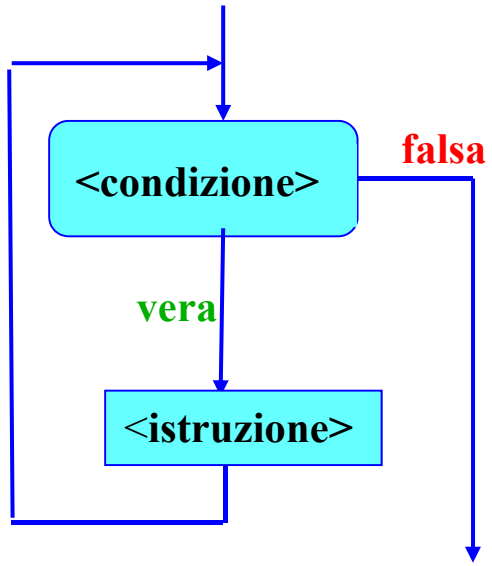
Ripasso: Istruzione iterativa `while`

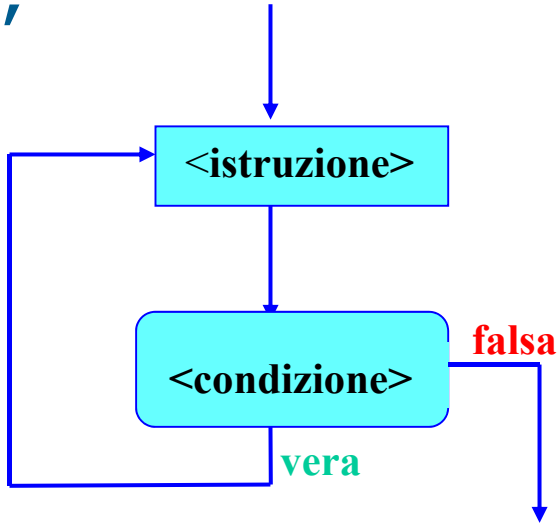
`while` (<*condizione*>) <*istruzione*>



- <*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

Ripasso: Istruzione iterativa **`do/while`**

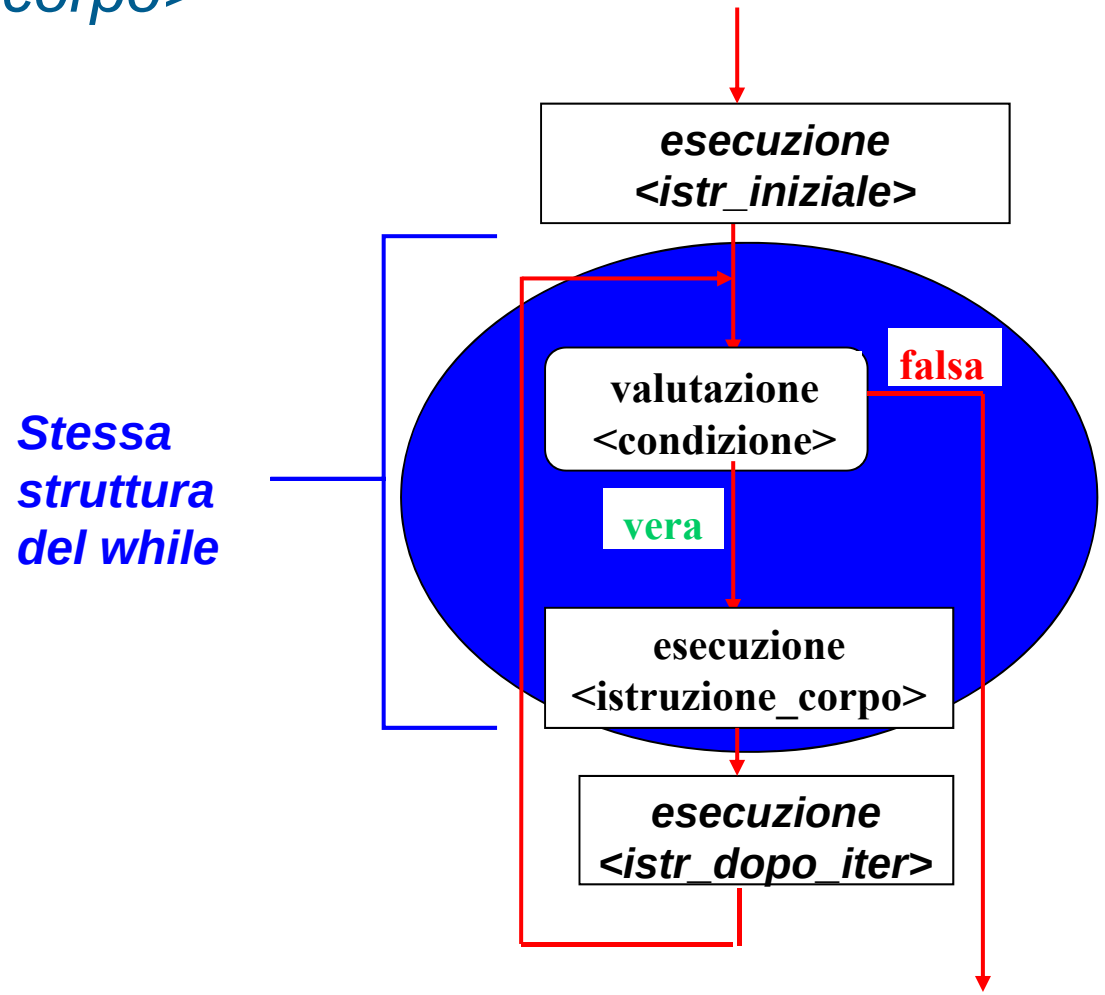
`do` <*istruzione*> **`while (`**<*condizione*>**`) ;`**



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

Ripasso: Istruzione iterativa `for`

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



Ripasso: Istruzione iterativa `for`

- 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++)
```

Riprendiamo dal problema iniziale

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

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

Confronto for/while

```
main()
{
   int N, somma = 0;
   cin>>N;

   for(int i = 1 ; i <= N ; i++)
      somma += i;

   cout<<somma<<endl;
}
```

for

Forma più compatta
Inizializzazione della variabile di appoggio e suo incremento incluse nell'intestazione

while

Inizializzazione della variabile di appoggio prima del ciclo e suo incremento dentro il corpo del ciclo

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

Esercizio (1)

Scrivere un programma che

- Legga da tastiera un intero N
- Calcoli il fattoriale di N

Il fattoriale di un numero naturale N si calcola come:

N! = N * (N-1) * (N-2) *...* 2 * 1

inoltre, per definizione:
1!=1
0!=1

Trascurare problemi di overflow.

Esercizio (2)

Scrivere un programma che

- legga un numero intero ***n***.
- Il numero ***n*** viene di nuovo richiesto finché non è compreso tra 0 ed una costante definita a tempo di scrittura del programma.
- legga ***n*** interi positivi
- se ne calcoli e stampi la somma ed il valore massimo (il valore più grande tra quelli inseriti).

- **VINCOLO**: Realizzare il calcolo prima con e poi senza l'istruzione **for**. Trascurare problemi di overflow.

Omissioni nell'intestazione del ciclo

- Sia ognuna delle due istruzioni che la condizione previste nell'intestazione del ciclo **`for`** possono essere omesse
  - Il separatore **`;`** deve rimanere
  - **Se manca** la condizione, la si assume **sempre vera**
- Esempi
  - Equivalente del **`while`**:
    - **`for ( ;`** *<condizione>* **`; )`** *<istruzione>*
  - Ciclo infinito:
    - **`for ( ; ; )`** *<istruzione>*

Cicli annidati

- Il corpo di un ciclo può a sua volta contenere altri cicli
  - Si denota come *annidato* un ciclo contenuto all'interno di un altro ciclo

Modifica esecuzione iterazioni

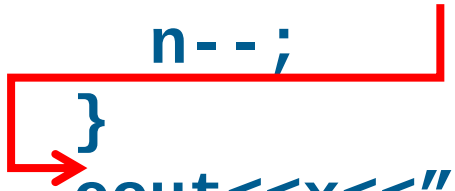
- Vi sono due istruzioni senza argomenti che permettono
  - di uscire immediatamente da un ciclo
    - **`break;`**
    - E' la stessa istruzione già vista per l'istruzione **`switch`**
  - di modificare la normale sequenza di esecuzione di una iterazione
    - **`continue;`**
    - Utilizzabile solo in un ciclo

Istruzione `break;`

- L'istruzione **`break;`** provoca l'immediata uscita da un ciclo o, come sappiamo, dal corpo di uno **`switch`**
- Nel caso di un ciclo, l'istruzione eseguita dopo **`break;`** è quella successiva al corpo del ciclo stesso

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;
 }
```



Flessibilità e pericolo

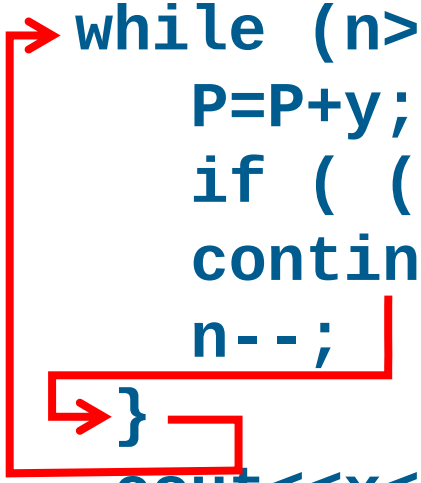
- L'istruzione **`break;`** fornisce quindi una uscita alternativa da un ciclo oltre la valutazione della condizione del ciclo
- Si ha quindi maggiore flessibilità, ma si può rischiare di aumentare la difficoltà di comprensione del programma

Istruzione `continue;`

- L'istruzione **`continue;`** si può utilizzare solo nel corpo di un ciclo
  - Fa saltare alla fine del corpo del ciclo
    - come se fosse un salto alla parentesi } che chiude il blocco
  - quindi causa una nuova valutazione della condizione del ciclo e l'eventuale inizio della prossima iterazione

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←3 e y←2

Istruzione vuota

- E' un semplice **;**
- Non fa nulla
- Sintatticamente è trattata come una qualsiasi altra istruzione

Può tornare utile con un ciclo **for**, perché nell'intestazione del ciclo si eseguono già delle istruzioni. Esempio:

```
int i ;
// legge i da stdin e si ferma solo se i != 0
for (i = 0 ; i == 0 ; cin>>i)
      ; // il corpo non fa nulla
cout<<i<<endl ; // stampa un numero diverso da 0
```

Esercizio (3)

Riprendiamo l'esercizio 2

Scrivere un programma che
- legga un numero intero n.
- Il numero n viene di nuovo richiesto finché non è compreso tra 0 ed una costante definita a tempo di scrittura del programma.
- legga n interi positivi
- se ne calcoli e stampi la somma ed il valore massimo (il valore più grande tra quelli inseriti).

Con dei nuovi vincoli

VINCOLI:
- L'inserimento di uno zero interrompe la sequenza di inserimenti (anche se non sono stati raggiunti gli n numeri)
- L'inserimento di un numero negativo è trascurato (come se non fosse stato inserito nessun numero)
- Usare l'istruzione for
- Incrementare la variabile contatore nell'intestazione del for.

Esercizio (4)

- Scrivere un programma che stampa il numero di secondi trascorsi dal suo avvio, in maniera tale che tale numero si aggiorni al passare del tempo
- Partendo da zero, ogni secondo stampiamo un numero incrementato di uno (una stampa su ogni riga)
- Come facciamo a scandire il passare dei secondi?

Il commando linux `sleep`

- Provare a scrivere su un terminale Linux il comando

```
sleep 5
```

- Il comando aspetta 5 secondi (cioè, blocca il terminale per 5 secondi) e poi termina.

- Potremmo utilizzare questo comando nel nostro programma per realizzare quanto richiesto dall'esercizio 4?

- **PROBLEMA**: `sleep` è un comando del terminale Linux, come `ls`, `cd`, `mkdir`, …
  - **Non è un'istruzione C/C++**

La funzione C/C++ `system`

- Per fortuna, è possibile invocare comandi di sistema da un programma C/C++ con la funzione system

```
system(<comando di sistema>)
```

- Quindi, per esempio, si può invocare il comando sleep del terminale Linux in un programma C/C++ come segue:

```
#include <stdlib.h>

main()
{
  cout<<``\nInizio programma..''<<endl;
  system (``sleep 5'');
  cout<<``\nOra sono passati 5 secondi..''<<endl;
}
```

Occorre includere questa libreria per usare `system`

Cicli infiniti

- Adesso che sappiamo come aspettare un secondo cosa ci manca per eseguire l'esercizio 4?
- **Un modo per ripetere la stampa dei secondi all'infinito..**
- Ma abbiamo già visto due modi per realizzare un ciclo infinito

1. `for ( ; ; ) { … }`
2. `while (true) { … }`

```
#include <stdlib.h>

main()
{
  int i = 0;
  cout<<''\nInizio programma..''<<endl;
  system (''sleep 5'');
```

Terminazione di un processo

- Quindi l'esercizio 4 si può risolvere in questo modo

```
#include <stdlib.h>

main()
{
  int i = 0;
  while (true)
  {
    cout<<i++<<endl;
    system (``sleep 5'');
  }
}
```

Cosa succede se eseguiamo il programma?
Non termina mai.. Come facciamo a terminare il processo?

Terminazione di un processo

- Premere *Ctrl* + *C* nel terminale su cui abbiamo lanciato il nostro processo (programma) manda il segnale SIGINTR (interruzione) al processo in esecuzione
  - A volte tale segnale può non bastare per interrompere il processo
- Chiudere il terminale in cui sta girando un processo fuori controllo di norma causa anche la terminazione del processo
- In casi ancora più complicati, chiudere il terminale non basta (!!!)

Terminazione di un processo

Un metodo che funziona sempre

- Ad ogni processo è associato un identificatore numerico: ***pid***

- Per elencare i propri processi (da terminale):

```
ps x
```

- Per elencare tutti i processi (da terminale)

```
ps ax
```

- Per trovare il processo relativo al proprio programma cercare il processo il cui nome è uguale a quello dell'eseguibile

Terminazione di un processo

Un metodo che funziona sempre

- Una volta individuato il ***pid*** del nostro processo
- Comando **`kill`**: spedisce segnali ai processi.
- Per uccidere un processo:

```
kill -9 <pid>
```

- Invocato da un altro terminale

Esercizio (5)

- Scrivere un programma che stampa il numero di secondi trascorsi dal suo avvio, in maniera tale che tale numero si aggiorni al passare del tempo (quindi sempre sulla stessa riga).
- Il programma deve terminare automaticamente quando è trascorso un numero di secondi scelto a tempo di scrittura del programma stesso (o immesso da tastiera)

- **SUGGERIMENTO**: se si mandano all'oggetto cout il carattere speciale `"\r"` seguito da caratteri o valori da stampare, ed infine dal manipolatore **`flush`**, allora tali caratteri o valori saranno stampati all'inizio delle riga corrente (sovrascrivendo il precedente contenuto della riga).

Ad esempio, l'istruzione

```
cout<<"\r"<<"prova"<<flush ;
// equivalente a cout<<"\rprova"<<flush ;
```

stampa «*prova*» all'inizio della riga corrente, sovrascrivendo l'eventuale precedente contenuto (solo) dei primi 5 caratteri di tale riga.

Esercizio (6)

Scrivere un programma che

- legga da input due valori interi
- ne calcoli il prodotto come sequenza di somme
- stampi il risultato su video.

- **VINCOLO**: Usare il numero minore possibile di operazioni (somme) per calcolare il prodotto

Esercizio BONUS (1)

Ancora su overflow

Riprendiamo l'esercizio Calcolatrice

Modificare il programma calcolatrice in modo che:

- Anche l'operazione **sottrazione** sia protetta da overflow
- Anche l'operazione **moltiplicazione** sia protetta da overflow

RICORDATE

Questa operazione è soggetta a overflow

- Per verificare se una somma è rappresentabile con un **int**
- a + b <= MAXINT → a <= MAXINT - b ← Questa è sicura
- E per la moltiplicazione? E per la sottrazione?

NOTA

- MAXINT per il tipo int (con segno) lo conosciamo già: **2147483647**
- Ma si può calcolare nel programma in questo modo

```
int bits = sizeof (<tipo-di-dato>) * 8
int max = exp2 (bits)
```

Calcola 2^{bits}

Per usarla serve includere

```
#include <math.h>
```

Esercizio BONUS (2)

- Riprendiamo l'esercizio visto nell'esercitazione 3

  Leggere da *stdin* un numero intero positivo, ~~che si assume essere compreso tra 100 e 999~~ e stamparlo al contrario
  (con le cifre in ordine inverso)

  Esempi:
  103 → 301
  230 → 032

  Ora rimuoviamo il vincolo sul numero di cifre che compongono il numero.
  Stampiamo al contrario un qualunque numero intero positivo (usare **`unsigned int`**), purché minore di MAXINT

RICORDATE:

- 235 / 10 = 23
- 235%10 = 5

Quando il risultato della divisione è 0 ho finito