

Sperimentazioni di Fisica I

mod. A – Lezione 10

Istruzioni in C++ (CAP. 5)

*Dipartimento di Fisica e Astronomia “G. Galilei”,
Università degli Studi di Padova*

Istruzioni Semplici e Composte

- Le **istruzioni** in un programma in C++ sono **eseguite sequenzialmente**
- Il linguaggio definisce un **insieme di istruzioni** che permettono di **controllare il flusso** del programma
- La maggior parte delle **istruzioni** termina con un **punto-e-virgola**

```
ival + 7;           // inutile, ma valido  
std::cout << ival;
```

- Un'**istruzione composta** viene definita come **blocco** se è formata una **serie** di istruzioni racchiuse da parentesi graffe

```
while (val <= 10) {  
    sum += val;  
    ++val;  
}
```

È necessario **racchiudere le istruzioni nel while in un blocco** per far sì che **vengano eseguite tutte nel ciclo** ⇒ il corpo del ciclo `while` è una istruzione o più istruzioni racchiuse in un blocco

Visibilità delle Istruzioni

- È possibile definire variabili dentro strutture di controllo `if`, `while`, `for`
- la loro **visibilità è limitata** al ciclo :

```
while (int i = get_num())
    std::cout << i << std::endl;
i = 0; // Errore: i non accessibile fuori dal while

for (int i = 0; i<n; i++)
    std::cout << i << std::endl;
i = 0; // Errore: i non accessibile fuori dal for
```

la stessa cosa si verifica per le **variabili definite in un blocco** :

```
while(std::cin >> s && s != valore_cercato) {
    int a = s;
}
std::cout << a; // Errore: a non accessibile fuori dal blocco
```

se è necessario l'accesso, vanno definite fuori dal blocco

```
auto beg = v.begin();
while (beg != v.end() && *beg >= 0 ) {
    ++beg;
}
```

Lo Statement Condizionale `if`

- Ne esistono due forme

```
if (espressione) statement  
if (espressione) statement_1 else statement_2
```

- E anche uno `statement` condizionale a più vie:

```
if (espressione_1)  
    statement_1  
else if (espressione_2)  
    statement_2  
  
...  
  
else  
    statement_n
```

Utilizzo dell' if

```
// calcolatore.cxx - Esegue semplici calcoli
#include <iostream>
int main( )
{
    using namespace std;
    cout << "Introduci una espressione (numero op numero) :";
    double n1, n2;
    char op;
    cin >> n1 >> op >> n2;
    double val;
    if (op == '+')    val = n1 + n2;
    else if (op == '-') val = n1 - n2;
    else if (op == '/') val = n1 / n2;
    else if (op == '*') val = n1 * n2;
    else {
        cout << "Operatore \"" << op << "\" sconosciuto \n";
        return 1;
    }
    cout << n1 << op << n2 << " = " << val << endl;
    return 0;
}
```

```
$ ./a.out
```

```
Introduci una espressione (numero op numero) : 4 % 2
```

```
Operatore "%" sconosciuto
```

The Dangling Else Problem (I)

Il codice seguente dà luogo ad una difficoltà semantica

```
if ( a > 0 )  
    if ( b == 1 )  
        cout << "***" << endl;  
    else  
        cout << "###" << endl;
```

A quale `if` appartiene lo statement `else` ?

Al secondo ?

The Dangling Else Problem (II)

Potevamo riscriverlo in questo modo

```
if ( a > 0 )  
    if ( b == 1 )  
        cout << "***" << endl;  
else  
    cout << "###" << endl;
```

Adesso sembra che lo statement `else` appartenga al primo `if`

Regola

Ogni `else` statement si concatena allo statement `if` più vicino

The Dangling Else Problem (III)

Per risolvere ogni ambiguità, racchiudere i blocchi tra parentesi ({ })

```
if ( a > 0 ) {  
    if ( b == 1 ) {  
        cout << "***" << endl;  
    } else {  
        cout << "###" << endl;  
    }  
}
```

Appartiene al `if` interno
default, senza { }

```
if ( a > 0 ) {  
    if ( b == 1 ) {  
        cout << "***" << endl;  
    }  
} else {  
    cout << "###" << endl;  
}
```

Appartiene al `if` esterno
sovrascrive la regola

Lo Statement switch

L'istruzione `switch` funziona come un **dispositivo di routing**, indicando al computer quale riga di codice eseguire:

Sintassi :

```
switch ( espressione_intera )
{
    case label_1 : statement(s)
    case label_2 : statement(s)

    ...

    default : statements(s)
}
```

la scelta viene fatta tramite una

`espressione_intera`

anche **le etichette** devono essere **costanti intere** (es `const int`, `const char`)

una volta che il programma **entra in una specifica linea 'case'**, **continua l'esecuzione sequenzialmente**, a meno che non venga diretto **fuori dallo switch** con una istruzione `break`

Utilizzo dello switch

```
#include <iostream>
using namespace std;
int main()
{
    char scelta;
    cout << "\nOpzione: ";
    cin >> scelta;

    while (scelta != 'Q' && scelta != 'q')
    {
        switch (scelta)
        {
            case 'a':
            case 'A': cout << "\nOpzione A"; break;
            case 'b':
            case 'B': cout << "\nOpzione B"; break;
            default : cout << "\nOpzione " << scelta << " non disponibile";
                     cout << "\nA. stampa A";
                     cout << "\nB. stampa B";
                     cout << "\nQ. esce dal programma";

        }
        cout << "\nOpzione: ";
        cin >> scelta;
    }
    return 0;
}
```

```
$ ./a.out
Opzione: a
-> Opzione A
Opzione: b
-> Opzione B
Opzione: c
-> Opzione c non disponibile
A. stampa A
B. stampa B
Q. esce dal programma
->Opzione: q
$
```

Utilizzo dello switch

```
#include <cctype>
```

```
int vocali = 0, consonanti = 0;  
char ch;  
while (cin >> ch)  
{  
    if ( isalpha(ch) ) {  
        switch (ch)  
        {  
            case 'a' :  
            case 'e' :  
            case 'i' :  
            case 'o' :  
            case 'u' :  
                ++vocali;  
                break;  
            default:  
                ++consonanti;  
        }  
    }  
}  
cout << "Vocali: " << vocali << endl;  
cout << "Consonanti: " << consonanti << endl;
```

```
$ ./a.out  
testo di prova  
Vocali: 5  
Consonanti: 7
```

Il Ciclo `for` tradizionale

• Fornisce una ricetta per eseguire azioni ripetute

- Sintassi

```
for (init; test_expr; update_expr)  
    body
```

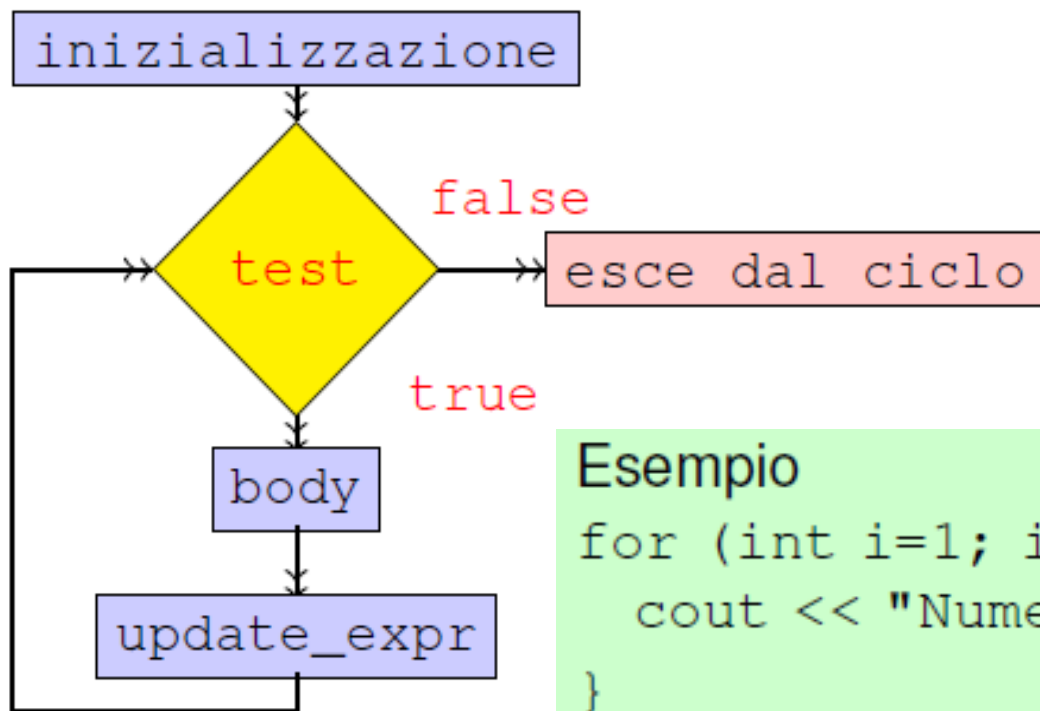
- Consta di una

- **Inizializzazione** eseguita una volta sola per tutto il ciclo
- **Verifica di una condizione** necessaria per ripetere il ciclo (eseguita ogni volta)
- **Esecuzione** delle istruzioni del **corpo del ciclo** (se sono più di una, vanno racchiuse tra parentesi graffe)
- **Aggiornamento** del/dei **valore/i** utilizzati nel test

Il Ciclo for

- Sintassi:

```
for (inizializzazione; test; update_expr)  
    body
```



Esempio

```
for (int i=1; i<=5; i++) {  
    cout << "Numero: " << i << endl;  
}
```

Utilizzo del Ciclo for

```
// forloop.cxx - Uso del ciclo for per contare
#include <iostream>
int main( )
{
    using namespace std;
    cout << "Fino a che numero vuoi contare ?";
    int max;
    cin >> max;

    int i; // Creo un contatore
    for (i=1; i <= max; i++) {
        cout << "Numero" << i << endl;
    }

    cout << "Alla fine del ciclo, i=" << i << endl;

    return 0;
}
```

```
$ ./a.out
Fino a che numero voi contare ? 3
Numero 1
Numero 2
Numero 3
Alla fine del ciclo i = 4
```

Inizializzazione del Ciclo for

- È possibile **dichiarare una variabile** all'interno dello statement di **inizializzazione** del ciclo for

```
int numero = 12;
for (int i = numero - 1; i > 0; --i) {
    numero *= i;
}
cout << "i = " << i;
```

- **Attenzione**, la variabile **i** non esiste più al di fuori del ciclo
- Un compilatore conforme allo standard ISO del C++ produce il seguente messaggio di errore:

```
$ g++ -std=c++11 if_scope.cxx
if_scope.cxx: In function 'int main()':
if_scope.cxx:11:21: error: 'i' was not declared in this scope
```

Cicli `for` e Precisione Finita

Nei cicli `for`, è una buona regola utilizzare espressioni relazionali, quando possibile, invece di espressioni di eguaglianza

Nel caso di variabili di tipo reale (float o double), un test di eguaglianza può andare al di là della precisione della macchina:

```
double sum = 0.0;
for (double x=0.0; x!=9.9; x+=0.1) {
    sum += x;
    cout << sum << x << endl;
}
```

Il ciclo entra in un loop infinito!



Il range for

Le **stringhe**, gli **array** e **tutti i contenitori della libreria standard** del C++ supportano il nuovo costrutto **range for**

```
#include <iostream>
#include <vector>
int main()
{
    std::vector<int> v = {0, 1, 2, 3, 4, 5}

    for (auto i : v) // access by value
        std::cout << i << ' ';
    std::cout << std::endl;

    for (auto & i : v) // access by reference
        std::cout << i << ' ';
    std::cout << std::endl;

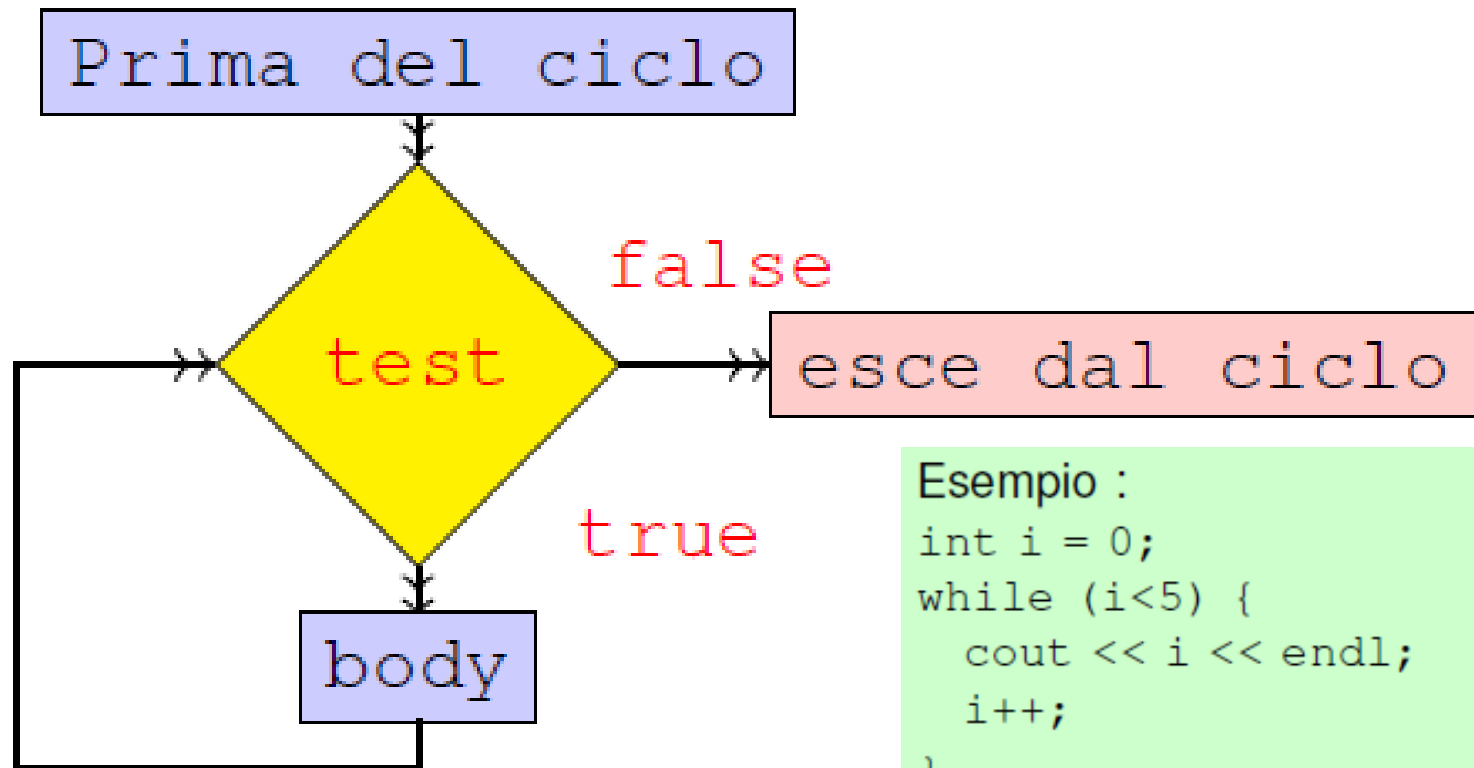
    for(int n : {0,1,2,3,4,5}) // a braced-init-list
        std::cout << n << ' ';
    std::cout << std::endl;

    return 0;
}
```

```
$ ./a.out
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
```

Struttura del Ciclo while

Sintassi: `while (test)`
`body`



Esempio :

```
int i = 0;
while (i<5) {
    cout << i << endl;
    i++;
}
```

Utilizzo del Ciclo while

```
#include <iostream>
int main( )
{
    using namespace std;
    int c, digits=0, letters=0;

    while ( (c = cin.get() ) != EOF) {

        if (c>='0' && c <= '9')
            digits++;

        else if ( (c>='a' && c <= 'z') ||
                  (c>='A' && c <= 'Z') )
            letters++;

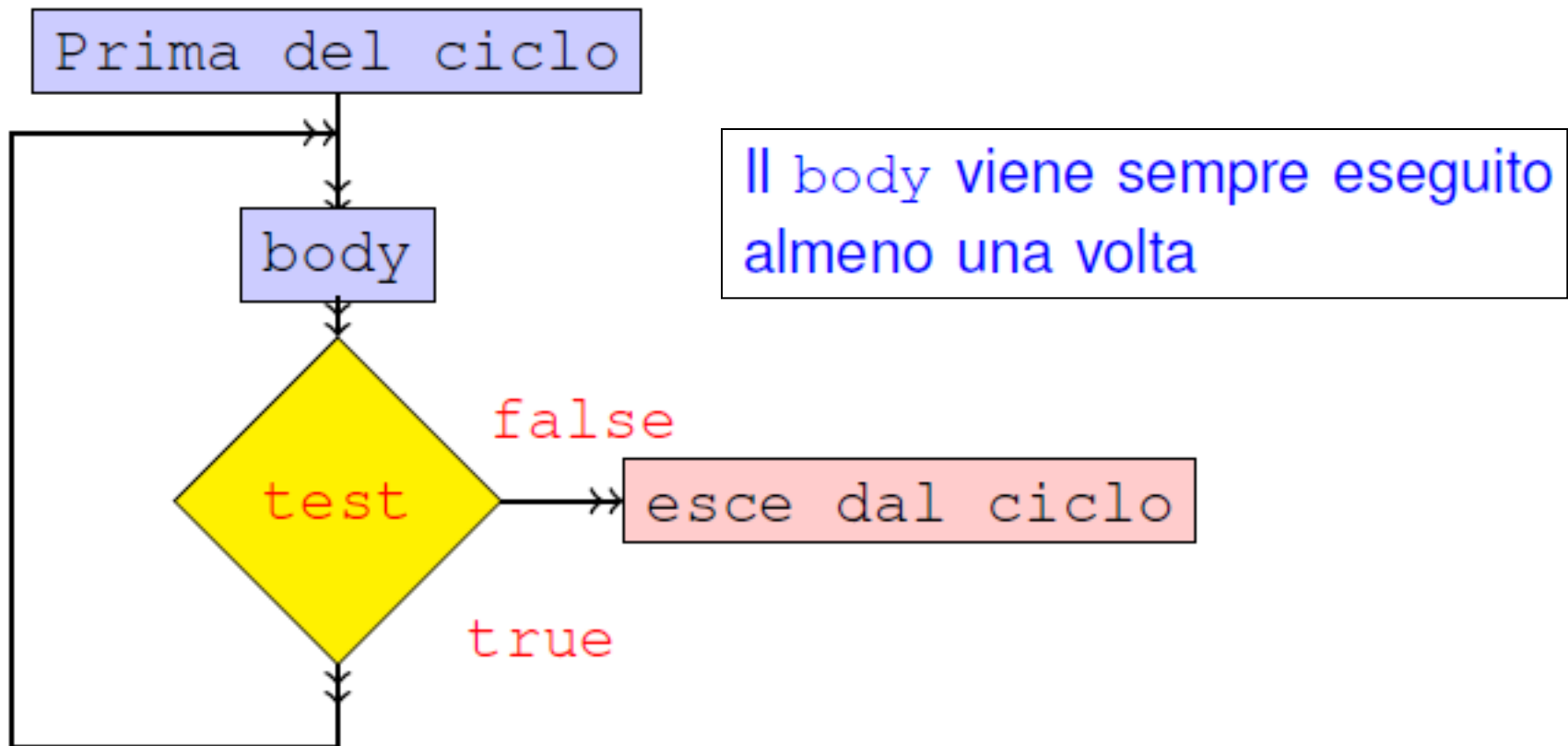
    }
    cout << "Letters :" << letters << endl;
    cout << "Digits  :" << digits << endl;

    return 0;
}
```

```
$ ./a.out
prova due 1243
Letters: 8
Digits : 4
```

Struttura del Ciclo do while

Sintassi: do
 body
 while (test);



Utilizzo del Ciclo do while

```
#include <iostream>
int main( )
{
    using namespace std;

    int i, error;
    do {
        cout << "Inserisci numero positivo: " << endl;
        cin >> i;
        if ( error = (i<=0) )
            cout << "Numero non valido :" << i << endl;

    } while (error);

    cout << "Numero valido: " << i << endl;
    return 0;
}
```

```
$ ./a.out
Inserisci numero positivo: -2
Numero non valido : -2
Inserisci numero positivo: 7
Numero valido : 7
```

break e continue

interrompono e modificano il normale flusso del programma :

➤ break

- 1 termina l'esecuzione del ciclo while, do, for e switch
- 2 trasferisce il controllo allo statement successivo alla fine del ciclo

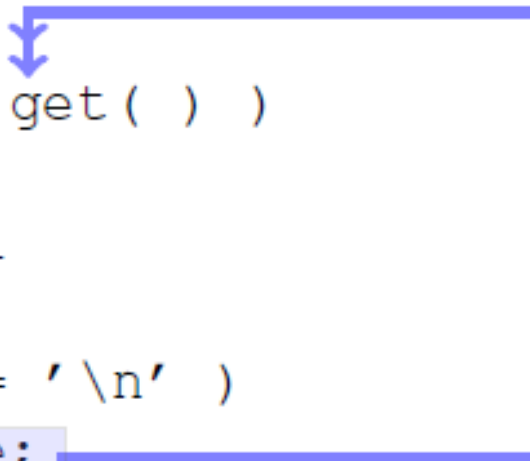
➤ continue

- 1 termina l'esecuzione del corpo del ciclo while, do o for
- 2 trasferisce il controllo alla fine del corpo del ciclo e l'esecuzione continua con la rivalutazione della condizione di test (o con l'espressione di incremento nel caso del for).

L'Istruzione continue

- permette di *saltare* alcune parti di codice.

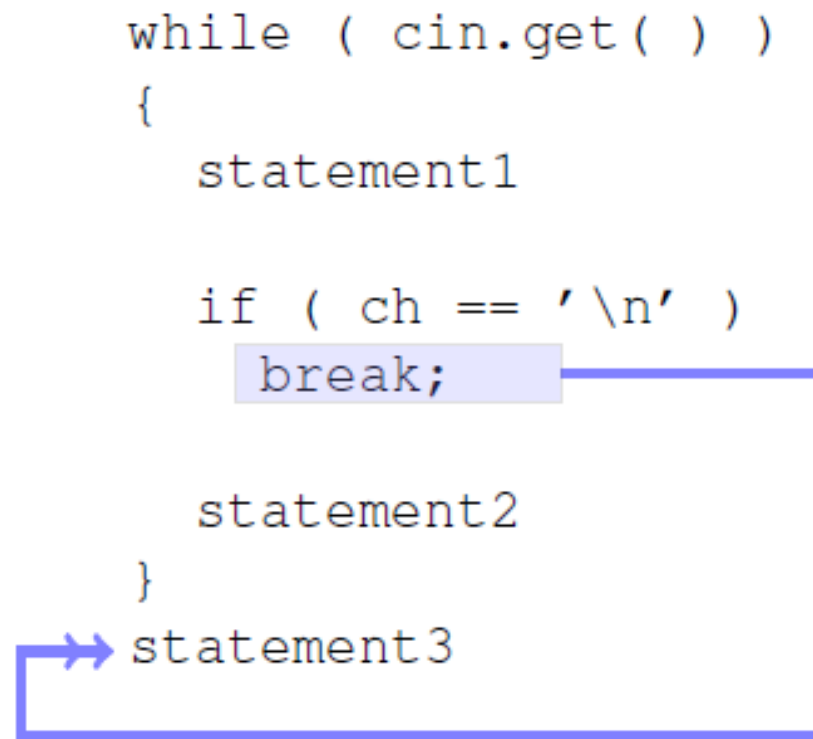
```
while ( cin.get( ) )  
{  
    statement1  
  
    if ( ch == '\n' )  
        continue;  
  
    statement2  
}  
  
statement3
```



- *continue* salta le istruzioni che seguono e fa ripartire un nuovo ciclo

L'Istruzione break

- permette di *uscire dal ciclo più interno*



- break* esce dal ciclo e continua alla prima istruzione successiva