

**C++: cicli for e comandi da non  
utilizzare (*break*, *continue*, *goto*)**  
**Il costrutto *do-while***  
**Il costrutto *switch-case***  
**Curiosità e stranezze del linguaggio**

**Liceo G.B. Brocchi**  
**Classi prime Scientifico - opzione scienze applicate**  
Bassano del Grappa, Marzo 2023

# Il costrutto *for*

```
int i = 1;
int square = 0;
while (i <= 10) {
    square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }

    i++;
}
```

Inizializzazione della variabile di controllo

Test di permanenza

Riassegnazione della variabile di controllo

**Possiamo scrivere la stessa cosa in modo più compatto?**

# Il costrutto *for*

Inizializzazione della variabile di controllo

Test di permanenza

```
int square = 0;
for (int i = 1; i <= 10; i++) {
    square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }
}
```

Riassegnazione della variabile di controllo

**Quando avviene l'incremento di i?**

# Il costrutto *for*

```
int i = 1;
int square = 0;
for (; i <= 10;) {
    int square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }

    i += 1;
}
```

**Sintatticamente è corretto... ma  
non stiamo sfruttando la  
compattezza del costrutto for!**

# Il costrutto *for*

```
int i = 1;
for (; i <= 10;) {
    int square = i * i;
    cout << square;
    if (i == 10) {
        cout << '\n';
    }
    else {
        cout << '\t';
    }

    i += 1;
}
```

**Sintatticamente è corretto... ma  
non stiamo sfruttando la  
compattezza del costrutto for!**

# Il costrutto *for*

*Scrivere un programma che stampi su stdout un quadrato di asterischi di dimensione  $n$  utilizzando un ciclo *for*.*

```
int n = 5;
for (int i = 1; i <= n*n; i++) {
    cout << '*';
    if (i % n == 0) {
        cout << '\n';
    }
}
```

**Tradurre il codice in  
diagramma di flusso**

# Il costrutto *for*

*Scrivere un programma che stampi su stdout un quadrato di asterischi di dimensione  $n$  utilizzando 2 cicli for annidati (i.e. uno dentro l'altro).*

```
int n = 5;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        cout << '*';
    }
    cout << '\n';
}
```

# Srotolamento al quadrato

Iterazione **esterna** 1) i: 1

Iterazione **interna** 1) j: 1;  
Iterazione **interna** 2) j: 2;  
Iterazione **interna** 3) j: 3;  
Iterazione **interna** 4) j: 4;  
Iterazione **interna** 5) j: 5;

Iterazione **esterna** 2) i: 2

Iterazione **interna** 1) j: 1;  
Iterazione **interna** 2) j: 2;  
Iterazione **interna** 3) j: 3;  
Iterazione **interna** 4) j: 4;  
Iterazione **interna** 5) j: 5;



# Srotolamento al quadrato

Iterazione **esterna** 3) i: 3

Iterazione **interna** 1) j: 1;  
Iterazione **interna** 2) j: 2;  
Iterazione **interna** 3) j: 3;  
Iterazione **interna** 4) j: 4;  
Iterazione **interna** 5) j: 5;

Iterazione **esterna** 4) i: 4

Iterazione **interna** 1) j: 1;  
Iterazione **interna** 2) j: 2;  
Iterazione **interna** 3) j: 3;  
Iterazione **interna** 4) j: 4;  
Iterazione **interna** 5) j: 5;

# Srotolamento al quadrato

Iterazione **esterna** 5) i: 5

Iterazione **interna** 1) j: 1;

Iterazione **interna** 2) j: 2;

Iterazione **interna** 3) j: 3;

Iterazione **interna** 4) j: 4;

Iterazione **interna** 5) j: 5;

# Il costrutto *do-while*

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

**Non viene eseguita alcuna iterazione se  
decimal\_number == 0.  
Ma l'utente potrebbe voler convertire  
anche 0!**

```
while (decimal_number != 0) {  
    binary_digit = decimal_number % 2;  
    cout << remainder;  
    decimal_number /= 2;  
}
```

# Il costrutto *do-while*

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

```
if (decimal_number == 0) {  
    cout << 0 << endl;  
}
```

```
while (decimal_number != 0) {  
    binary_digit = decimal_number % 2;  
    cout << binary_digit;  
    decimal_number /= 2;  
}
```

Aggiungiamo un controllo per il caso  
limite `decimal_number == 0`

# Il costrutto *do-while*

```
int decimal_number;  
cin >> decimal_number;  
int binary_digit;
```

```
do {  
    binary_digit = decimal_number % 2;  
    cout << binary_digit;  
    decimal_number /= 2;  
} while (decimal_number != 0);
```

Il corpo del ciclo viene eseguito almeno una volta perché il controllo di permanenza è posto dopo il corpo! Quindi la conversione sarà eseguita anche con `decimal_number == 0`.

Corpo del ciclo

Test di permanenza

# Il costrutto *do-while*

```
int previous_input;  
int current_input;  
cin >> current_input;  
  
do {  
    previous_input = current_input;  
    cin >> current_input;  
} while (current_input != -1 && current_input >= previous_input);
```

# Il costrutto *switch-case*

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
switch (grade_c) {  
  case 'A':  
    grade_i = 10;  
  break;  
  case 'B':  
    grade_i = 8;  
  break;  
  case 'C':  
    grade_i = 7;  
  break;  
  case 'D':  
    grade_i = 6;  
  break;  
  default:  
    grade_i = -1;  
  break;  
}  
cout << "numeric grade is" << grade_i << endl;
```


I vari casi sono identificati da costanti intere (qui sono char, che comunque sono interpretabili come interi)

Viene eseguito solo se non si verifica nessuno dei casi identificati dai **case**

# Il costrutto *switch-case*

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
char x = 'A';  
switch (grade_c) {  
    case x:←  
        grade_i = 10;  
        break;  
    case 'B':  
        grade_i = 8;  
        break;  
    case 'C':  
        grade_i = 7;  
        break;  
    case 'D':  
        grade_i = 6;  
        break;  
    default:  
        grade_i = -1;  
        break;  
}
```

x è una variabile, non una costante.  
Il compilatore dà errore.



```
ecture2.cpp  
ecture2.cpp(19): error C2051: espressione case non costante
```



# Il costrutto *switch-case* – l'importanza del comando *break*

```
char grade_c;  
cin >> grade_c;  
int grade_i;  
char x = 'A';  
switch (grade_c) {  
    case 'A':  
        grade_i = 10;  
    case 'B':  
        grade_i = 8;  
    break;  
    case 'C':  
        grade_i = 7;  
    break;  
    case 'D':  
        grade_i = 6;  
    break;  
    default:  
        grade_i = -1;  
    break;  
}
```

manca il comando **break**  
Ecco cosa succede:

```
enter character grade: A  
grade converted to number is: 8
```

Il controllo è passato al case successivo,  
che in questo caso è **case 'B'**


```
cout << "grade converted to number is: " << grade_i << endl;
```

# Il costrutto *switch-case* – equivalenza con *if – else if - else*

```
char grade_c;  
cout << "enter character grade: ";  
cin >> grade_c;  
int grade_i;  
  
if (grade_c == 'A') {  
    grade_i = 10;  
}  
else if (grade_c == 'B') {  
    grade_i = 8;  
}  
else if (grade_c == 'C') {  
    grade_i = 7;  
}  
else if (grade_c == 'D') {  
    grade_i = 6;  
}  
else {  
    grade_i = -1;  
}  
  
cout << grade_i;
```

# L'istruzione *break*

```
int in = 0;  
while (in != 5) {  
    cin >> in;  
    cout << in << endl;  
    cout << "hello world" << endl;  
}
```

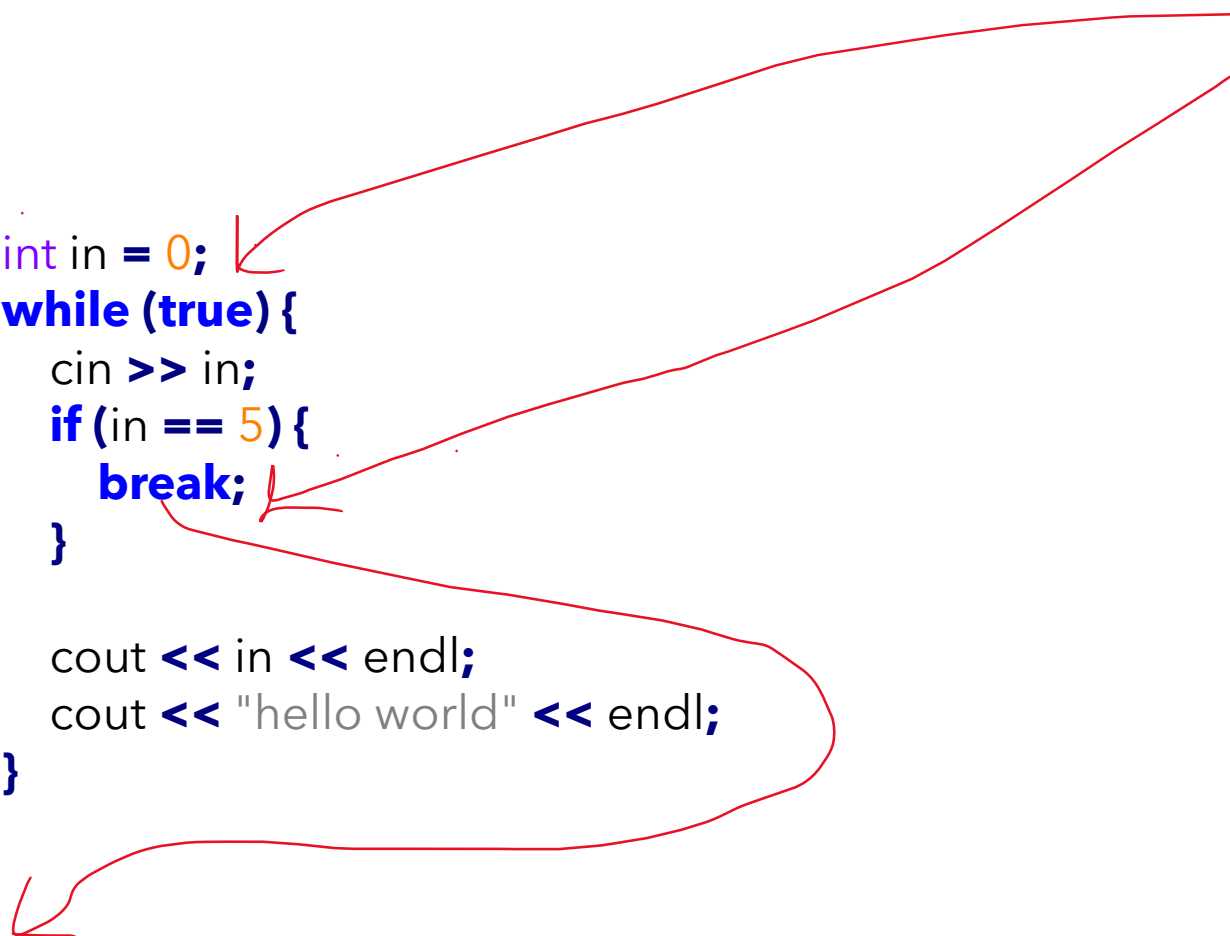


La condizione di uscita dal ciclo è in un unico punto ben definito. Non dobbiamo cercare da nessun'altra parte per capire quando il controllo passerà all'istruzione successiva al ciclo

# L'istruzione *break*

```
int in = 0;
while (true) {
    cin >> in;
    if (in == 5) {
        break;
    }

    cout << in << endl;
    cout << "hello world" << endl;
}
```



**salta qui**

La condizione di permanenza indica che il ciclo è infinito. Ma nel corpo effettivamente si esce con il comando **break**.


Poco leggibile e causa di confusione.

**Contro i principi della programmazione strutturata.**

# L'istruzione *continue*

```
for (int i = 1; i <= 10; i++) {  
    if (i == 5) {  
        continue;  
    }  
    cout << i << " ";  
}
```

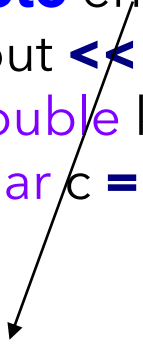
Commenti sull'output?






1 2 3 4 6 7 8 9 10

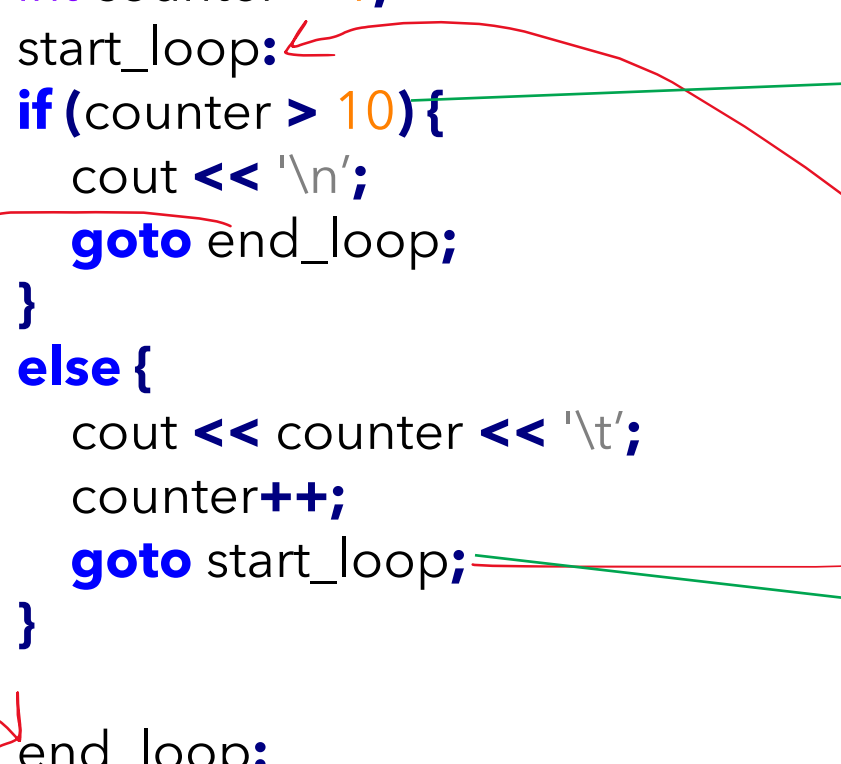
# L'istruzione *goto*

```
int i = 0;  
int j = 0;  
goto end;  
cout << "hello world";  
double k = 1.0;  
char c = 'c';  
  
end:  
    cout << "jumped to label \"end\"";
```

A black arrow originates from the text 'goto end;' and points diagonally down and to the left to the label 'end:'.

# L'istruzione *goto*

```
int counter = 1;  
start_loop:   
if (counter > 10) {  
    cout << '\n';  
    goto end_loop;  
}  
else {  
    cout << counter << '\t';  
    counter++;  
    goto start_loop;   
}  
end_loop:   
    cout << "out of goto-like loop";
```



Condizione di uscita dal  
«ciclo»

Jump al controllo di  
permanenza nel «ciclo»

# L'istruzione goto

```
int counter = 1;
start_loop:
if (counter > 10) {
    cout << '\n';
    goto end_loop;
}
else {
    cout << counter << '\t';
    counter++;
    goto start_loop;
}

end_loop:
;
```

## Traduzione in linguaggio Assembly x86

```
mov ax, 1;
start_loop:
    cmp ax, 10;
    jg end_loop;
    inc ax;
    jmp start_loop;
end_loop:
    nop;
```

Utilizzare il goto porta alla scrittura di codice a «basso livello», poco leggibile e poco manutenibile (ma magari efficiente)  
(<https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf>)



# L'istruzione *goto*

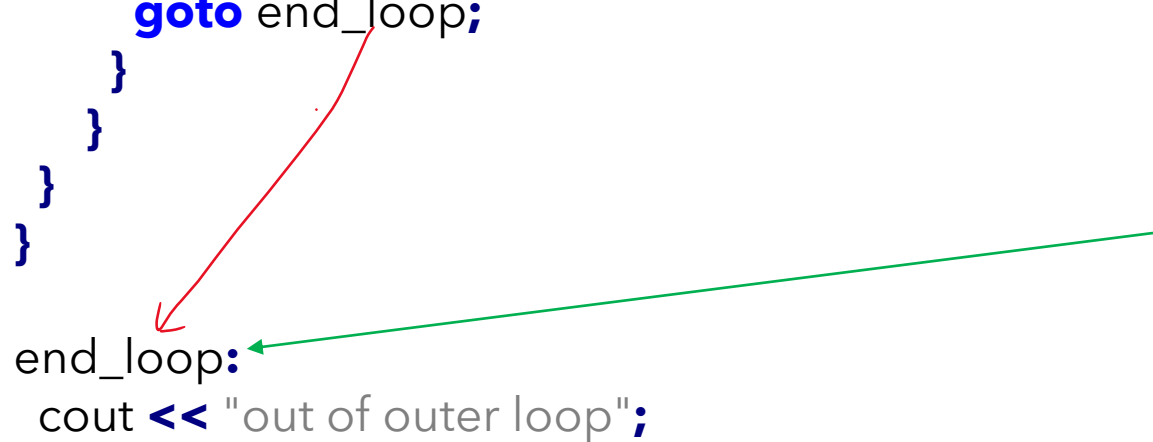
```
char in = 'z';  
for (int i = 0; i < 3; i++) {  
    cout << "outer loop index is: " << i << endl;  
    for (int j = 0; j < 3; j++) {  
        cout << "***first inner loop index is: " << j << endl;  
        for (int k = 0; k < 3 && in != 'e'; k++) {  
            cout << "***second inner loop index is: "  
                << k << endl;  
            cout << "***enter a character: ";  
            cin >> in;  
        }  
    }  
}
```

```
*outer loop index is: 0  
**first inner loop index is: 0  
***second inner loop index is: 0  
***enter a character: e  
**first inner loop index is: 1  
**first inner loop index is: 2  
*outer loop index is: 1  
**first inner loop index is: 0  
**first inner loop index is: 1  
**first inner loop index is: 2  
*outer loop index is: 2  
**first inner loop index is: 0  
**first inner loop index is: 1  
**first inner loop index is: 2
```

# L'istruzione *goto*

```
char in = 'z';
for (int i = 0; i < 3; i++){
    cout << "*outer loop index is: " << i << endl;
    for (int j = 0; j < 3; j++){
        cout << "**first inner loop index is: " << j << endl;
        for (int k = 0; k < 3; k++){
            cout << "***second inner loop index is: " << k << endl;
            cout << "***enter a character: ";
            cin >> in;
            if (in == 'e'){
                goto end_loop;
            }
        }
    }
}

end_loop:
cout << "out of outer loop";
```



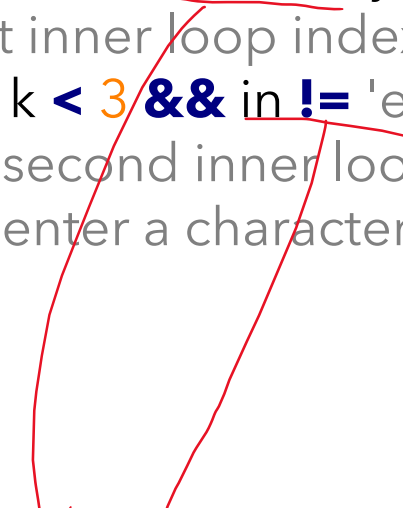
```
*outer loop index is: 0
**first inner loop index is: 0
***second inner loop index is: 0
***enter a character: e
out of outer loop
```

salto diretto alla label **end\_loop**.

Non vengono più testate le condizioni di permanenza nei 3 cicli.

# L'istruzione *goto*

```
char in = 'z';
for (int i = 0; i < 3; i++) {
    cout << "*outer loop index is: " << i << endl;
    for (int j = 0; j < 3 && in != 'e'; j++) {
        cout << "**first inner loop index is: " << j << endl;
        for (int k = 0; k < 3 && in != 'e'; k++) {
            cout << "***second inner loop index is: " << k << endl;
            cout << "***enter a character: ";
            cin >> in;
        }
    }
}
```

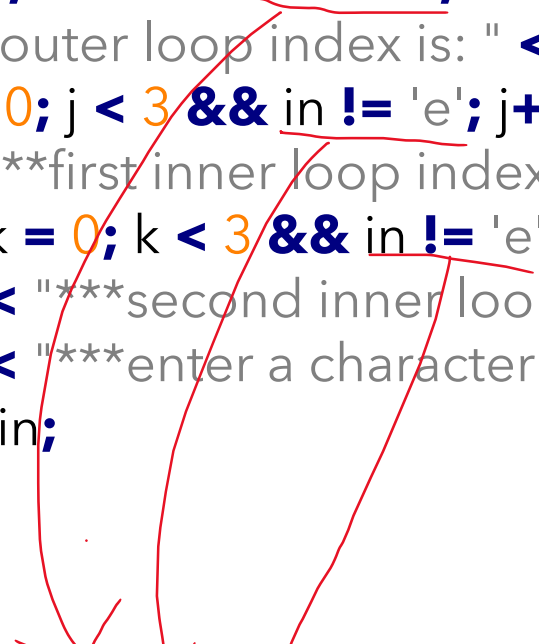


**controllo ripetuto 2 volte**

```
*outer loop index is: 0
**first inner loop index is: 0
...
***second inner loop index is: 0
***enter a character: e
*outer loop index is: 1
...
*outer loop index is: 2
```

# L'istruzione *goto*

```
char in = 'z';
for (int i = 0; i < 3 && in != 'e'; i++) {
    cout << "*outer loop index is: " << i << endl;
    for (int j = 0; j < 3 && in != 'e'; j++) {
        cout << "**first inner loop index is: " << j << endl;
        for (int k = 0; k < 3 && in != 'e'; k++) {
            cout << "***second inner loop index is: " << k << endl;
            cout << "***enter a character: ";
            cin >> in;
        }
    }
}
```



**controllo ripetuto 3 volte**

il risultato è uguale alla versione con il **goto**. Ma prima di uscire dai 3 cicli qui vengono effettuati tutti i test di permanenza. Non si esce bruscamente dai 3 cicli annidati con un unico salto.

```
*outer loop index is: 0
**first inner loop index is: 0
***second inner loop index is: 0
...***enter a character: e
```

# Curiosità e stranezze

Volete accedere ad una cassaforte controllata elettronicamente. Solo chi è in possesso del codice 1324 ha il permesso di aprirla. Il computer interno alla cassaforte controlla il pin di accesso con queste righe di codice:

```
int secret;  
cin >> secret;  
  
if (secret = 1324) {  
    cout << "ACCESS TO SAFE GRANTED" << endl;  
}  
else {  
    cout << "WRONG CODE. ACCESS NOT GRANTED" << endl;  
}
```

# Curiosità e stranezze

Volete accedere ad una cassaforte controllata elettronicamente. Solo chi è in possesso del codice 1324 ha il permesso di aprirla. Il computer interno alla cassaforte controlla il pin di accesso con queste righe di codice:

```
int secret;  
cin >> secret;  
  
if (secret = 1324) {  
    cout << "ACCESS TO SAFE GRANTED" << endl;  
}  
else {  
    cout << "WRONG CODE. ACCESS NOT GRANTED" << endl;  
}
```

Operatore di assegnazione!!!  
Restituisce il valore assegnato alla variabile,  
che è positivo.  
Un valore positivo nel test di un **if** viene  
convertito nel valore booleano **true**

# Curiosità e stranezze

Ecco cosa succede quando inserite un pin:

```
123234  
ACCESS TO SAFE GRANTED  
C:\Users\user>lecture1.exe  
6754567  
ACCESS TO SAFE GRANTED  
C:\Users\user>lecture1.exe  
34534352  
ACCESS TO SAFE GRANTED
```