



## COSTRUTTI DI PROGRAMMAZIONE STRUTTURATA

---

Roberto Nardone, Luigi Romano

- ❑ Costrutti di Base
- ❑ Sequenze (o blocchi) di Istruzioni
- ❑ Costrutto condizionale (o di selezione) **if** e **if-else**
- ❑ Costrutto condizionale (o di selezione) **switch**
- ❑ Il ciclo **while** e il **do-while**
- ❑ Il ciclo **for**
- ❑ Esempio finale

- Un algoritmo strutturato è scomponibile in blocchi di istruzioni
  - Annidati o in sequenza
- Il linguaggio C++ permette di evidenziare **blocchi di istruzioni** tramite parentesi graffe '{' e '}'
- Fornisce 3 **costrutti base**:
  - Sequenze di istruzioni {...}
  - Selezioni (if (...) { ... } else { ... })
  - Cicli (while (...) { ... })

- Un esempio di blocco di istruzioni lo abbiamo visto la scorsa lezione:

```
#include <iostream>
```

```
int main() {  
    int prima_variable;  
    prima_variabibile=5;  
}
```

- Un blocco di istruzioni è una sequenze di istruzioni racchiuse tra {}
- Può contenere definizioni di variabili e sequenze di istruzioni
  - Istruzioni: assegnamenti, invocazioni di funzione
  - Ricordare che le istruzioni terminano con “;”

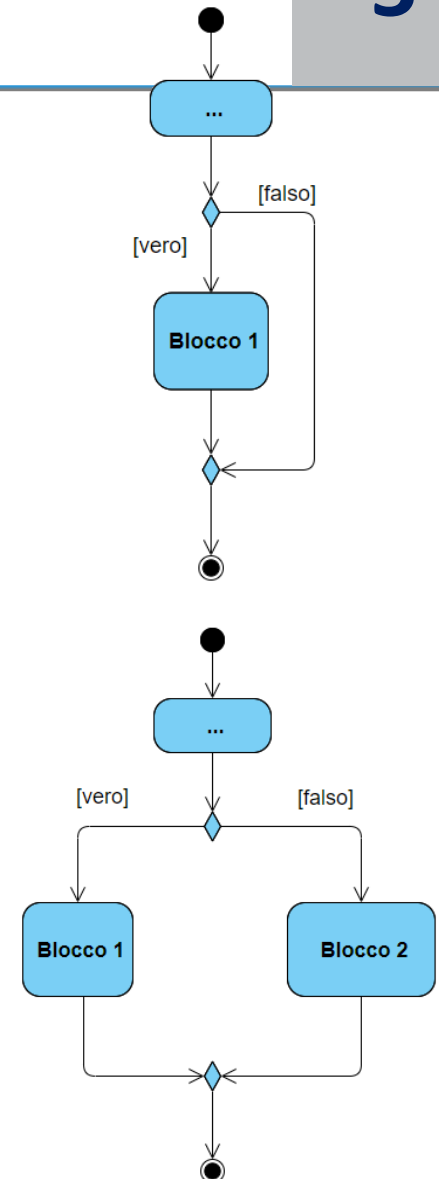
- Il costrutto “*se predicato allora fai istruzioni*” si mappa in C++ come segue:

```
if (predicato) {  
    //Blocco istruzioni  
}
```

- A tempo di esecuzione, il calcolatore valuterà il predicato. Se verificato allora eseguirà il blocco di codice

- Il costrutto “*se predicato allora fai Blocco1 altrimenti Blocco2*” si mappa in C++ come segue:

```
if (predicato) {  
    //Blocco istruzioni 1  
} else {  
    //Blocco istruzioni 2  
}
```



- Il costrutto “*se predicato allora fai istruzioni*” si mappa in C++ come segue:

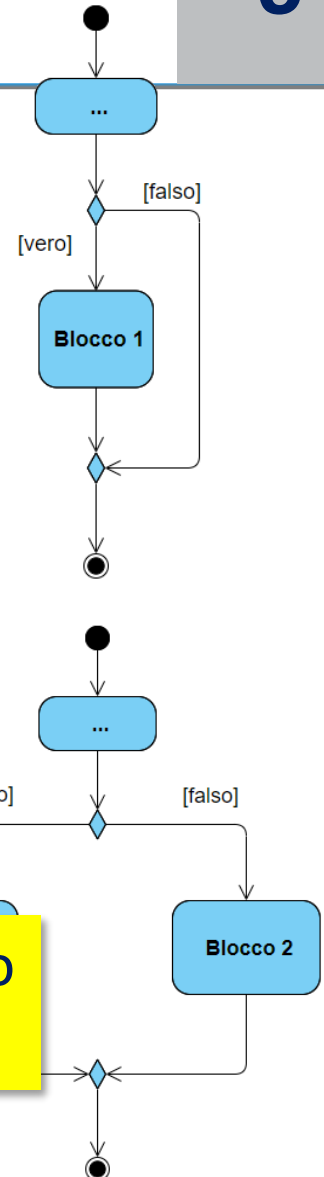
```
if (predicato) {  
    //Blocco istruzioni  
}
```

- A tempo di esecuzione, il calcolatore valuterà il predicato. Se verificato allora eseguirà il blocco di codice

- Il costrutto “*se predicato allora fai Blocco1 altrimenti Blocco2*” si mappa in C++ come segue:

```
if (predicato) {  
    //Blocco istruzioni 1  
} else {  
    //Blocco istruzioni 2  
}
```

**Notare che if e else sono  
Parole chiave!**



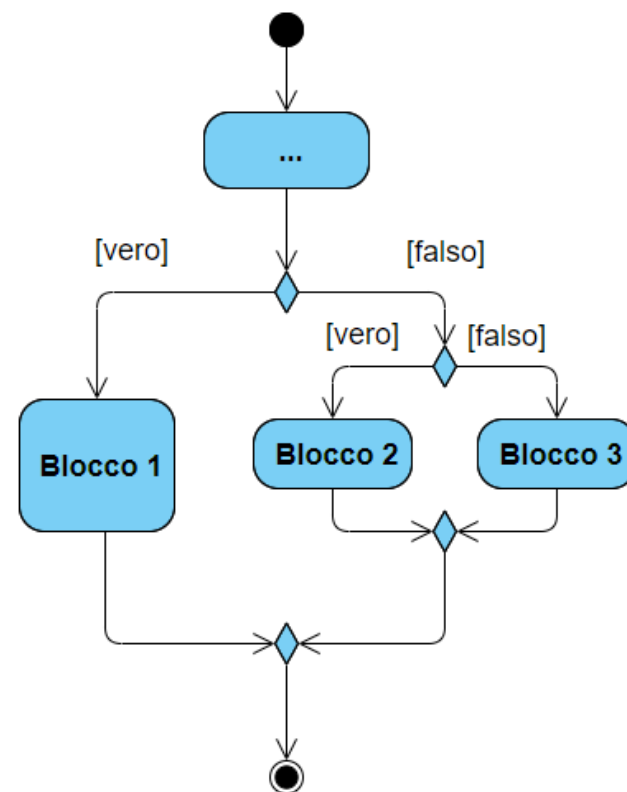
- Il predicato consiste in questo caso nel verificare se la variabile è maggiore di 5, e nell'altro caso se è maggiore o uguale di 5

```
#include <iostream>
using namespace std;
int main() {
    int var=5;
    if (var>5) {
        cout<<"Dentro ramo If"<<endl;
    }
}
```

```
#include <iostream>
using namespace std;
int main() {
    int var=5;
    if (var>=5) {
        cout<<"Dentro ramo If"<<endl;
    } else {
        cout<<"Dentro ramo else"<<endl;
    }
}
```

- Il predicato consiste in questo caso nel verificare se la variabile è maggiore di 5, e nell'altro caso se è maggiore o uguale di 5

```
#include <iostream>
using namespace std;
int main() {
    int var=5;
    if (var>0) {
1      cout<<"Numero Positivo"<<endl;
    } else {
2      if (var==0) {
          cout<<"Numero Nullo"<<endl;
3      } else {
          cout<<"Numero Negativo"<<endl;
      }
    }
}
```





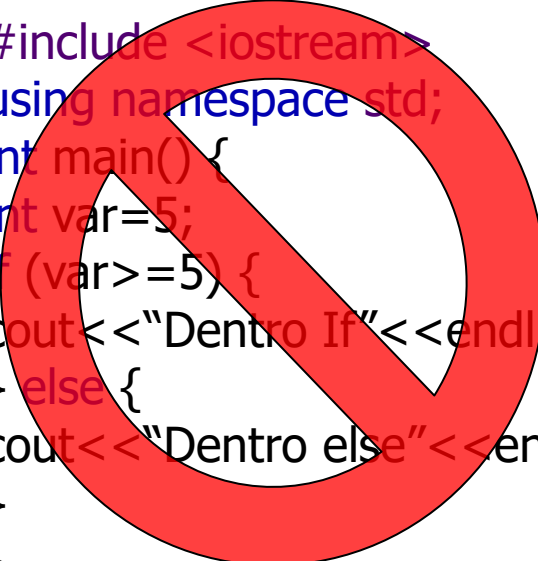
- **L'indentazione** è fondamentale affinché un programma sia **leggibile**. Ogni volta che si apre un nuovo blocco di codice {} è importante spaziare in avanti le istruzioni.
- Quale tra i due è più **leggibile**?

```
#include <iostream>
using namespace std;
int main() {
int var=5;
if (var>=5) {
cout<<"Dentro If"<<endl;
} else {
cout<<"Dentro else"<<endl;
}
}
```

```
#include <iostream>
using namespace std;
int main() {
    int var=5;
    if (var>=5) {
        cout<<"Dentro If"<<endl;
    } else {
        cout<<"Dentro else"<<endl;
    }
}
```

- **L'indentazione** è fondamentale affinché un programma sia **leggibile**. Ogni volta che si apre un nuovo blocco di codice {} è importante spaziare in avanti le istruzioni.
- Quale tra i due è più **leggibile**?

Assolutamente NO!



```
#include <iostream>
using namespace std;
int main() {
int var=5;
if (var>=5) {
cout<<"Dentro If"<<endl;
} else {
cout<<"Dentro else"<<endl;
}
}
```

```
#include <iostream>
using namespace std;
int main() {
    int var=5;
    if (var>=5) {
        cout<<"Dentro If"<<endl;
    } else {
        cout<<"Dentro else"<<endl;
    }
}
```

- **L'indentazione** è fondamentale affinché un programma sia **leggibile**. Ogni volta che si apre un nuovo blocco di codice {} è importante spaziare in avanti le istruzioni.
- Quale tra i due è più **leggibile**?

**All'esame non si  
accetteranno prove NON INDENTATE!!!**

- IF può essere seguito da uno o più ELSE IF per specificare condizioni alternative
- Un solo ELSE finale può essere espresso a chiusura della sequenza degli ELSE IF
- Un solo ramo sarà esplorato
  - L'intera sequenza viene interrotta dopo che una delle condizioni ELSE IF è verificata
- Es. 

```
if (temperatura < 35) {  
    cout << "sei freddissimo!!!" << endl;  
} else if (temperatura > 39) {  
    cout << "hai una febbre da cavallo" << endl;  
} else if (temperatura > 37) {  
    cout << "riguardati, hai la febbre" << endl;  
} else {  
    cout << "sei in forma!" << endl;  
}
```

- IF può essere seguito da uno o più ELSE IF per specificare condizioni alternative
- Un solo ELSE finale può essere espresso a chiusura della sequenza degli ELSE IF
- Un solo ramo sarà esplorato
  - L'intera sequenza viene interrotta dopo che la condizione è stata verificata
- Es. 

```
if (temperatura < 35) {  
    cout << "sei freddissimo!!!" << endl;  
} else if (temperatura > 39) {  
    cout << "hai una febbre da cavallo" << endl;  
} else if (temperatura > 37) {  
    cout << "riguardati, hai la febbre" << endl;  
} else {  
    cout << "sei in forma!" << endl;  
}
```

Attenzione alla sequenza dei valori!!!

mettendo prima:

`temperatura > 37`

anche 39 era vero il che avrebbe impedito che venisse valutato il ramo successivo (e corretto)

`temperatura > 39`

- IF può essere seguito da uno o più ELSE IF per specificare condizioni alternative
- Un solo ELSE finale può essere espresso a chiusura della sequenza degli ELSE IF
- Un solo ramo sarà esplorato

- L'intera sequenza viene interrotta o verificata

- Es. 

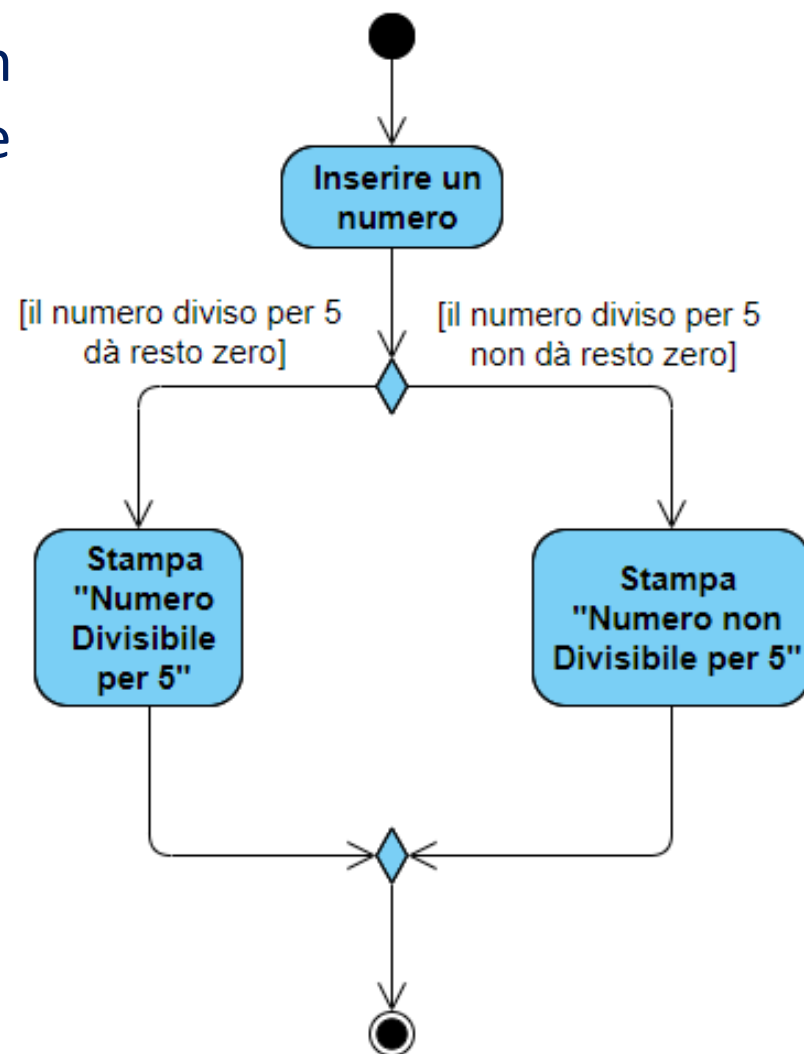
```
if (temperatura < 35) {  
    cout << "sei freddissimo!!!" << endl;  
}  
else if (temperatura > 39) {  
    cout << "hai una febbre da ca" << endl;  
}  
else if (temperatura > 37) {  
    cout << "riguardati, hai la feb" << endl;  
}  
else {  
    cout << "sei in forma!" << endl;  
}
```

In alternativa potevamo mettere:

- `temperatura > 37 && temperatura <= 39`
- `temperatura >= 35 && temperatura <= 37`
- `temperatura > 39`
- `temperatura < 35`

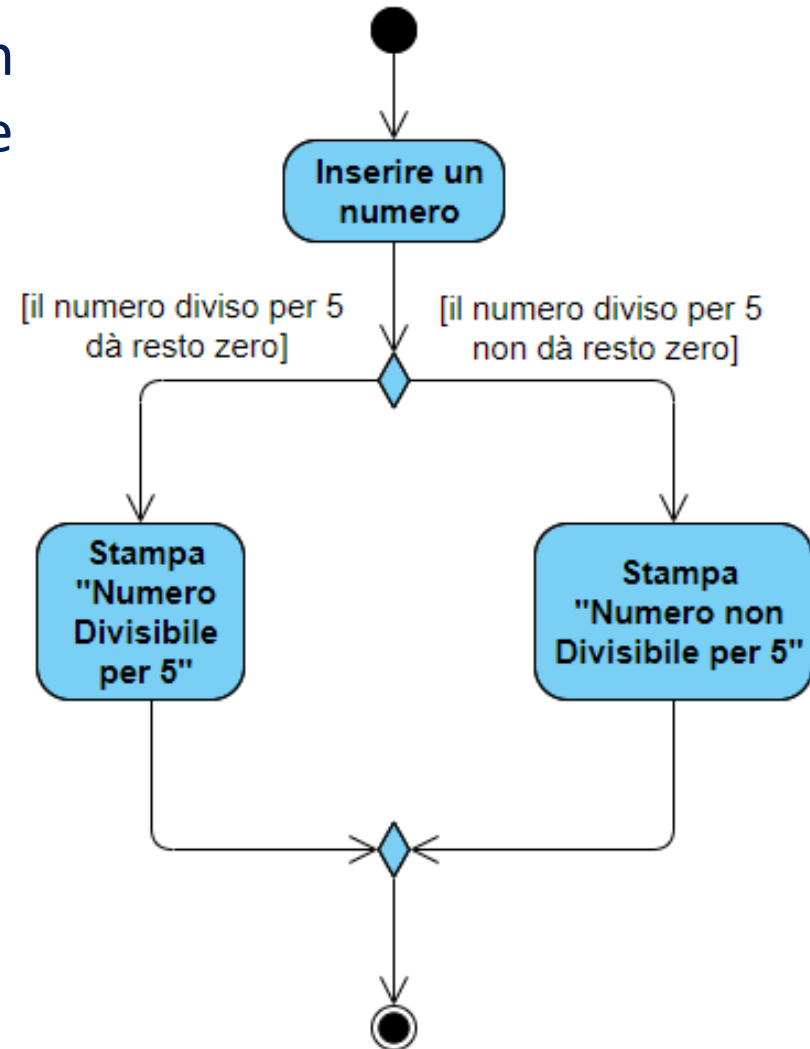
cioè creare condizioni che partizionano l'insieme delle possibilità (una sola può essere vera)

- Scrivere un programma che riceva in input un valore intero. Se tale valore è divisibile per 5, allora il programma stampa a video il messaggio “Il numero inserito è divisibile per 5”, altrimenti il messaggio “Il numero inserito non è divisibile per 5”.



- Scrivere un programma che riceva in input un valore intero. Se tale valore è divisibile per 5, allora il programma stampa a video il messaggio “Numero divisibile per 5”, altrimenti il messaggio “Numero non divisibile per 5”.

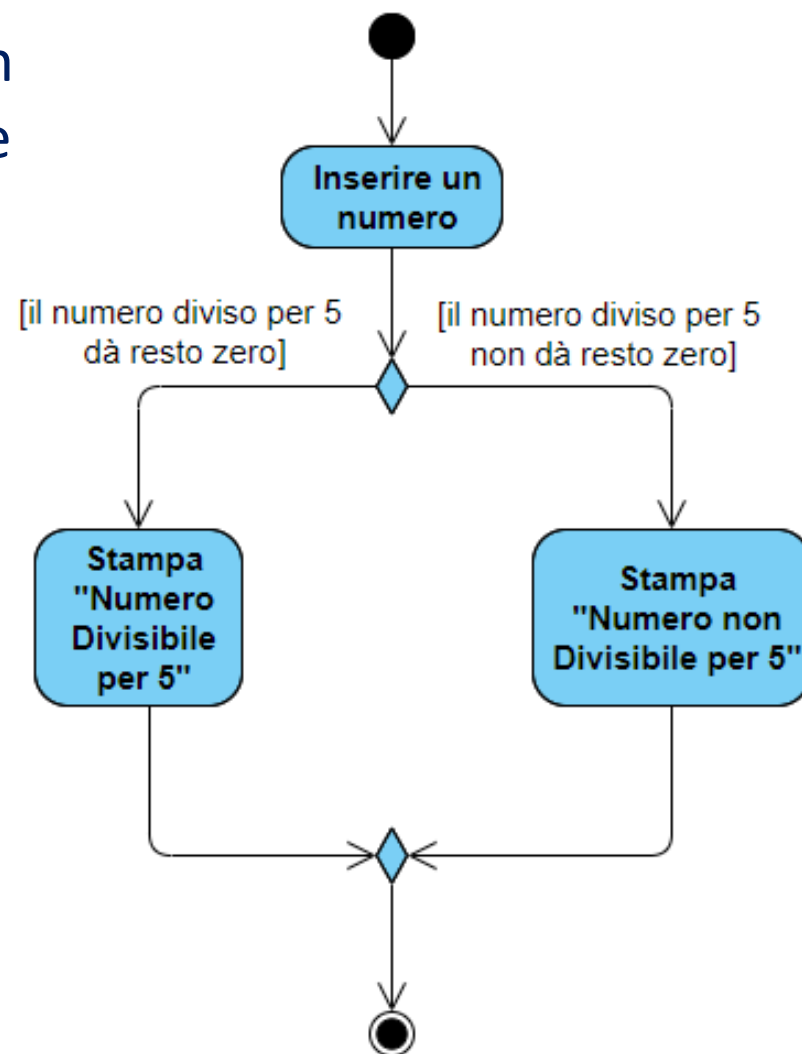
```
if (N%5 == 0) {  
    //numero divisibile per 5  
} else {  
    //numero non divisibile per 5  
}
```





- Scrivere un programma che riceve in input un valore intero. Se tale valore è divisibile per 5 stampi a video un messaggio, altrimenti un altro messaggio.

- esercizio\_05\_01.cpp



- ☐ Si vuole realizzare un programma che dato un anno verifichi se esso è bisestile
- ☐ Un anno è bisestile se è divisibile per 4 ma non per 100, oppure se è divisibile per 400.

Casi di test:

- ☐ Sono bisestili: 1600; 1604; 1608
- ☐ Non sono bisestili: 1603; 1700; 1500

```
/*
  File: Bisestile.cpp
*/

#include <iostream>

using namespace std;

int main(){

    int anno;
    |
    cout << "Inserire anno: ";
    cin >> anno;

    if ( !(anno%4) && (anno%100) || !(anno%400) ) {
        cout << "Bisestile" << endl;
    } else {
        cout << "Non bisestile" << endl;
    }
    return 0;
}
```

- <https://www.hackerrank.com/challenges/c-tutorial-conditional-if-else/problem>

- Il costrutto switch-case serve a eseguire codice diverso a fronte di valori diversi assunti da una espressione.
- Ad esempio risulta utile quando dobbiamo impostare il comportamento del programma in seguito alla scelta di un'opzione da parte di un utente.
  1. valuta il valore dell'*espressione intera* passata come parametro all'istruzione **switch**;
  2. rimanda lo svolgimento del programma al blocco in cui il parametro dell'istruzione case ha lo stesso valore di quello dell'istruzione switch;

```
switch(<espressione intera>)  
{  
    case (<valore costante 1>):  
        <sequenza di istruzioni 1>  
        break;  
    case (<valore costante 2>):  
        <sequenza di istruzioni 2>  
        break;  
    ...  
    case (<valore costante N>):  
        <sequenza di istruzioni N>  
        break;  
    default:  
        <sequenza di istruzioni N+1>  
}
```

3. se il blocco individuato termina con un'istruzione **break** allora il programma esce dallo switch. Altrimenti, vengono eseguiti anche i blocchi successivi finchè un'istruzione **break** non viene individuata oppure non si raggiunge l'ultimo blocco dello switch.
4. se nessun blocco corrisponde ad un valore uguale a quello dell'istruzione switch allora viene eseguito il blocco **default**, se presente.

```
switch(<espressione intera>)  
{  
    case (<valore costante 1>):  
        <sequenza di istruzioni 1>  
        break;  
    case (<valore costante 2>):  
        <sequenza di istruzioni 2>  
        break;  
    ...  
    case (<valore costante N>):  
        <sequenza di istruzioni N>  
        break;  
    default:  
        <sequenza di istruzioni N+1>  
}
```

```
1  #include <iostream>
2  using namespace std;
3
4  int main () {
5      // local variable declaration:
6      char grade = 'D';
7
8      switch(grade) {
9          case 'A' :
10             cout << "Excellent!" << endl;
11             break;
12          case 'B' :
13          case 'C' :
14             cout << "Well done" << endl;
15             break;
16          case 'D' :
17             cout << "You passed" << endl;
18             break;
19          case 'F' :
20             cout << "Better try again" << endl;
21             break;
22          default :
23             cout << "Invalid grade" << endl;
24      }
25      cout << "Your grade is " << grade << endl;
26
27      return 0;
28  }
```

Esempio di uso  
costrutto Switch

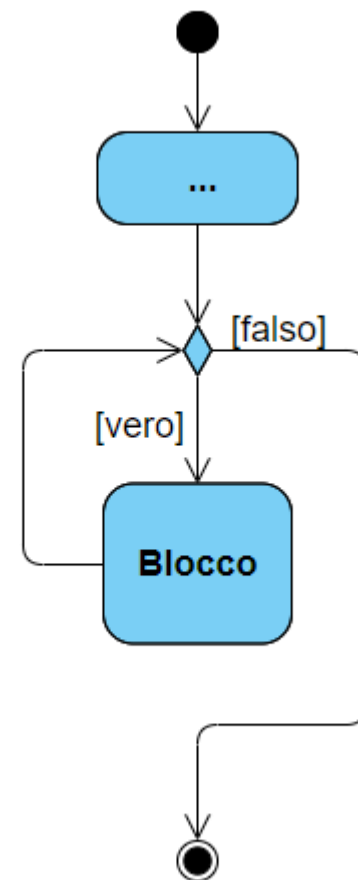
- Vogliamo realizzare il programma di un chiosco automatico per le immatricolazioni. L'immatricolazione ha un costo pari a X prestabilito, ma esistono delle condizioni di agevolazione:
  - Figlio orfano: 10% di sconto
  - Figlio di famiglia numerosa ( $\geq 4$  figli): 20% di sconto
  - Studente lavoratore: 5% di sconto
- Le condizioni non sono tra loro cumulabili
- Si crei un programma che chieda all'utente quale condizione lo rappresenta e gli mostri l'importo da pagare.
- Testcase:  $X = 1000$  Stato = {O, N, L}
- {'O', 900}; {'N', 800}; {'L', 950}



- Il costrutto “*fintanto che predicato* allora fai istruzioni” si mappa in C++ come segue:

```
while (predicato) {  
    //Blocco istruzioni  
}
```

- La parola chiave **while** è seguita da un predicato fra parentesi
- Il predicato viene valutato e, se vero, viene eseguito il blocco di istruzioni che segue la parentesi chiusa
- Alla fine dell'esecuzione del blocco, il predicato viene valutato ancora e se vero, il blocco viene eseguito nuovamente
- Il blocco di istruzioni viene eseguito ripetutamente finché il predicato non diviene falso



- La variabile *var* è **inizializzata** a 0. Il **while** si traduce in:
  - “Fintanto che *var* è minore di dieci, esegui il blocco di codice tra parentesi”
- La variabile è incrementata di uno ad ogni ciclo
- Dunque, dopo 10 iterazioni il while terminerà

```
#include <iostream>
using namespace std;
int main() {
    int var=0;
    while (var<10){
        cout<<"Dentro ciclo While: "<< var+1 <<endl;
        var++;
    }
}
```

## Output

```
> Dentro ciclo While: 1
> Dentro ciclo While: 2
> Dentro ciclo While: 3
> Dentro ciclo While: 4
> Dentro ciclo While: 5
> Dentro ciclo While: 6
> Dentro ciclo While: 7
> Dentro ciclo While: 8
> Dentro ciclo While: 9
> Dentro ciclo While: 10
```

- Può succedere che per un errore nel codice, oppure per una scelta progettuale si generi un **ciclo infinito**.
- Significa che il predicato non cambia, è sempre vero
- **NOTA** - Se dovesse succedere, per terminare l'esecuzione sul terminale è necessario digitare insieme due tasti: **ctrl** e **c**

```
int var=0;
while (true){
    var=var+1;
}
```

```
int var1=0;
int var2=5;
while (var2!=0){
    var1=var1+1;
}
```

In ambo i casi la variabile è incrementata all'infinito

- Il ciclo **do-while** è una particolare forma di **while**
- La differenza fondamentale tra il ciclo **do-while** e il ciclo **while** è che il **do-while** esegue l'esecuzione del ciclo almeno per una volta.

```
int main() {  
    int var=0;  
    do{  
        cout<<"Dentro ciclo While"<<endl;  
        var--;  
    }while (var>0);  
}
```

```
int main() {  
    int var=0;  
    while (var>0){  
        cout<<"Dentro ciclo While"<<endl;  
        var--;  
    }  
}
```

- Nel primo caso il predicato è valutato dopo aver eseguito le istruzioni. Nel secondo caso il predicato è valutato prima di eseguire le istruzioni

- Il **for** è un altro ciclo iterativo che può essere sempre ricondotto ad un **while**.
  - A differenza del while è apriori noto il numero di iterazioni del ciclo
- Ha la seguente sintassi:

```
for(valore_iniziale; condizione_di_test; incremento)
{
    (<istruzioni da eseguire all'interno del ciclo>)
}
```
- Il valore iniziale indica quale sarà la **variabile contatore** nel ciclo e ne impostiamo il valore iniziale
- La **condizione di test** rappresenta la condizione da verificare ad ogni passo del ciclo per valutare se è necessario continuare oppure uscire dall'iterazione
- L'**incremento** descrive come modificare, incrementare o decrementare il contatore ad ogni esecuzione

```
int totale=0;
for(int i=0;i<=10; i++){
    totale=totale+i;
}
cout<<"Il totale è:"<<totale;
```

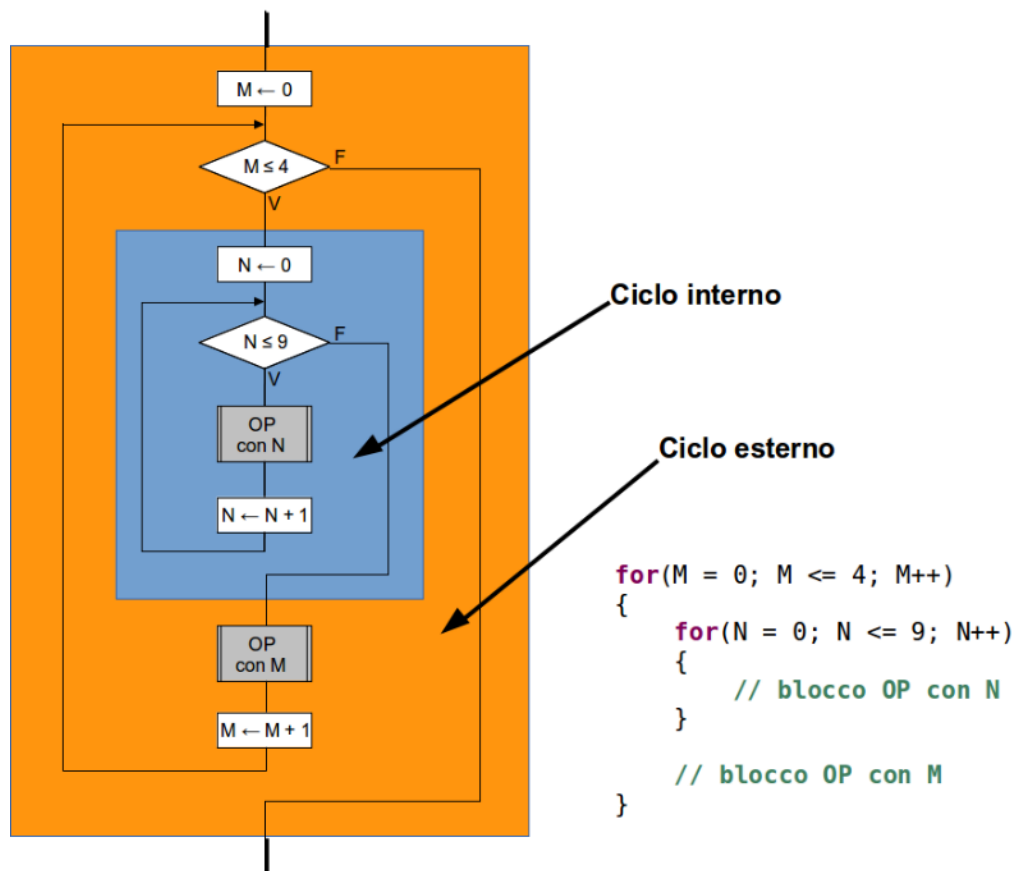
- ❑ Definiamo il **valore iniziale** attraverso l'espressione `int i=0`, con cui oltre a indicare la variabile `i` come variabile contatore inizializzata a 0, ne effettuiamo anche la dichiarazione.
- ❑ Nella **condizione di test** verifichiamo che `i` sia minore o uguale a 10, per rimanere all'interno del ciclo.
- ❑ Nell'espressione **incremento** ci limitiamo ad aggiungere 1 alla variabile `i` ad ogni passo del ciclo
- ❑ Il valore finale di `i` sarà 11 e non 10 perchè il conteggio è iniziato da 0 ed essendoci la condizione `<=` (invece di `<`) il codice è eseguito 11 volte

```
int i=0;  
int totale=0;  
while (i<=10){  
    totale=totale+i;  
    i++;  
}
```



```
int totale=0;  
for(int i=0;i<=10; i++){  
    totale=totale+i;  
}  
cout<<"Il totale è:"<<totale;
```

- E' chiaramente possibile definire costrutti, di tipo ciclico e/o selettivo, **annidati**. Ad esempio, nel caso del **for**:





```
#include<iostream>
#include<string>

using namespace std;

/* main */
int main() {
    cout<<"Quanti elementi vuoi inserire?"<<endl;
    int num_elems;
    cin>>num_elems;
    int num,sum;
    sum=0;
    for (int i=0;i<num_elems;i++){
        cout<<"Fornisci un numero: ";
        cin>>num;
        sum = sum + num;
    }
    cout<<"La somma e': "<<sum<<endl;
    return 0;
}
```

```
osboxes@osboxes:~/Documents/examples$ ./for_ex
Quanti elementi vuoi inserire?
3
Fornisci un numero: 3
Fornisci un numero: 4
Fornisci un numero: 3
La somma e': 10
osboxes@osboxes:~/Documents/examples$
```

```
#include<iostream>

using namespace std;

/* main */
int main() {
    int answer;

    cout << "Pick a choice from the list: " << endl;
    cout << "Pick from choices 1 (run), 2 (walk), or 3 (talk), depending on what you want from the menu"<<endl;

    do{
        cin >> answer;
        switch (answer){
            case 1:
                cout << "I want to run"<<endl;
                break;

            case 2:
                cout << "I want to walk"<<endl;
                break;
            case 3:
                cout << "I just want to talk to my friends."<<endl;
                break;
            default:
                cout << "Bad choice! Please try again"<<endl;
        }
    } while (answer <= 0 || answer > 3);

    return 0;
}
```

```
osboxes@osboxes:~/Documents/examples$ ./switch_do_ex
Pick a choice from the list:
Pick from choices 1 (run), 2 (walk), or 3 (talk), depending on what you want from the menu
5
Bad choice! Please try again
4
Bad choice! Please try again
2
I want to walk
osboxes@osboxes:~/Documents/examples$
```

1. Scrivere un programma che richiede 10 numeri decimali all'utente per farne poi la media e stamparne il risultato a video. Il programma dovrà anche verificare che l'utente non inserisca valori minori di zero. In tal caso, il numero non dovrà essere considerato nel calcolo della media.
  - Il programma dovrà essere scritto sia usando il **for** che il **while**

1. Scrivere un programma che somma i primi  $n$  interi ( $n$  fornito dall'utente)
2. Scrivere un programma che somma i numeri divisibili per 4 e 7 compresi tra 0 e 100
3. Scrivere un programma che somma i primi 10 numeri divisibili sia per 5 che per 8
4. Scrivere un programma che conta da 1 a 100 sommando i numeri pari e sottraendo i numeri divisibili per 3 (se un numero è pari e divisibile per 3 va prima sommato e poi sottratto, ossia saltato)
5. Scrivere un programma che dato un numero assegnato (tra 0 e 1000), chiede all'utente di indovinare il numero.
  - Ad ogni tentativo errato il programma suggerisce all'utente se il numero da indovinare è  $>$  o  $<$  del numero di tentativo.
  - L'utente deve indovinare in 10 tentativi altrimenti perde
6. Considerando lo stesso gioco del punto precedente, invertire i ruoli: l'utente pensa il numero e il sistema prova ad indovinare, ad ogni tentativo proposto dal programma, l'utente deve rispondere con una tra tre opzioni: indovinato, minore, maggiore. Il programma andrà avanti finché non indovina il numero.

HackerRank:

- <https://www.hackerrank.com/challenges/c-tutorial-for-loop/problem>