclando)

Analisi dei cicli

```
while (exp)
    blocco1;
/*prima di eseguire istr1 so che (!exp) \'e true*/
istr1;
```

- Nell'esempio sappiamo che se il ciclo termina `(b+n <= 1)`



Terminazione del ciclo

- Una tecnica per capire se un ciclo termina certamente è quella di definire un'espressione con dominio nei numeri naturali e verificare che ad ogni iterazione diminuisce
  - variante del ciclo
- Nell'esempio possiamo osservare che ad ogni pescaggio si elimina comunque una pallina dall'urna
- Pertanto prima o poi si arriverà ad avere una sola pallina nell'urna e il ciclo termina
- Nel caso della ripetizione della lettura di un numero da standard input fino a quando non si legge un pari, quale può essere il variante?

Invariante del cliclo

- Viene chiamata invariante del ciclo una relazione tra gli elementi di un processo iterativo che vale durante tutta l'esecuzione del processo
- L'invariante è usato per dimostrare cosa fa il processo
- L'invariante non è unico e va scelto in funzione di ciò che si vuole dimostrare

Invarianti nell'esempio

- I1: `1 <= b + n, b>=0, n>=0` (c'è sempre almeno una pallina nell'urna)
- I2: `b%2 == B%2`
  - Il numero delle palline bianche rimane sempre pari o sempre dispari (infatti nelle operazioni o tolgo due bianche o ne lascio inalterato il numero)
- Alla fine del ciclo cosa sappiamo?
  - Gli invarianti sono veri ma la condizione del ciclo è falsa:
    1. `b + n >= 1, b >=0, n>=0` (I1)
    2. `b%2 == B%2` (I2)
    3. `b + n <= 1` (negazione cond. ciclo)
  - Da cui otteniamo:
    - `b == 1` e `n == 0` se `B%2 == 1`, oppure
    - `b == 0` e `n == 1`, altrimenti

Alcune considerazioni

- Per scrivere un ciclo che termina bisogna essere certi che prima o poi l'espressione in testa al ciclo diventi falsa
- Quindi se nell'espressione compaiono degli identificatori, almeno uno di questi sarà corrispondente ad una variabile che viene alterata nel ciclo
- Esempio: quando termina questo ciclo?

```
int a;
scanf(''%d'', &a);
while (a != 0) {
    a = a-2;
}
...
```

Alcuni esercizi

Usare il codice pseudo-C visto in precedenza per scrivere dei cicli corrispondenti alle seguenti situazioni (si consideri sempre l'urna inizialmente con B palline bianche e N palline nere. Determina se il ciclo termina, e il colore dell'ultima pallina

1. Se si pescano due palline dello stesso colore allora si buttano e se ne inserisce una nera; Se sono di colore diverso si butta solo la bianche e si tiene la nera
2. Se si pescano due palline dello stesso colore allora una si butta e l'altra si reinserisce; se sono di colore diverso, si butta la bianca e si reintroduce la nera

Algoritmo di Euclide per il calcolo del MCD

- Calcola il Massimo Comun Divisore (MCD) tra due numeri naturali
- Siano `a` e `b` i due numeri naturali in ingresso
  1. Se `a == b` allora il `MCD = a`
  2. Se `a < b` allora `b = b - a`
  3. altrimenti `a = a - b` (condizione per questo caso?)
  4. Torna al punto 1.
- Il ciclo termina sempre sotto le condizioni $a \geq 0$ e $b \geq 0$?
- Siamo certi che al termine del ciclo la variabile `MCD` contiene davvero il risultato corretto?

Alcune prove

Usa l'algoritmo di Euclide per calcolare il MCD tra

- 20 e 6
- 27 e 6
- 0 e 7
- 62 e 84

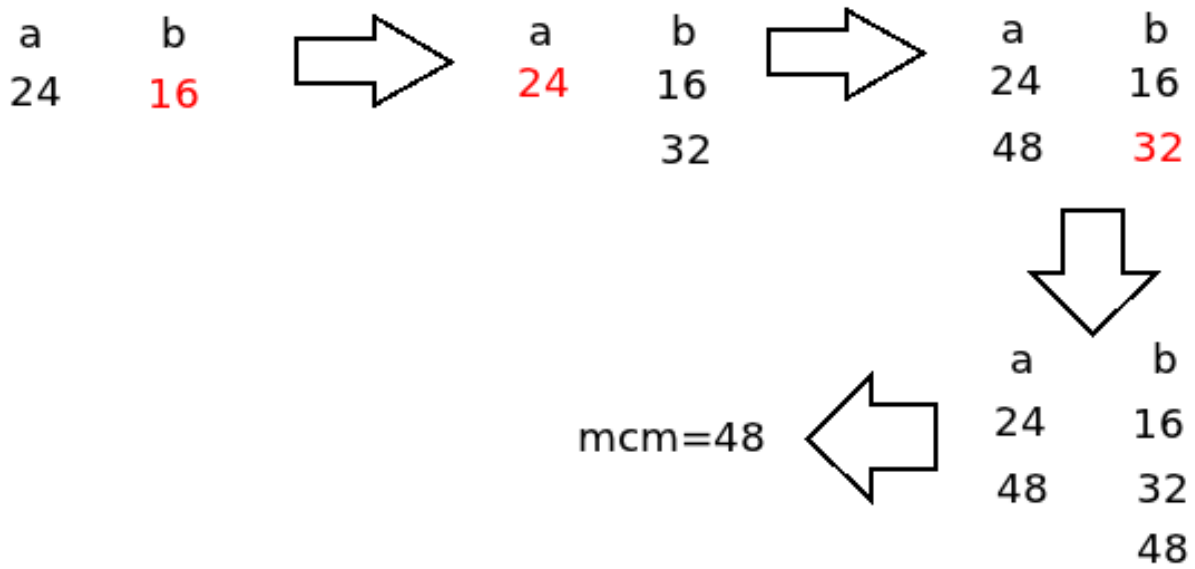
```
#include <stdio.h>

int a, b, mcd;

int main() {
  scanf(''%d'', &a);
  scanf(''%d'', &b);
  if ((a > 0)&&(b > 0)) {
     while (a != b) {
        if (a > b) {
          a = a - b;
        }
        else {
          b = b - a;
        }
     }
     printf(''MCD=%d'', a);
   }
   else {
      printf(''Inserire due numeri strettamente positivi'');
   }
   return 0;
}
```

Terminazione e correttezza

- Per convincersi della terminazione del ciclo basta osservare (provare?) che l'espressione a+b decresce sempre ma ha come minimo il valore 2 $\Rightarrow$ variante del ciclo
- Per convincersi della correttezza del ciclo osserviamo che il massimo comun divisore tra a e b è sempre uguale a quello tra i valori iniziali acquisiti $\Rightarrow$ Invariante del ciclo

Minimo comune multiplo

Algoritmo per il calcolo del minimo comune multiplo tra a e b: sommare al più piccolo dei due il suo valore iniziale fino a quando i due numeri diventano uguali



Esercizio

- Realizzare un programma C che calcoli il minimo comune multiplo tra due numeri naturali acquisiti da standard input
- Argomentare la terminazione del ciclo
- Argomentare la correttezza del ciclo