Informatica - Mod. Programmazione
Lezione 04

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Istruzioni di Controllo

- **Istruzioni Condizionali**
- **Cicli**

Cicli

Che cosa è un ciclo?

- Un **ciclo** è un blocco di istruzioni che deve essere ripetuto più volte
- per non scrivere più volte lo stesso codice
- perché non è il caso di scrivere tante volte lo stesso codice
- o perché il numero di volte da ripetere l'azione è grande
- Tutti i linguaggi di programmazione procedurale hanno almeno un'istruzione per i cicli

Cicli

In C e C++ esistono tre istruzioni:

- `while`
  L'istruzione fondamentale
- `for`
  La più usata
- `do ... while`
  La meno usata, ma comunque utile

while

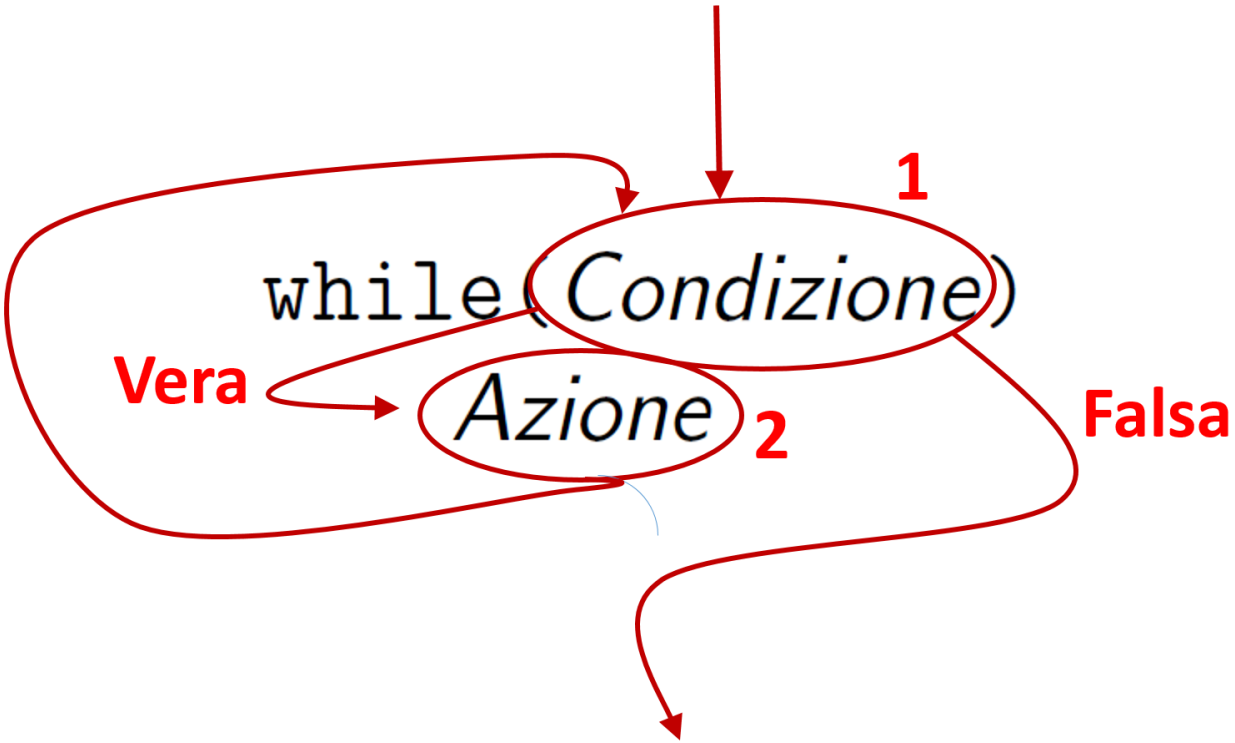
Sintassi:

`while(`*Condizione*`)`

    Azione

- Per prima cosa, si valuta la *Condizione*
  Se è **FALSA**, si passa oltre
  Se è **VERA**, si esegue l'*Azione*
- Terminata l'esecuzione dell'*Azione*,
  si valuta di nuovo la *Condizione*

while



Esempio

- Vogliamo scrivere un programma che
- legge da tastiera un numero intero
- se questo numero è divisibile per tre, scrive `"OK"` e si ferma
- altrimenti chiede un altro numero, e continua fino a che non trova un numero divisibile per 3

Programma: Divisione_per_3.cpp

```
int main()
{
    int procedi=1;
    int v;

    while( procedi )
    {
        cout << "inserisci un valore" << endl;
        cin >> v;
        if( v % 3 == 0 )
        {
            cout << "OK" << endl;
            procedi = 0;
        }
    }

    return 0;
}
```

while

- La variabile `procedi` si comporta da **FLAG** (bandiera):
  se vale 1, indica che si deve procedere
  se vale 0, indica che non si deve procedere
- All'inizio, nel ciclo si entra sicuramente,
  se l'utente inserisce un numero divisibile per 3, il valore di `procedi` diventa 0 e si termina

Esempio

- Vogliamo scrivere un programma
- che legge da tastiera un numero intero positivo `n`
- e calcola la somma dei primi `n` numeri positivi

Programma: Somma_Primi_Numeri.cpp

```
int main()
{
    int n;
    int somma = 0;
    int i;

    cout << "inserisci un valore" << endl;
    cin >> n;

    i=1;
    while( i <= n )
    {
        somma += i;
        i++;
    }

    cout << "Somma primi " << n << " numeri=" << somma;

    return 0;
}
```

while

- La variabile `i` si comporta da **contatore**:
  serve per contare quante volte l'azione viene eseguita
- La variabile `somma` si comporta da **sommatore**: serve per calcolare la somma e DEVE essere inizializzata a 0

Cicli a Contatore

- I cicli a contatore sono MOLTO FREQUENTI nei programmi
- Per questo, tutti i linguaggi di programmazione procedurale forniscono un'istruzione per gestire i cicli a contatore
- Questa è l'istruzione `for`

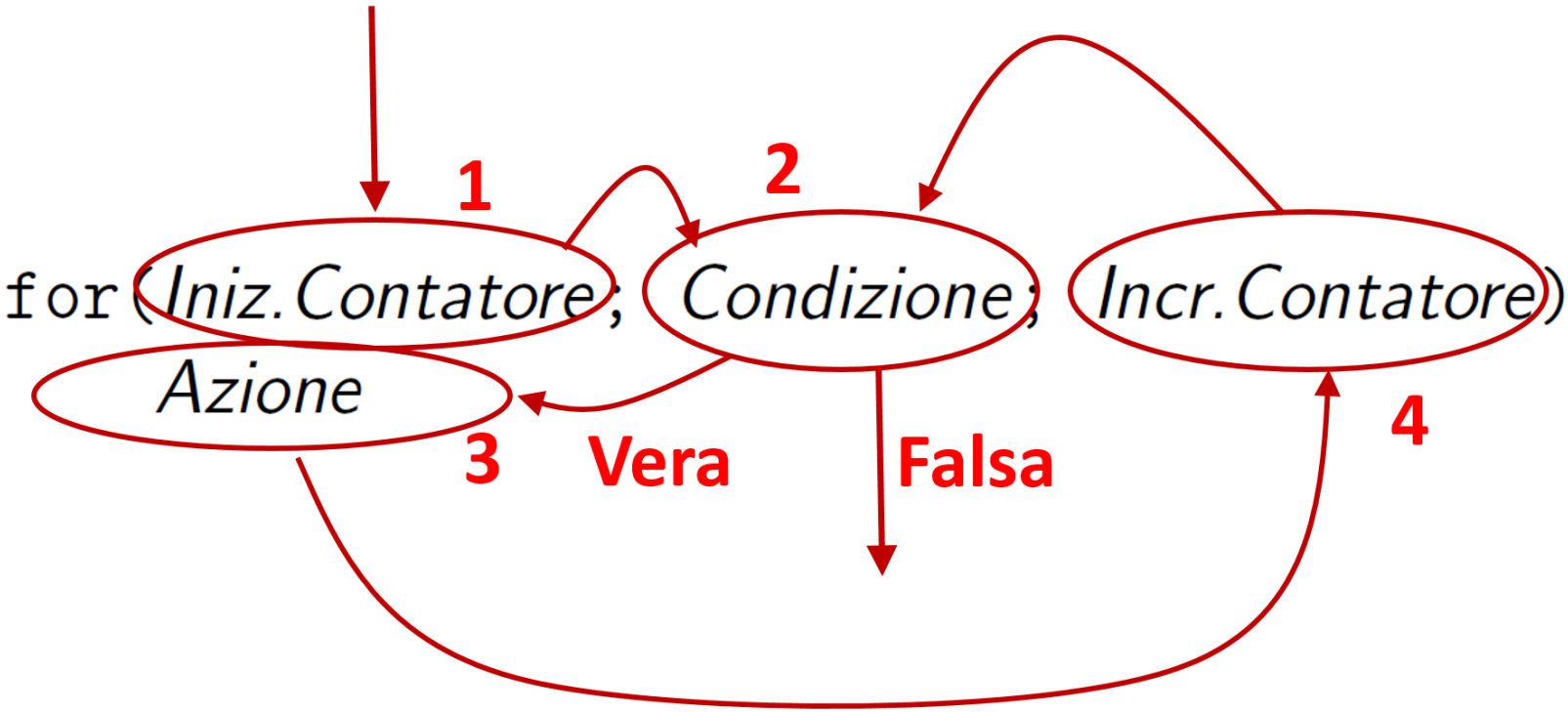
for

Sintassi:

`for(`*Iniz.Contatore*`;` *Condizione*`;` *Incr.Contatore*`)`
Azione

- Per prima cosa, si esegue *Iniz.Contatore*
  che assegna un valore iniziale alla variabile che conta quante volte ripetere l'*Azione*
- Quindi, si valuta la *Condizione*:
  se è **FALSA**, si passa oltre
  se è **VERA**, si esegue l'*Azione*
- Terminata l'esecuzione dell'*Azione*, si esegue *Incr.Contatore*, per incrementare la variabile contatore
- quindi, si passa a valutare di nuovo la *Condizione*

for


1
2
for (Iniz. Contatore; Condizione; Incr. Contatore)
Azione
3
Vera
Falsa
4

Esempio: Somma dei Primi n Numeri Positivi

- Adattiamo il precedente esempio
- usando l'istruzione `for`

Programma: Somma_Primi_Numeri_02.cpp

```
int main()
{
    int n;
    int somma = 0;
    int i;

    cout << "inserisci un valore" << endl;
    cin >> n;

    for(i=1; i <= n; i++)
    {
        somma += i;
    }

    cout << "Somma primi " << n << " numeri=" << somma;

    return 0;
}
```

Vantaggi

- Nelle parentesi tonde del `for` troviamo tutto ciò che serve per capire **quante volte verrà ripetuto il ciclo**
- L'azione che viene ripetuta **non contiene nient'altro che ciò che deve essere ripetuto**
- Se usato bene, consente di tenere rigorosamente separate l'attività di coordinamento del ciclo e l'azione da ripetere

do ... while

Almeno una Ripetizione

- Quando l'azione del ciclo va ripetuta **almeno una volta**,
- ma a priori **non si sa quante volte andrà ripetuta**
- allore è meglio usare la terza istruzione per i cicli: `do ... while`

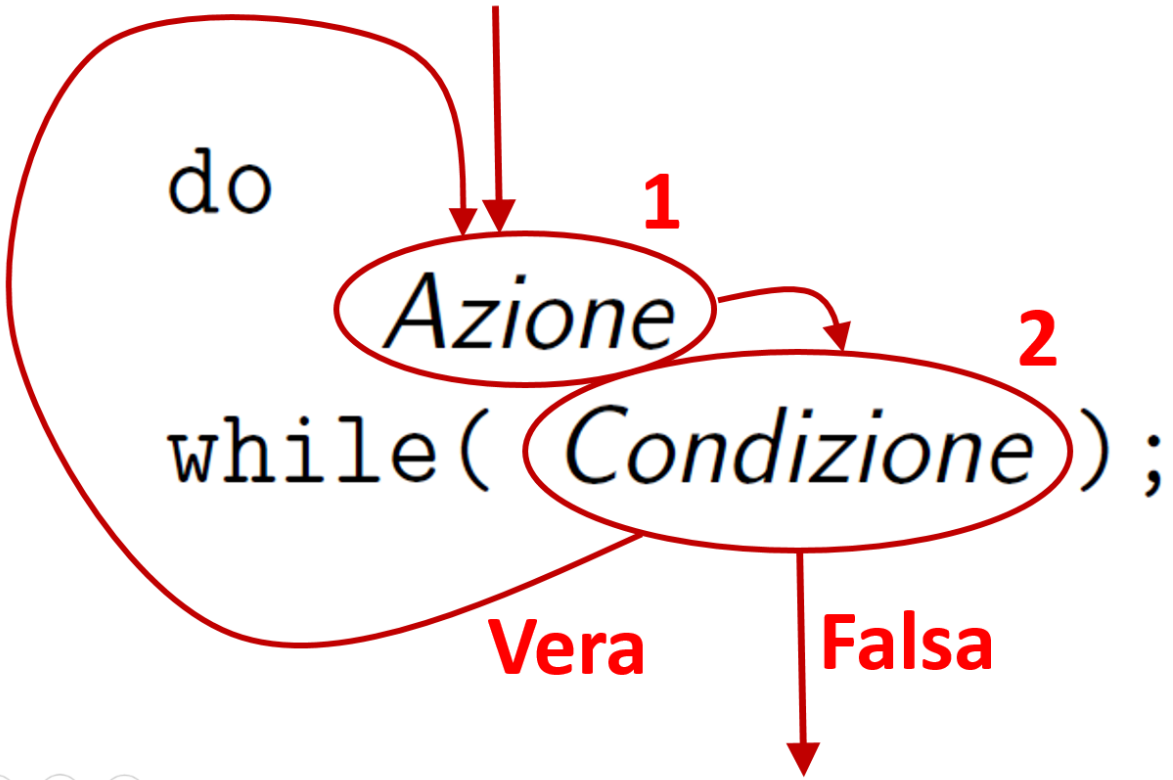
do ... while

Sintassi:

```
do
      Azione
while( Condizione );
```

- Per prima cosa, si esegue l'*Azione*
- Quindi, si valuta la *Condizione*:
  se è **FALSA**, si prosegue oltre
  se è **VERA**, si esegue di nuovo l'*Azione*
- terminata l'*Azione*, si passa a valutare di nuovo la *Condizione*, ecc.

do ... while



Programma: Divisione_per_3_bis.cpp

```
int main()
{
    int v;

    do
    {
        cout << "inserisci un valore" << endl;
        cin >> v;
    } while( v % 3 != 0 );
    cout << "OK" << endl;

    return 0;
}
```

do ... while

- Notate come il programma si semplifica
- La variabile `procedi` non serve più
- Il ciclo continua a iterare se la divisione per 3 dà resto diverso da 0
- Se si esce dal ciclo, vuol dire che il valore inserito è divisibile per 3

break e continue

- Il linguaggio mette a disposizione due istruzioni per **forzare** il comportamento del ciclo
- `break`
  forza l'uscita dal ciclo, senza passare dalla condizione
- `continue`
  forza la valutazione della condizione, senza terminare l'azione

Programma: Divisione_per_3_ter.cpp

```
int main()
{
    int v;

    while( 1 )
    {
        cout << "inserisci un valore" << endl;
        cin >> v;
        if( v % 3 == 0 )
        {
            cout << "OK" << endl;
            break;
        }
    }

    return 0;
}
```

break e continue

- Funziona, ma
- Non si può dire che venga fuori un bel programma
- `while(1)`
  sembra un ciclo **infinito**, ma non lo è
- La logica di funzionamento del ciclo è nascosta nell'azione, senza che sia evidente nella condizione
- Veramente **BRUTTO**

Approccio ad Accumulo

Approccio ad Accumulo?

- Si risolvono con questo approccio quei problemi che richiedono di "**accumulare**" passo-passo i valori di una sequenza.
- Ad ogni passo, si ha il risultato parziale; non serve mantenere l'intera sequenza in memoria.

Somma di Numeri

- Questo è un esempio tipico.
- Si definisce una variabile per memorizzare la somma parziale, inizializzata a zero. Per esempio, `somma`.
- Ad ogni passo, la variabile viene incrementata con il valore della sequenza considerato in quel passo.
- Alla fine, la somma parziale è la somma complessiva.

Massimo Valore

- Anche questo è un problema da risolvere con l'approccio ad accumulo.
- Si definisce una variabile per memorizzare il massimo valore parziale. Per esempio, `max`.
- Il primo valore della sequenza è il primo massimo parziale.
- Ad ogni passo successivo, la variabile `max` viene confrontata con un nuovo valore: se questo è maggiore del precedente valore di `max`, esso ne diventa il nuovo valore, altrimenti `max` rimane inalterata.
- Alla fine, il massimo valore parziale è il massimo valore complessivo.

Oltre ai Cicli ...

- Spesso, in base a valori singoli di una variabile, si devono svolgere azioni diverse
- Una soluzione è mettere tanti `if` in sequenza
- Ma si può usare un'istruzione specifica, chiamata `switch`

switch

Esempio: Mini-Calcolatrice

- Il programma legge due numeri e chiede l'operazione da fare
- In base alla scelta dell'utente, svolge l'operazione corrispondente

Programma: Calcolatrice.cpp

```
int main()
{
    int a;
    int b;
    int r;
    int scelta;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    cout << "1 - Somma" << endl;
    cout << "2 - Differenza" << endl;
    cout << "3 - Prodotto" << endl;
    cout << "Scelta: ";
    cin >> scelta;
```

Programma: Calcolatrice.cpp

```
if( scelta == 1 )
{
    r = a + b;
    cout << "Somma = " << r;
}
if( scelta == 2 )
{
    r = a - b;
    cout << "Differenza = " << r;
}
if( scelta == 3 )
{
    r = a * b;
    cout << "Prodotto = " << r;
}
if( scelta < 1 || scelta > 3)
   cout << "Scelta non valida";

return 0;
}
```

switch

```
switch(Espressione)
{
case Valore1 :
        Azione1
        break;
...
case ValoreN :
        AzioneN
        break;
default:
        AzioneDefault
        break;
}
```

Programma: Calcolatrice_02.cpp

```
switch( scelta )
{
 case 1:
    r = a + b;
    cout << "Somma = " << r;
    break;
 case 2:
    r = a - b;
    cout << "Differenza = " << r;
    break;
case 3:
    r = a * b;
    cout << "Prodotto = " << r;
    break;
 default:
   cout << "Scelta non valida";
   break;
}
```

switch

- ATTENZIONE a non dimenticare i `break;`
- Se omessi, i vari rami `case` si fondono
- al termine dell'esecuzione di un ramo `case`, senza `break` si esegue il successivo

Programma: Calcolatrice_03.cpp

```
switch( scelta )
{
 case 1:
    r = a + b;
    cout << "Somma = " << r;
//          break;
 case 2:
    r = a - b;
    cout << "Differenza = " << r;
    break;
case 3:
    r = a * b;
    cout << "Prodotto = " << r;
    break;
 default:
   cout << "Scelta non valida";
   break;
}

return 0;
```

Ciclo Infinito?

Programma: Infinito.cpp

```
int main()
{
    int i;
    int a;

    for( i=0; i >= 0; i++)
        a=1;

    cout << "i = " << i ;

    return 0;
}
```

Ciclo Infinito?

- In teoria sì
- In pratica NO, perchè il numero di bit a disposizione è FINITO
- Quindi a un certo punto il contatore raggiunge il massimo valore rappresentabile
- Sommando 1 a quel valore ... si ottiene il valore più piccolo rappresentabile (con 32 bit), che è negativo

Il Calcolo del Massimo

Sfida:

- Scrivere un programma che legge una sequenza di numeri interi terminata dal valore 0
- Terminata la lettura, il programma stampa il valore massimo (numericamente più grande) tra quelli letti
- Se il primo valore letto è già 0, stampa `"SEQUENZA VUOTA"`

Programma: Massimo_01.cpp

```
int n, massimo, primo = 1;

do
{
    cout << "INSERIVE UN VALORE ";
    cin >> n;
    if( n!=0)
    {
       if( primo )
       {
          massimo = n;
         primo = 0;
       }
       else
       {
         if( n > massimo )
            massimo = n;
       }
    }
} while( n != 0);
```

Il Calcolo del Massimo

Programma: Massimo_01.cpp (parte finale)

```
if( primo )
    cout << "SEQUENZA VUOTA" << endl;
 else
    cout << "Massimo=" << massimo << endl;
```

Somme dei Primi n Numeri

Sfida:

- Scrivere un programma che legge da tastiera una sequenza di numeri interi, terminata dal valore 0
- per ciascun valore `n` inserito, se questo è maggiore di 0, il programma calcola la somma dei primi `n` numeri

Programma: Somme_Primi_Numeri.cpp

```
int main()
{
   int n;
   int i, somma;

   do
   {
       cout << "INSERIVE UN VALORE ";
       cin >> n;
       if( n>0)
       {
           somma = 0;
           for(i=1; i <= n; i++)
              somma += i;
            cout << "Somma: " << somma << endl;
       }
   } while( n != 0);

   return 0;
}
```

Calcolatrice

Sfida:

- Prendete il programma della mini-calcolatrice visto prima.
- Modificatelo per fare in modo che:
  - il menu di scelta dell'operazione sia in un ciclo,
  - in modo tale che se l'utente inserisce una scelta non valida, venga riproposto il menu, senza procedere con il resto del programma.

Calcolatrice_02_bis.cpp

```
cout << "inserisci due valori" << endl;
cin >> a;
cin >> b;

do
{
    cout << "1 - Somma" << endl;
    cout << "2 - Differenza" << endl;
    cout << "3 - Prodotto" << endl;
    cout << "Scelta: ";
    cin >> scelta;
} while(scelta < 1 || scelta > 3);

switch( scelta )
```