Lezione 5

Istruzioni iterative
Operatore virgola

Semplice programma 1/2

- Scrivere un programma che, dato un numero naturale *N*, letto a tempo di esecuzione del programma stesso, stampi *N* volte la stringa **`Ciao mondo`**

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 1/2

- 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 come facciamo per 'contare' il numero di stampe effettuate e capire così quando fermarci?

Idee 2/2

- Ci vorrebbe una variabile da incrementare ad ogni stampa ...
- Proviamo a tradurre tutte queste idee in un algoritmo

Verso un algoritmo

- Ci serve una variabile inizializzata al valore 1 (o, se preferite, 0)
- Il valore di tale variabile deve essere **incrementato di 1** dopo ogni stampa
- Come facciamo a sapere quando dobbiamo fermarci?
  - Ad ogni 'giro' dovremmo <u>confrontare il valore corrente della variabile con *N*</u> 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 *"Ciao mondo"*
  - 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<<"Ciao mondo"<<endl; i++; }
}
```

Programma più complesso

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

Idee 1/2

- Siccome non sappiamo a priori di quante istruzioni di stampa abbiamo bisogno,
  - un'idea sarebbe quella di far ripetere più volte <u>la stessa istruzione</u> di stampa, come nel precedente programma
- Dov'è la differenza rispetto al precedente programma?

Idee 2/2

- La differenza in questo nuovo programma è che
  - **ogni volta** l'istruzione deve stampare un valore **diverso**!
    - potremmo allora far stampare a tale istruzione il valore di una <u>variabile</u>
    - l'importante è che **dopo ogni stampa** tale valore venga **incrementato**!

Verso un algoritmo 1/2

- 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?
  - Come nel precedente programma, ad ogni 'giro' dovremmo confrontare il valore corrente della variabile con *N* per capire se siamo o meno andati oltre

Verso un algoritmo 2/2

- 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*
    - Ossia, in questo nuovo programma possiamo usare tale variabile sia per la stampa che per capire quando fermarci!

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 mancante in entrambi i programmi?
- 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

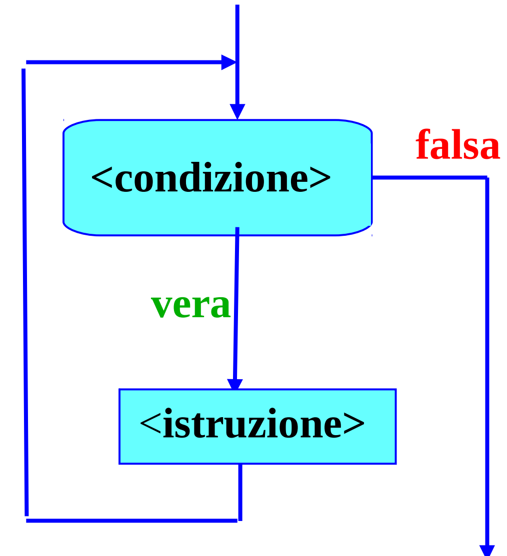
- L'istruzione da ripetere fintanto che la condizione rimane vera viene tipicamente chiamata **corpo del ciclo**
  - A seconda dell'istruzione iterativa usata, si parla di corpo del **`while`**, del **`do … while`** o del **`for`**
- Ogni ripetizione dell'esecuzione del corpo del ciclo viene tipicamente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

Istruzione iterativa **`while`**

Istruzione iterativa `while`

<istruzione-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

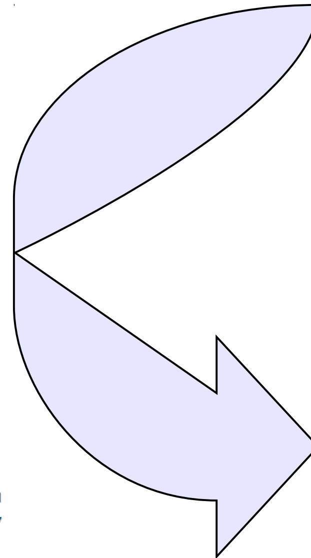
- Direttamente o indirettamente, *<istruzione>* deve modificare prima o poi la condizione, altrimenti si ha un **ciclo infinito**
- Per questo motivo, molto spesso *<istruzione>* è una istruzione composta, che contiene, tra le varie istruzioni, anche un'istruzione di modifica di qualcuna delle variabili 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

Esercizi

- Svolgere la quinta esercitazione fino a *stampa_secondi_trascorsi.cc*

Struttura dati

- Per descrivere i prossimi algoritmi ci conviene introdurre il concetto di struttura dati
- Per ora definiamo una **struttura dati** semplicemente come un insieme di oggetti, ossia di variabili e/o costanti con nome
- Spesso una struttura dati viene utilizzata per **rappresentare i dati del problema reale**, o almeno la parte di dati necessaria per permettere all'algoritmo di risolverlo

Domanda

- Da cosa era composta la struttura dati nel precedente esercizio di stampa dei primi *N* numeri naturali?

Risposta

- Dalle variabili **N** ed **i**

Problema più complesso

- Dato un numero naturale *N*, letto da *stdin*, stampare la corrispondente somma dei primi *N* numeri naturali
- Bisogna cioè calcolare e stampare il valore di: $S = 1 + 2 + \ldots + N$
- Cominciamo come al solito dall'idea
- Ci sarà sicuramente da eseguire un ciclo ...

Analisi 1/4

- Prima di tutto, utilizzando la solita variabile 'contatore' **i**, da incrementare e confrontare con *N*, possiamo gestire il numero di ripetizioni da effettuare
- Ora però dobbiamo riuscire a mettere la somma da qualche parte
- Ci vorrebbe una variabile in cui
  - Se N == 1, ci finisca 1
  - Se N == 2, ci finisca 1 + 2
  - ...

Analisi 2/4

- Supponendo di inizializzare **i** ad 1 e di incrementarlo solo alla fine di ogni iterazione, che valore è memorizzato dentro **i** all'inizio della prima iterazione?
  - Uno
- Ed all'inizio della seconda?
  - Due
- Ed all'*N*-esima?
  - *N*

Analisi 3/4

- Quindi, nel caso di *N* iterazioni:
  - $S = 1 + 2 + 3 + \ldots + N$

Valore di **i** alla prima iterazione

Valore di **i** alla seconda iterazione

Valore di **i** alla terza iterazione

Valore di **i** alla *N*-esima iterazione

- Ma allora, se sommiamo i diversi valori assunti da **i** ad ogni iterazione, non è che otteniamo proprio il valore *S* cercato?

Analisi 4/4

- Come riuscire a ritrovarsi il risultato della somma dei diversi valori assunti da **i** memorizzato all'interno di una qualche variabile?

Idea

- Possiamo memorizzare (accumulare) il valore assunto dalla somma **fino all'iterazione corrente** all'interno di una ulteriore variabile
  - Ad ogni nuova iterazione, aggiungiamo al valore corrente di tale variabile il nuovo valore assunto da **i**
    - Ovviamente continueremo fintanto che **i** sarà minore o uguale di **N**
  - Alla fine dell'ultima iterazione tale variabile conterrà necessariamente (avrà accumulato) la somma di tutti i valori assunti da **i**

Struttura dati

- Ora abbiamo gli elementi per definire la struttura dati da utilizzare
  - Una variabile **N** che rappresenti il numero naturale *N* dato
  - Una variabile **somma** che rappresenti la somma calcolata
  - Una variabile contatore **i** che rappresenti un ausilio per *scorrere* tutti i valori naturali da 1 fino a *N*
- Definite l'algoritmo (senza dimenticare le inizializzazioni!)

Algoritmo

- Inizialmente, **somma** vale 0, **i** vale 1
- Finché **i**<=**N**, ripetere:
  - aggiungere a somma il valore di **i**
  - incrementare di 1 il valore di **i**

Programma quasi completo

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

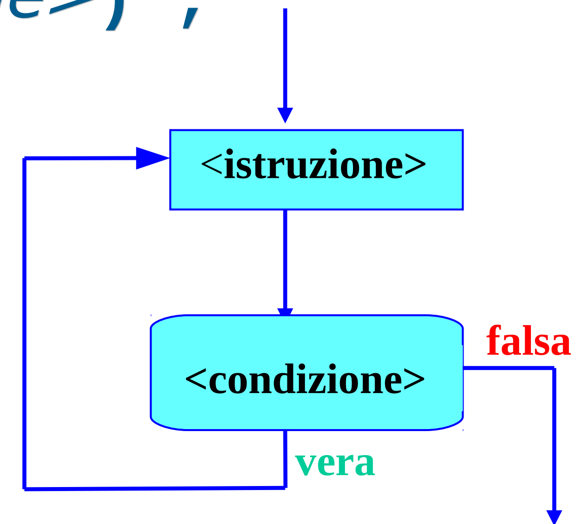
Programma

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

Istruzione iterativa **`do … while`**

Istruzione `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)

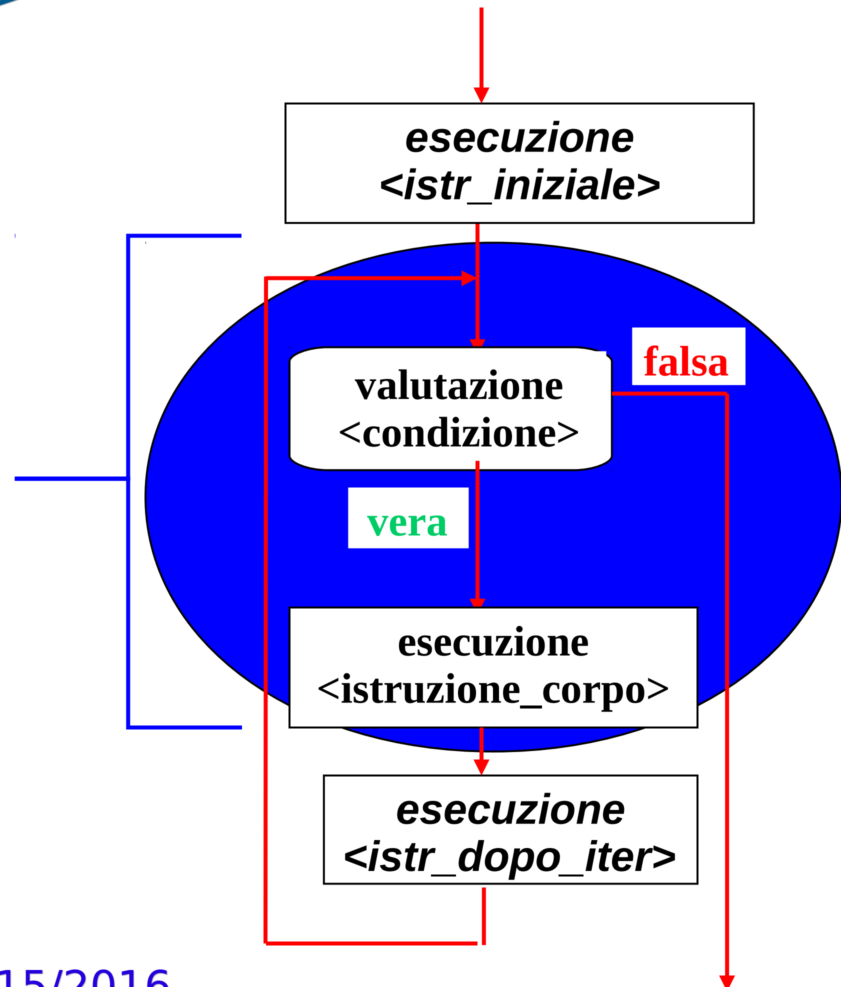
Istruzione iterativa **for**

- 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

<istruzione-for> ::=
for (*<istr_iniziale>* **;** *<condizione>* **;** *<istr_dopo_iter>* **)**
<istruzione_corpo>

Stessa struttura del while



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

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:

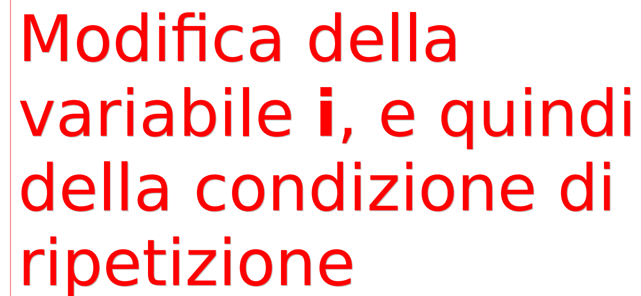
```
main()
{
   int i = 1, N;
   cin>>N;
   finché resta vero che (i<=N),
      ripetere il blocco { cout<<i<<endl; i++; }
}
```

Soluzione con **for**

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

    cin>>N;

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



Nuova forma alternativa

- A partire dallo standard C++11 è disponibile anche una nuova forma per l'istruzione **`for`**
- La vedremo dopo aver introdotto i tipi di dato utilizzati nella definizione di questa nuova forma dell'istruzione **`for`**

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++)
```

Secondo problema

- Completiamo la panoramica risolvendo anche l'altro problema utilizzando l'istruzione **`for`**
- Dato un numero naturale *N*, letto da *stdin*, stampare la corrispondente somma dei primi *N* numeri naturali
- Nella soluzione sfruttare opportunamente la possibilità di definire variabili nell'intestazione del ciclo

Soluzione

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

   cin>>N;

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

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

Confronto for/while 1/2

- Terminiamo confrontando le soluzioni del problema scritte utilizzando le due istruzioni

Confronto for/while 2/2

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

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

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

for

while

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

Semplificazione corpo del ciclo

- Una volta eliminato il bisogno di modificare le variabili di controllo del ciclo all'interno del corpo del ciclo stesso, vi rimane solo l’operazione vera e propria da ripetere:
  - migliore leggibilità
  - spesso non è più necessaria un'istruzione composta

Omissioni nell'intestazione

- 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>*

Esercizi

- Proviamo ora a svolgere un po' di esercizi per familiarizzare ulteriormente con le istruzioni cicliche
  - *somma_e_max_1.cc* e *fattoriale.cc* della quinta esercitazione

Esercizi per casa

- Seguono ora tre esercizi commentati
  - le varie fasi di sviluppo sono descritte in dettaglio
- Si tratta di slide non ancora convertite nel nuovo formato utilizzato in queste lezioni
  - Possono dare problemi in caso di stampa
  - Trovate la versione pronta per la stampa in bianco e nero nel file *Esercizi_casa-Lez_05-bn.ps*

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

```
P=0;
for (n=x; n>0; n--)
        P=P+y;
```

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;
}
```

x → [ ] n → [ ]
y → [ ] P → [ ]

x → [3] n → [3]
y → [5] P → [0]

x → [3] n → [~~2~~] ~~1~~ 0
y → [5] P → [~~5~~] ~~10~~ 15

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,*

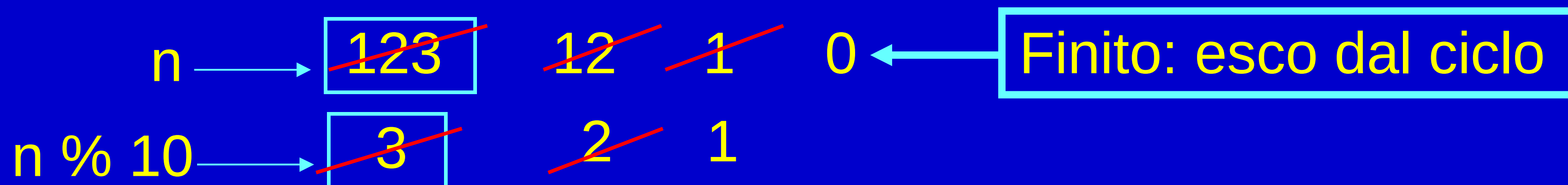
    allora, n / 10 == 0, e viceversa

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);

}
```

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
- Svolgere gli **esercizi sui cicli annidati** della quinta esercitazione

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;
}
```

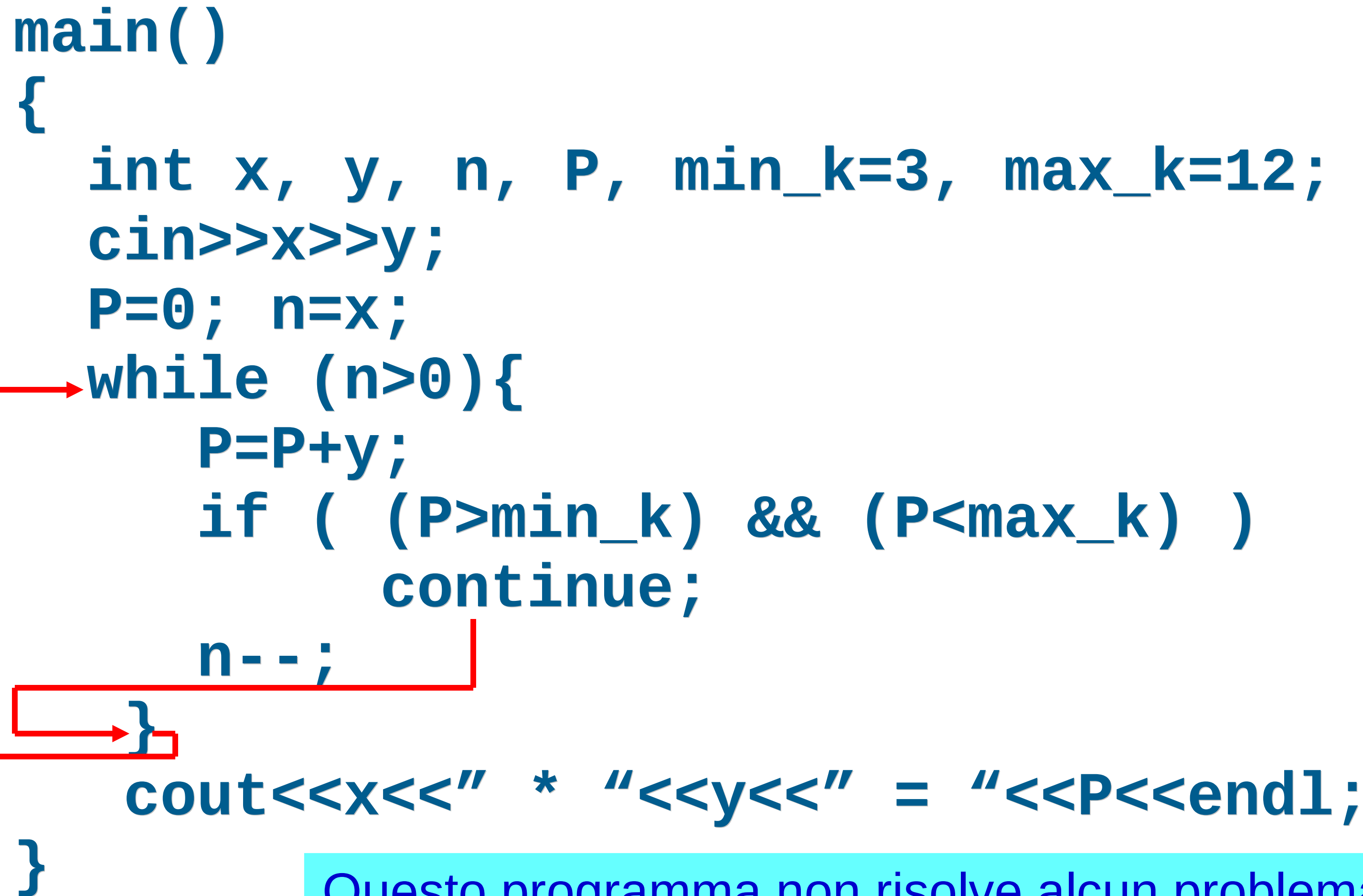
Flessibilità e pericolo

- L'istruzione **`break;`** fornisce quindi una <u>uscita alternativa</u> 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

Esercizi

- Svolgere tutti i rimanenti esercizi della quinta esercitazione fino a *catena_omogenea.cc*
- Prima prova pratica di autovalutazione
  - Svolgere *catena.cc* della quinta esercitazione

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
```