

Istruzioni Iterative del Linguaggio C/C++

Semplice programma

- Scrivere un programma che, dato un numero naturale N , letto a tempo di esecuzione del programma stesso, stampi i primi N numeri naturali
- Al momento non sappiamo scriverlo!

Istruzioni iterative

- Le istruzioni iterative (o di iterazione) forniscono strutture 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 al blocco e a un'istruzione di scelta) per implementare qualsiasi algoritmo
- Tuttavia, per migliorare l'espressività del linguaggio, vengono rese disponibili vari tipi di istruzioni iterative:
 - **for (; ;)**
 - **while ()**
 - **do ... while ()**

Primo problema iterativo

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

Proviamo a pensare ad una soluzione ...

Rappresentazione dati

- Una variabile N che rappresenti il numero naturale N dato
- Una variabile i che rappresenta un ausilio per “scorrere” tutti i valori naturali da 1 fino a N

Algoritmo e Programma

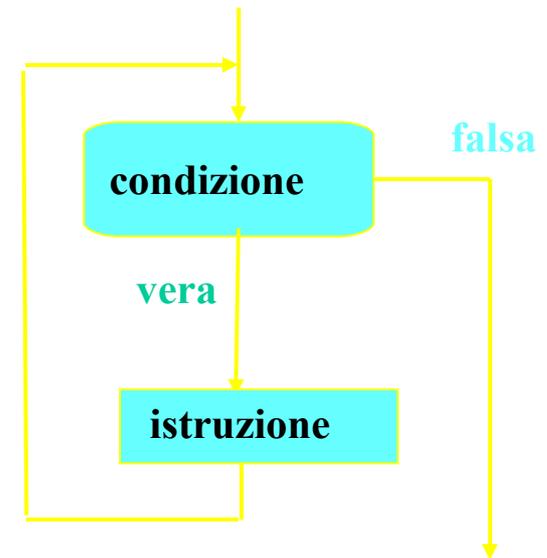
- Inizialmente, i vale 1
- Finché $i \leq N$, ripetere:
 - stampare il valore attuale di i
 - incrementare di 1 il valore attuale di i

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

Istruzione iterativa “while”

<istruzione-while> ::=

while (*condizione*) *<istruzione>*



- *<istruzione>* viene ripetuta per tutto il tempo in cui l'espressione *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

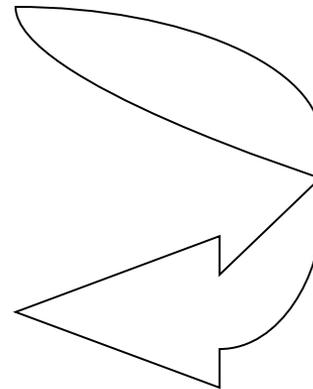
Osservazioni

- Direttamente o indirettamente, **<istruzione> deve modificare prima o poi la condizione,** altrimenti si ha un ciclo infinito
- Per questo motivo, **molto spesso <istruzione> è in realtà un blocco,** che contiene tra le altre una **istruzione di modifica** di qualche variabile che compare nella *condizione*

Esempio (precedente)

```
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++; }  
}
```



Istruzione di modifica
della condizione del while

Problema più complesso

- Dato un numero naturale N , calcolare la corrispondente somma dei primi N numeri naturali
- Calcolare il valore: $S = 1 + 2 + \dots + N$

Rappresentazione dati

- Una variabile N che rappresenti il numero naturale N dato
- Una variabile **somma** che rappresenti la somma calcolata
- Una variabile i che rappresenti un ausilio per “scorrere” tutti i valori naturali da 1 fino a N

Algoritmo

- Inizialmente, **somma** vale 0, **i** vale 1
- Finché **$i \leq N$** , ripetere:
 - aggiungere a **somma** il valore attuale di **i**
 - incrementare di 1 il valore attuale di **i**

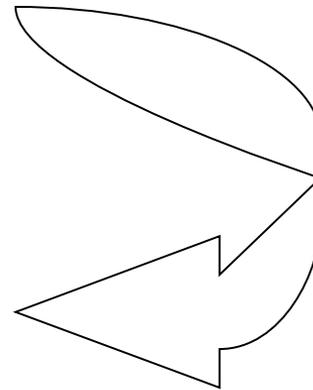
(Quasi) Programma

```
main ()  
{  
    int i = 1, somma = 0, N;  
    cin>>N;  
    finché resta vero che (i<=N),  
        ripetere il blocco { somma = somma+i; i++; }  
}
```

Ultimo passo

```
main ()  
{  
    int i = 1, somma = 0, N;  
    cin>>N;  
    finché resta vero che (i<=N),  
        ripetere il blocco { somma = somma+i; i++; }  
}
```

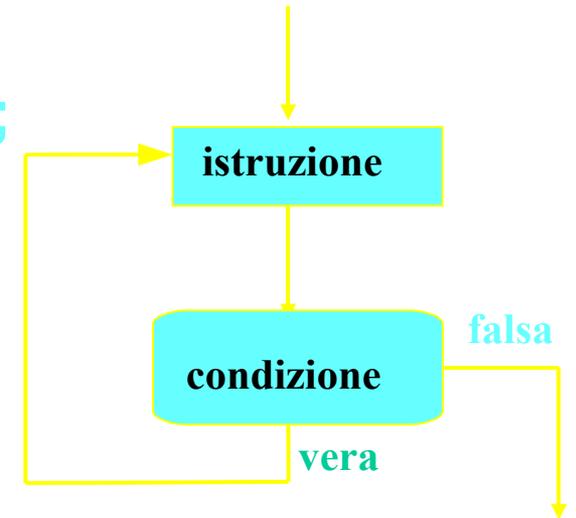
```
main ()  
{  
    int i = 1, somma = 0, N;  
    cin>>N;  
    while (i<=N) { somma = somma+i; i++; }  
}
```



Istruzione di modifica
della condizione del while

Istruzione iterativa “do...while”

<istruzione-do-while> ::=
do < istruzione> while (condizione) ;



- È una “variazione sul tema” dell’istruzione “while”
- A differenza del “while”, la *condizione* è controllata **dopo** aver eseguito <istruzione>

Quindi il ciclo viene sempre eseguito almeno una volta

Osservazioni

- 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
- **È adatta** a quei casi in cui, per valutare *condizione*, è necessario aver già eseguito <istruzione> (esempio tipico: **Controllo valori di input**)
- **Non è adatta** a quei casi in cui il ciclo può non dover essere *mai eseguito*.

Controllo dei 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” 1/2

- **Non** è l’istruzione per implementare cicli “definiti” come in altri linguaggi
 - Es: “ripeti per *i* che va da 1 a N”
- **È una evoluzione dell’istruzione *while***, rispetto a cui mira ad eliminare alcune frequenti sorgenti di errore:
 - mancanza delle necessarie *inizializzazioni delle variabili*
 - mancanza dell’*istruzione di modifica della condizione del ciclo*
→ rischi di cicli infiniti

Istruzione iterativa “for” 2/2

- Per questo, l'istruzione *for* comprende esplicitamente:
 - una *inizializzazione* che può essere una *definizione* (solo C++) o una *espressione*
 - una *condizione* che deve essere vera ad ogni iterazione
 - una *espressione di modifica* del ciclo

Istruzione iterativa “for” (esempio pratico)

```
#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;
}
```

AVENIO 10-3

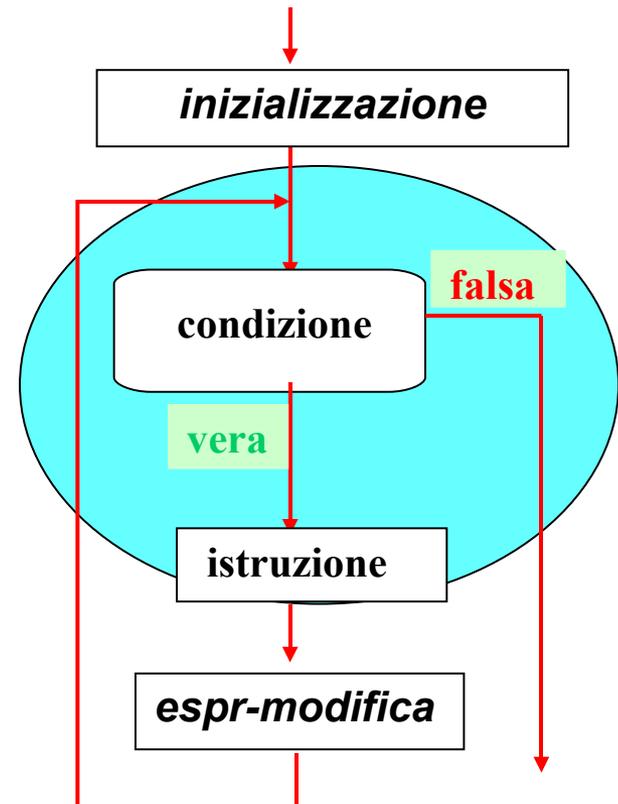


Istruzione iterativa “for” (sintassi)

<istruzione-for> ::=

for (*definizione* | *espr-inizializz.* ; *condizione* ; *espr-modifica*)
 <istruzione>

Stessa
struttura
del while



Esempio precedente

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

- Implementare (tradurre) mediante ciclo for

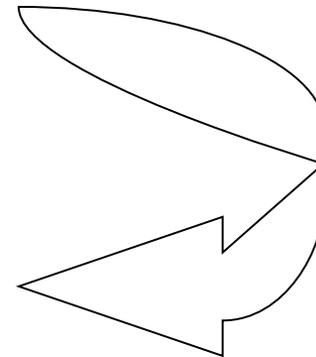
Soluzione

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

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

Espressione di
inizializzazione
del ciclo

Espressione di modifica
della condizione del for



Oppure, solo in C++

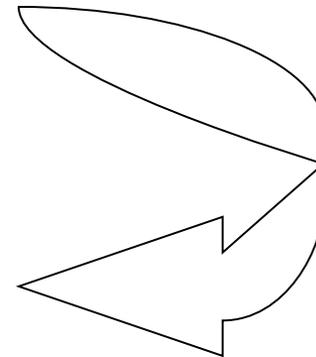
```
main ()  
{  
  int N;  
  cin>>N;  
  for (int i=1 ; i<=N; i++)  
    cout<<i<<endl;  
}
```

Esempio più complesso

```
main ()  
{  
    int i, somma = 0, N;  
    cin>>N;  
    finché resta vero che (i<=N),  
        ripetere il blocco { somma = somma+i; i++; }  
}
```

```
main ()  
{  
    int i, somma = 0, N;  
    cin>>N;  
    for (i=1 ; i<=N; i++) somma = somma+i;  
}
```

Espressione di
inizializzazione
del ciclo



Espressione di modifica
della condizione del for

Oppure, solo in C++

```
main ()  
{  
    int somma = 0, N;  
    cin>>N;  
    for (int i=1 ; i<=N; i++)  
        somma = somma+i;  
}
```

for() e while() a confronto

```
main ()
{
  int N = 6;
  int i = 1, somma = 0;
  while (i<=N) { somma += i; i++; }
}
```

```
main ()
{
  int N = 6;
  int somma = 0;
  for (int i=1; i<=N ; i++)  somma += i;
}
```

Osservazioni

- Per inizializzare (o modificare) più variabili si può usare una *definizione o espressione composta* (operatore ,)

```
for ([int] i=1, somma = 0 ; i<=N ; i++)  
    somma = somma + i ;
```

- L'espressione di modifica della condizione appare in modo più evidente (la sua mancanza "si nota"!)


L' <istruzione> che compare nel corpo del ciclo è *solo l'operazione vera e propria da ripetere*:

- migliore leggibilità
 - (spesso non è più necessario un blocco)
- Le variabili definite nell'inizializzazione del *for* possono essere utilizzate solo nel corpo del *for*

Osservazioni

- Ognuna delle espressioni può essere omessa, ma il separatore ; deve rimanere

Se manca *condizione*, la si assume sempre vera

- Esempio: ciclo infinito

```
for ( ; ; ) <istruzione>
```

Esercizio (*Specifica*)

- Leggere da input due valori naturali, calcolarne il prodotto come sequenza di somme e stampare il risultato su video.
- Se ci riusciamo, dopo aver ovviamente definito l'algoritmo, proviamo a scrivere il programma solo carta e penna

Esercizio (*Algoritmo “banale”*)

- **Idea:**

- Dati i due numeri x e y , sommare y a y , x volte

- **Algoritmo:**

- Leggo i valori da input
- Utilizzo una variabile ausiliaria n , inizializzata al valore di x , che mi serve come contatore del numero di somme delle y
- Utilizzo anche una variabile ausiliaria P , inizializzata a 0, che mi serve per memorizzare le somme parziali delle y
- Effettuo n somme di y , mettendole in P
- Al termine, il risultato sarà contenuto in P

Esercizio (*Rappresentazione informazioni*)

- Servono 2 variabili (*int*) per rappresentare i valori da moltiplicare: **x**, **y**
- Servono, poi, due variabili ausiliarie (*int*) per rappresentare l'indice delle somme e le somme parziali: **n**, **P**
- Possiamo usare le istruzioni `while` o `for`
 - Provare con entrambe

Esercizio (*Programma*)

```
main()
```

```
{
```

```
  int x, y, n, P;
```

```
  cin>>x;
```

```
  cin>>y;
```

```
  P=0; n=x;
```

```
  while (n>0)
```

```
  { P=P+y;
```

```
    n--;
```

```
  }
```



```
  P=0;
```

```
  for (n=x; n>0; n--)
```

```
    P=P+y;
```

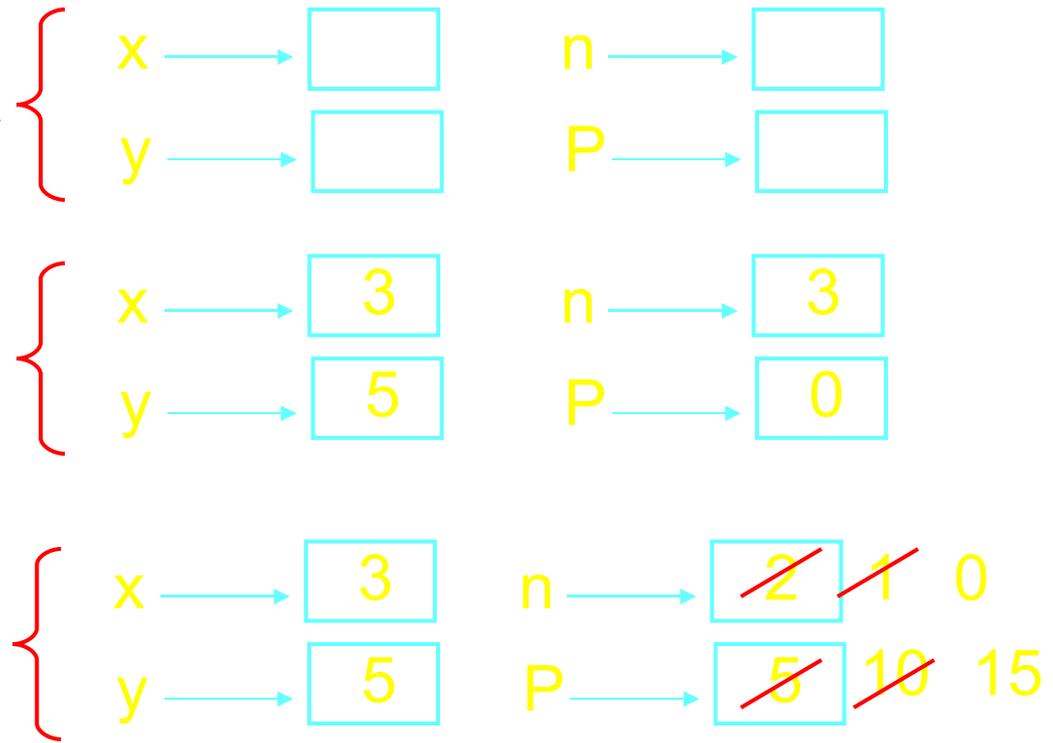
```
  cout<<x<<" * "<<y<<" = "<<P<<endl;
```

```
}
```

Esercizio (*verifica per un caso*)

main()

```
{
  int x, y, n, P;
  cin>>x;
  cin>>y;
  P=0; n=x;
  while (n>0)
  { P=P+y;
    n--;
  }
  cout<<x<<" * "<<y<<" =
    "<<P<<endl;
}
```



Output: 3 * 5 = 15

Esercizio (*Algoritmo “intelligente”*)

- **Idea!!:**

- Se i due numeri sono molto distanti fra di loro, ...
- Quindi, dati i due numeri x e y , conviene controllare chi è il maggiore, e sommare il maggiore tante volte quante sono indicate dal minore

- **Algoritmo:**

- Leggo i valori da input
- Calcolo il maggiore tra x e y
- Utilizzo una variabile ausiliaria n , inizializzata al valore del minore, che mi serve come contatore della somme del maggiore
- Utilizzo anche una variabile ausiliaria P , inizializzata a 0, che mi serve per memorizzare le somme parziali
- Effettuo n somme del numero maggiore, mettendole in P
- Al termine, il risultato sarà contenuto in P

Esercizio

- **Scrivere un programma che legga un numero intero non negativo in ingresso e lo divida progressivamente per 10, finché non si riduce al valore 0. Il programma stampa il numero iniziale ed il risultato intermedio di ogni divisione**
- **Esempi:**
Immettere un numero intero non negativo: 145
145 14 1
Immettere un numero intero non negativo: 0
0

Tentativo 1/2

```
main ()  
{  
    int i ;  
    cout<<Immettere un numero intero non negativo: " ;  
  
    cin>>i ;  
  
    ...  
    cout<<i<<"\t" ;  
    i /= 10 ;  
  
    ...  
    cout<<endl ;  
}
```

Tentativo 2/2

```
main ()
{
  int i ;
  cout<<Immettere un numero intero non negativo: " ;

  cin>>i ;
  while ( i > 0) {
    cout<<i<<"\t" ;
    i /= 10 ;
  }
  cout<<endl ;
}
```

Cosa viene stampato se si legge 0 dallo stdin?

Possibile soluzione

```
main ()
{
    int i ;
    cout<<Immettere un numero intero non negativo: " ;

    cin>>i ;
    do {
        cout<<i<<"\t" ;
        i /= 10 ;
    } while (i > 0) ;
    cout<<endl ;
}
```

Esercizio: stampa numero al contrario

- Scrivere un programma che legga un numero intero non negativo in ingresso e lo stampi al contrario
- Esempi:
Immettere un numero intero non negativo: 164
461
Immettere un numero intero non negativo: 0
0

Suggerimenti

- **Supponiamo di contare l'*ordine* delle cifre di un numero a partire da destra e dall'indice zero**
- **Nel precedente esercizio**
 - Che proprietà ha la cifra di ordine 0 del numero contenuto nella variabile i dopo d divisioni ?
 - Forse è la cifra di ordine d del numero iniziale?
- **Con quale operatore si 'cattura' la cifra di ordine 0 di un numero n in base 10 ?**

Algoritmo 1/2

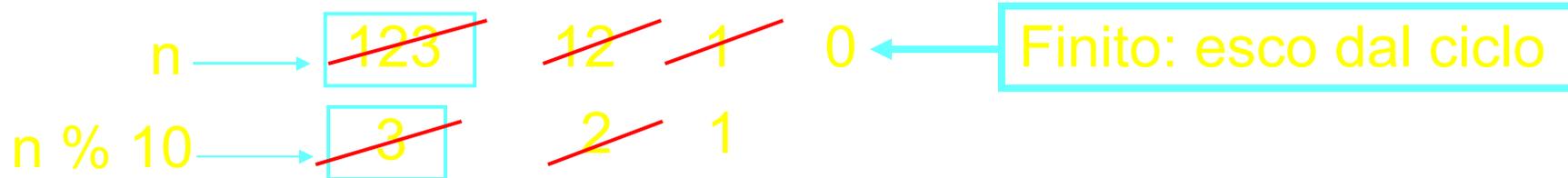
- Leggo il valore n da input (es: 123)
- Stampo la cifra di ordine 0, che mi è data da $n \% 10$ (= $123 \% 10 = 3$)
- Divido n per 10 (dopodiché $n = 123 / 10 = 12$)
 - *NOTA: se n contiene solo una cifra, ossia $n < 10$, allora, $n / 10 == 0$, e viceversa*

Algoritmo 2/2

1. Finché $n > 0$

a) *Stampo $n \% 10$*

b) *Divido n per 10, ossia $n = n / 10$*



Possibile soluzione

```
main()
{
    int n;

    cout<<"Inserire un numero intero non negativo : " ;
    cin>>n ;

    do
    {
        cout<<n % 10;
        n /= 10;
    } while (n>0);

}
```

Istruzioni “break” e “continue”

- Istruzioni senza argomenti che modificano il normale percorso dei cicli e dell'istruzione switch
- Il “**break**” provoca l'immediata uscita da un blocco o ciclo
 - L'istruzione eseguita dopo il “break” è quella successiva all'istruzione composta in cui compare il “break”
 - L'utilizzo del “break” è già stato visto insieme all'istruzione “switch ... case”
 - All'interno di cicli, serve soprattutto per interrompere l'esecuzione in caso di errori o di condizioni irregolari
- Il “**continue**” si può utilizzare solo nel corpo di un ciclo: fa saltare l'esecuzione delle istruzioni del blocco successive (*come se fosse un salto alla parentesi } che chiude il blocco*), quindi causa la nuova valutazione dell'espressione condizionale e l'eventuale inizio della prossima iterazione

Esempio “break”

```
main()
{
  int x, y, n, P;

  cin>>x>>y;
  P=0; n=x;
  while (n>0)
  { P=P+y;
    if (P>250000) break;
    n--;
  }
  cout<<x<<" * "<<y<<" = "<<P<<endl;
}
```



Esempio “continue”

```
main()
{
  int x, y, n, P, min_k=3, max_k=12;
  cin>>x>>y;
  P=0; n=x;
  while (n>0)
  { P=P+y;
    if ( (P>min_k) && (P<max_k) ) continue;
    n--;
  }
  cout<<x<<" * "<<y<<" = "<<P<<endl;
}
```

Questo programma non risolve alcun problema concreto. Tuttavia, per esercizio, calcolare cosa viene stampato per $x \leftarrow 3$ e $y \leftarrow 2$