

Programmazione strutturata

Algoritmo

Dato un **problema** e un **esecutore**, l'algoritmo:

- è una **successione finita** di passi elementari (direttive);
- eseguibile **senza ambiguità** dall'esecutore;
- risolve il **problema** dato.

Algoritmo

Un algoritmo è una sequenza **ordinata** e **finita** di **passi elementari** che risolvono una **classe di problemi**.

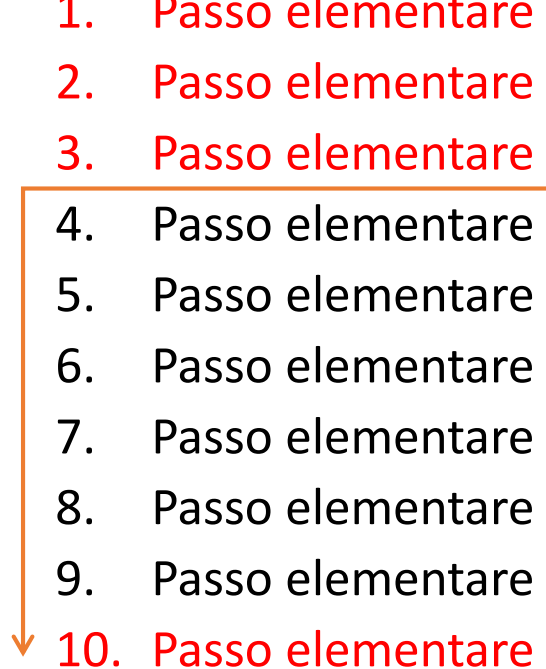
Algoritmo

Un **passo elementare** non è ulteriormente **scomponibile**: è un comando chiaro e inequivocabile.

- Sequenza di passi elementari
 - Passo elementare
 - Passo elementare
 - Passo elementare
 - Passo elementare
 - Passo elementare
 - Passo elementare

Algoritmo

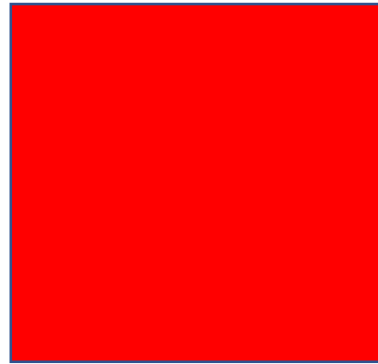
- Sequenza **ordinata** e **finita**

1. Passo elementare
 2. Passo elementare
 3. Passo elementare
 4. Passo elementare
 5. Passo elementare
 6. Passo elementare
 7. Passo elementare
 8. Passo elementare
 9. Passo elementare
 10. Passo elementare
- 

Algoritmo

Risolve una **classe di problemi**

Es. Calcolo dell'area di un quadrato



$l=30\text{cm}$



$l=10\text{cm}$



$l=6\text{cm}$

Algoritmo

Lavare il bucato in lavatrice

1. Se l'oblò è chiuso, aprilo
2. Metti i panni nel cestello
3. Chiudi l'oblò
4. Accendi la lavatrice
5. Imposta il programma
6. Metti il detersivo
7. Metti l'ammorbidente
8. Avvia il ciclo

Programma?

Che definizione dare?

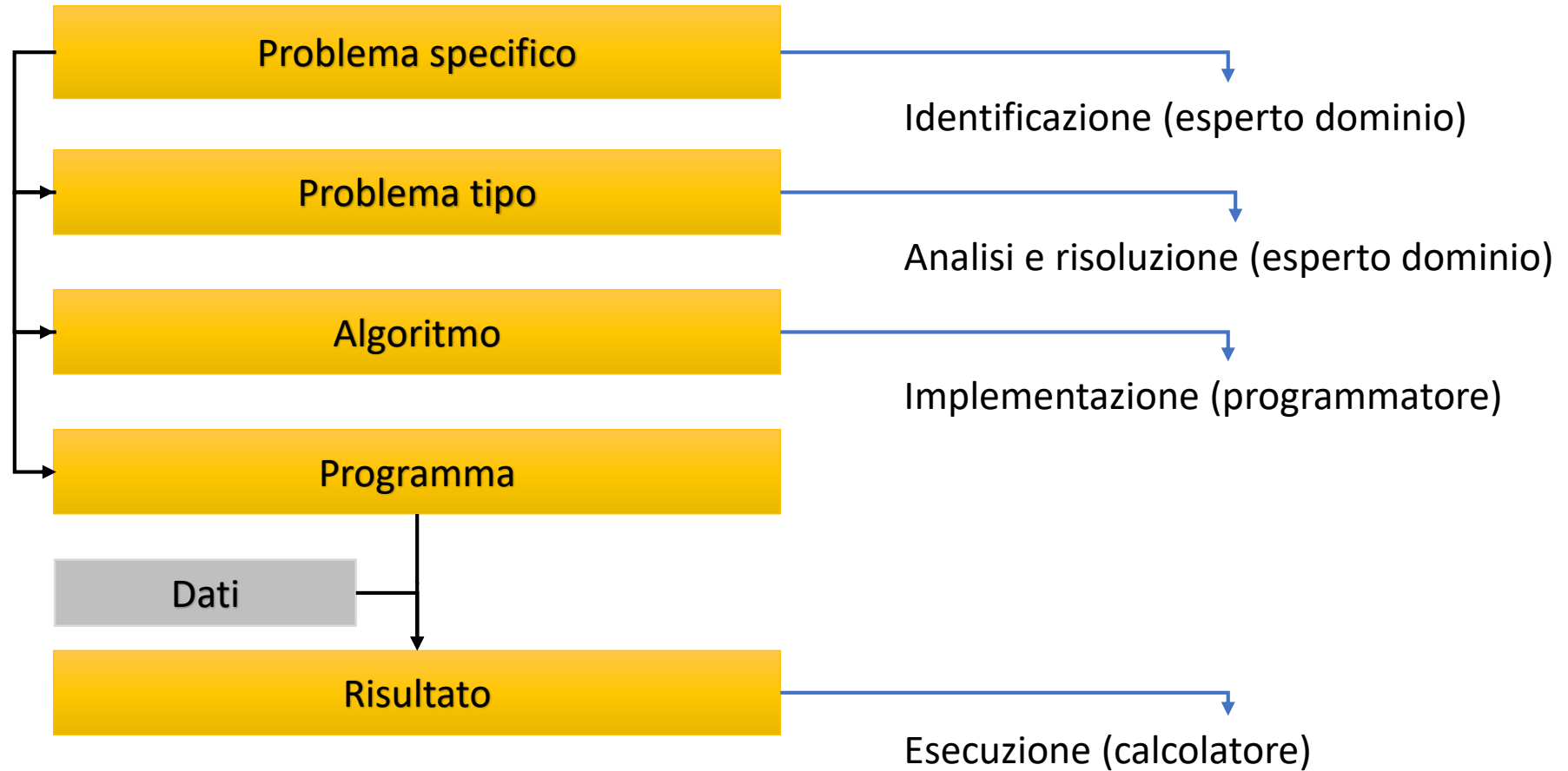
Programma vs algoritmo

- **Descrizione** di un algoritmo in un **linguaggio di programmazione**
- E' composto da un numero finito di **istruzioni**
- Ogni istruzione descrive una **(oper)azione**

Programma

*I programmi sono formulazioni
concrete di algoritmi astratti,
che si basano sulle particolari
rappresentazioni e strutture dei dati.
(N. Wirth)*

Programma vs algoritmo



Teorema di Böhm-Jacopini

Il teorema di **Böhm-Jacopini**, enunciato nel 1966 dagli informatici Corrado Böhme Giuseppe Jacopini, afferma che **qualunque algoritmo** può essere implementato utilizzando tre sole strutture:

- La **sequenza**
- La **selezione**
- Il **ciclo** (o **iterazione**) da applicare ricorsivamente alla composizione di istruzioni elementari.

Teorema di Böhm-Jacopini

- Il teorema è alla base della definizione delle linee guida della **programmazione strutturata** che si sono avuti intorno al 1970;
- ha contribuito alla critica dell'uso sconsigliato delle istruzioni **goto** (spaghetti programming)

Istruzione

Un **passo elementare/istruzione** può servire a:

- Acquisire un input
- Presentare un output
- Modificare il valore di una struttura dati
- Dirigere il flusso di esecuzione

Istruzione

- Qualsiasi algoritmo / programma, per adattarsi alle circostanze potrebbe non richiedere l'esecuzione in sequenza di tutti i suoi passi elementari.
- In altre parole, non è detto che dopo il passo 6 si esegua sempre il passo 7.

Istruzione/algoritmo

Lavare il bucato in lavatrice

1. Se l'oblò è chiuso, aprilo
2. Metti i panni nel cestello
3. Chiudi l'oblò
4. Accendi la lavatrice
5. Imposta il programma
6. Metti il detersivo
7. Metti l'ammorbidente
8. Avvia il ciclo

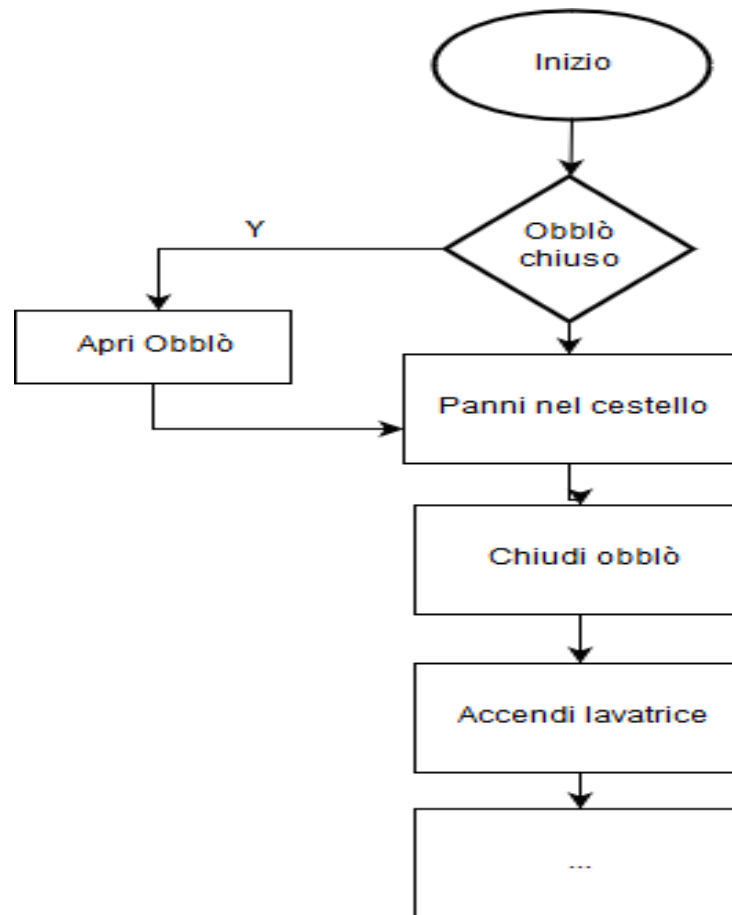
Lavare il bucato in lavatrice

1. Se l'oblò è chiuso,
 - a. Aprilo
2. Metti i panni nel cestello
3. Chiudi l'oblò
4. Accendi la lavatrice
5. Imposta il programma
6. Metti il detersivo
7. Metti l'ammorbidente
8. Avvia il ciclo

Istruzione/algoritmo

1. Il passo uno è scomponibile in due passi:
 - a. Se l'oblò è chiuso, aprilo
2. il secondo potrebbe non essere eseguito tutte le volte.
 - a. Se l'oblò è chiuso
 - 1) Aprilo

Lavare il bucato in lavatrice



Schemi a blocchi strutturati

- Si basano su **poche** strutture di base con **un solo ingresso** e **una sola uscita**;
- Impongono severe restrizioni nella costruzione dei diagrammi di flusso;
- Tali **strutture sono sufficienti** per descrivere qualsiasi algoritmo;
- Esse possono essere **annidate** tra loro o fluire in **sequenza** una dopo l'altra;

Schemi a blocchi strutturati

- Nel linguaggio del C e i suoi derivati i blocchi vengono evidenziati dai simboli:

«{» e «}»

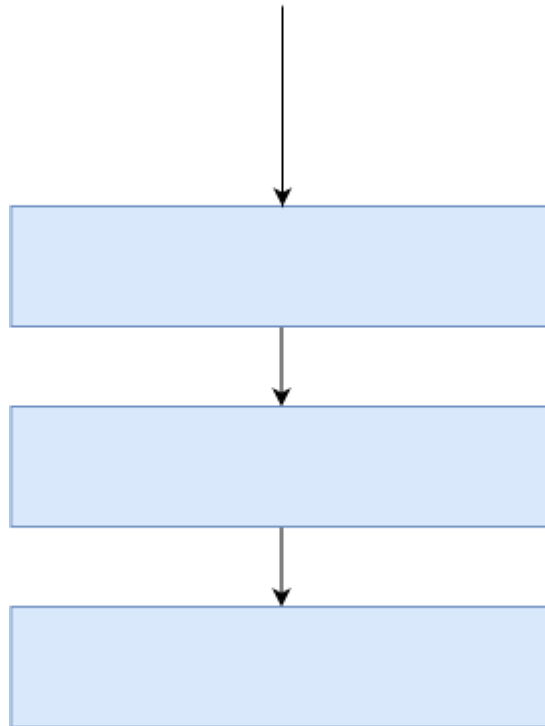
Sono utilizzate per **raggruppare** in un'unica istruzione composta detta anche **blocco**, dichiarazioni ed istruzioni, in modo che, dal punto di vista sintattico, esse formino un'entità **equivalente** ad una sola istruzione.

Schemi a blocchi strutturati

- Un esempio di blocco di istruzioni è il corpo di una funzione;

```
public void Create ()  
{  
    Istruzione1;  
    Istruzione2;  
    blocco di istruzioni { ... }  
    ...  
}
```

Blocchi strutturati - Sequenza




- La struttura sequenziale è **implicita** nei linguaggi C;
- Se non disposto diversamente le istruzioni vengono eseguite **una dopo l'altra**, così come sono scritte **incodizionalmente**;
- Per la sua rappresentazione si usa il simbolo **rettangolo**, chiamato anche simbolo di **azione**, per indicare un'attività che includa un calcolo o un'operazione di I/O;
- Non è possibile eseguire due istruzioni contemporaneamente;

Blocchi strutturati - Sequenza

- Può contenere definizioni di **variabili**
- **Istruzioni**: assegnamenti, invocazione di funzioni
- Terminano con «**;**» (punto e virgola)
- Il punto e virgola termina un'istruzione o dichiarazione/definizione di variabile;

Blocchi strutturati - Sequenza

- E' possibile inserire «;» anche dove **non** strettamente **necessario**:
- Esempio: **a = 1;;** (notare il doppio «;»)
- Che significato ha?
 - «;» indica sempre la fine di un'istruzione dunque:
 - Se l'istruzione non c'è, **l'istruzione è vuota**
 - Equivale a **non fare niente**
 - Equivale a **blocco vuoto «{}»**
 - **Evitare** questo fraintendimento (a = 1;;  a = 1; {};

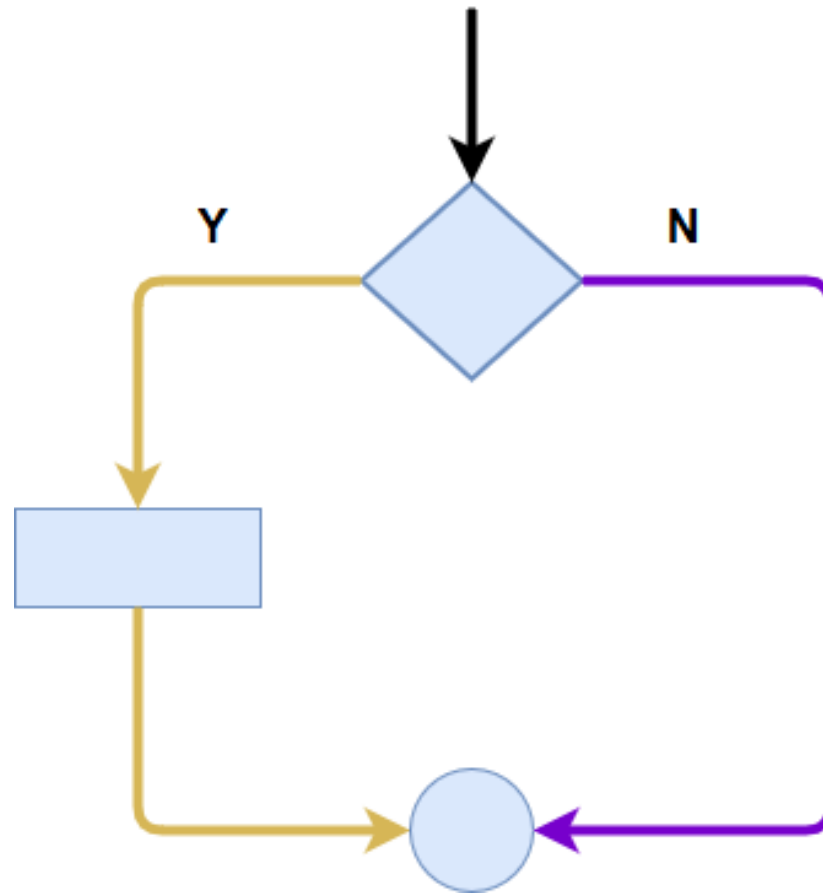
Blocchi strutturati - Selezioni

- Le istruzioni (o passi elementari) vengono **eseguite** solo dopo aver verificato una **condizione**;
- Le selezioni possono essere di **tre tipi**:
 - Selezione ad **una via**;
 - Selezione a **due vie**;
 - Selezione a **n vie**;

Selezione ad una via

if

Blocchi strutturati – Selezioni - If



Blocchi strutturati – Selezioni - If

- In inglese **If** significa «**se**»;
- Esprime il costrutto:
 - se **predicato** allora fai **istruzioni**
- La parola chiave **If** permette di seguire un **cammino di esecuzione diverso**;
- La **condizione** (predicato) è una espressione logica che può essere **vera** o **falsa**;
- Il comando **If** si aspetta **una sola istruzione** nel proprio corpo. Per includere più istruzioni è necessario creare un **blocco**.

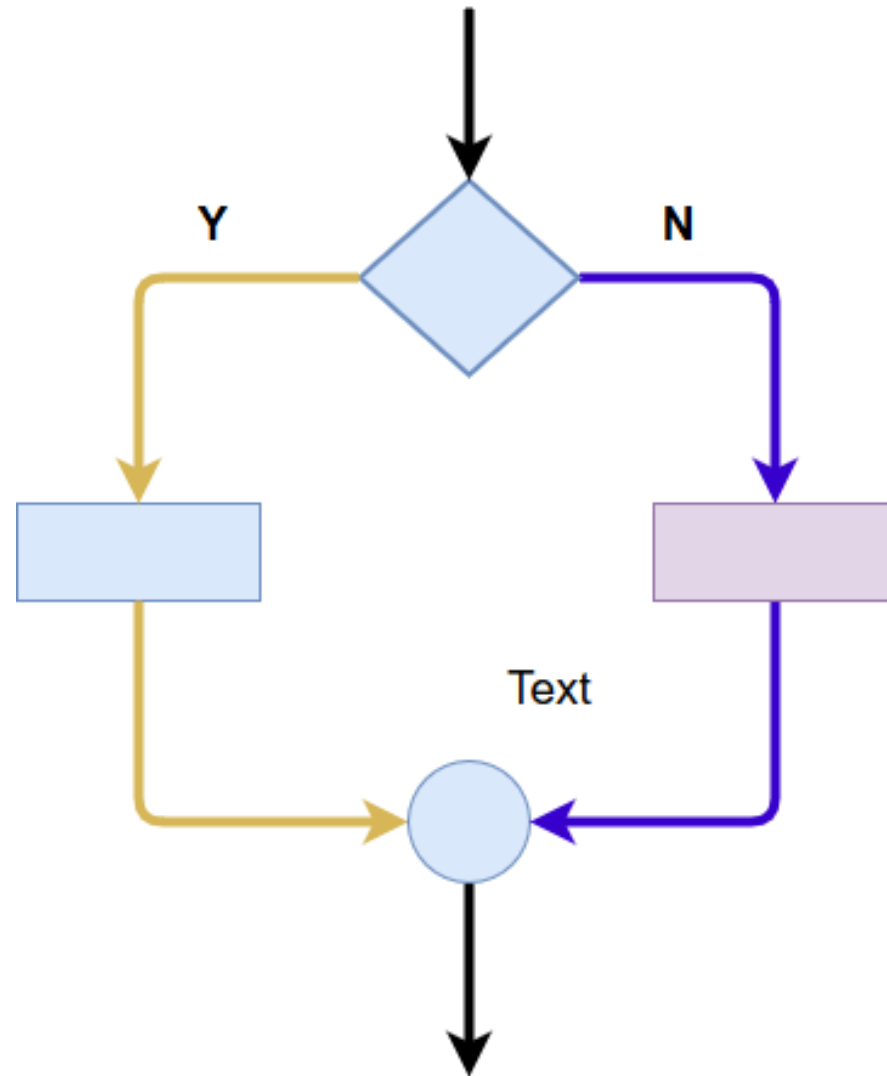
Blocchi strutturati – Selezioni - If

```
If (predicato/condizione)
{
    . . .
    istruzioni
}
```

Selezione a due vie

If-else

Blocchi strutturati – Selezioni – If-else



Blocchi strutturati – Selezioni – If-else

- In inglese **else** significa «**altrimenti**»;
- Le parole chiave **If** e **else** permettono di definire due **cammini di esecuzione alternativi**;
- La **condizione** (predicato) è una espressione logica che può essere **vera** o **falsa**;
- Il blocco **If** e il blocco **else** sono eseguiti in **alternativa**;
- Il blocco riferito **all'If** è **obbligatorio**;
- Il blocco riferito **all'else** è **opzionale**;
- Il costrutto crea **2 sottoblocchi** di istruzioni;

Blocchi strutturati – Selezioni – If-else

```
If (predicato/condizione)
{
    ...
    istruzioni
}
else
{
    ...
    istruzioni
}
```

Blocchi strutturati – Selezioni – If-else

- La parola **If** è seguita da un predicato/condizione tra **parentesi** tonde
- Il predicato/condizione viene **valutato** e se è **vero** viene eseguito il blocco di istruzioni subito seguente alla **parentesi** tonda chiusa
 - In questo caso il blocco che segue **else non viene eseguito**
- Se il predicato/condizione è **falso**, viene eseguito il blocco di istruzioni che segue la keyword **else**
 - In questo caso, il blocco che segue il predicato non viene eseguito

Blocchi strutturati – Selezioni - Errori

- La **condizione** deve essere sempre inclusa tra **parentesi** tonde
 - L'errore è sintattico e pertanto rilevato in compilazione
- L'istruzione **If non deve terminare con il «;»**
 - Questo non è un errore sintattico ma semantico visto che **significa**: la **condizione è vera non fare nulla!!**
- Un **«;»** sbagliato causa l'**esecuzione incondizionata** del blocco di istruzioni

Blocchi strutturati – Selezioni - Errori

```
If (n > 0) ; {
```

```
Console.WriteLine(numero maggiore di 0);
```

```
}
```

```
If (n > 0) {} ; {
```

```
Console.WriteLine(numero maggiore di 0); }
```

```
If (n > 0) {} ;
```

```
Console.WriteLine(numero maggiore di 0);
```

Blocchi strutturati – Selezioni - Errori

Non strutturato

```
Private uint GetValore(int n)
{
    if(n >= 0)
    {
        return Convert.ToUInt32(n);
    }
    return Convert.ToUInt32(-n);
}
```

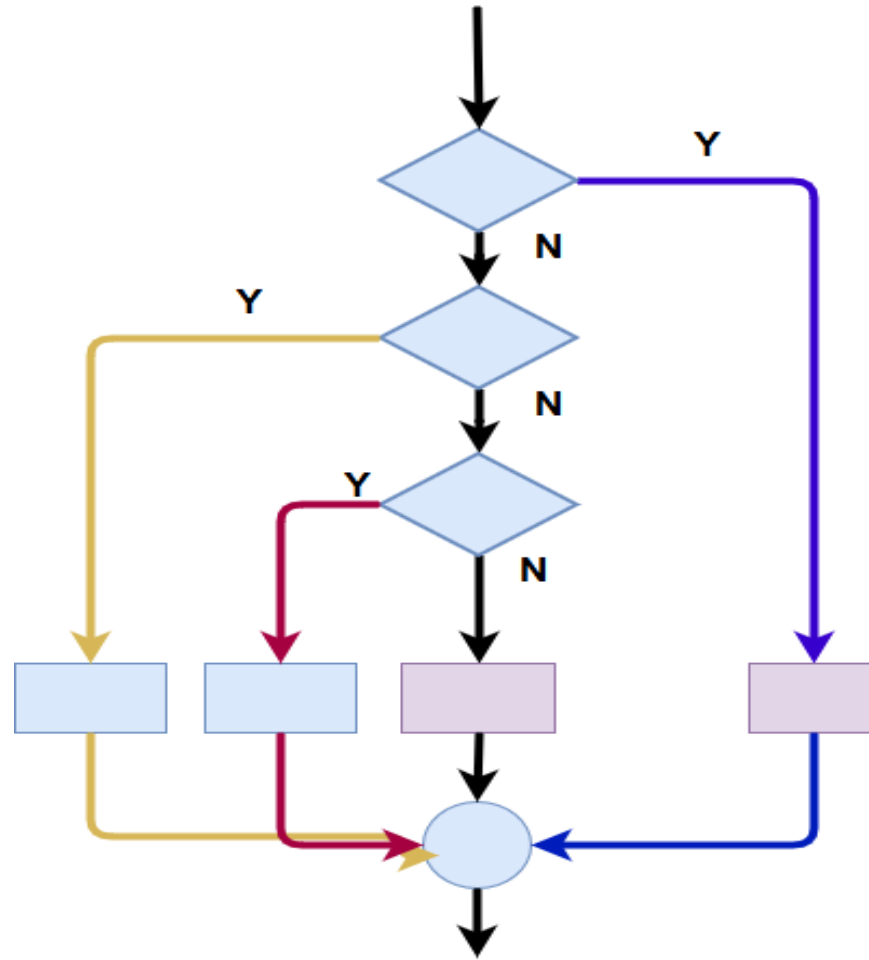
Strutturato

```
Private uint GetValore(int n)
{
    UInt32 risultato = 0;
    if(n >= 0)
    {
        risultato = n;
    }
    else
    {
        risultato = -n;
    }
    return risultato;
}
```

Selezione a **n vie**

else-if / switch

Blocchi strutturati – Selezioni – else-if



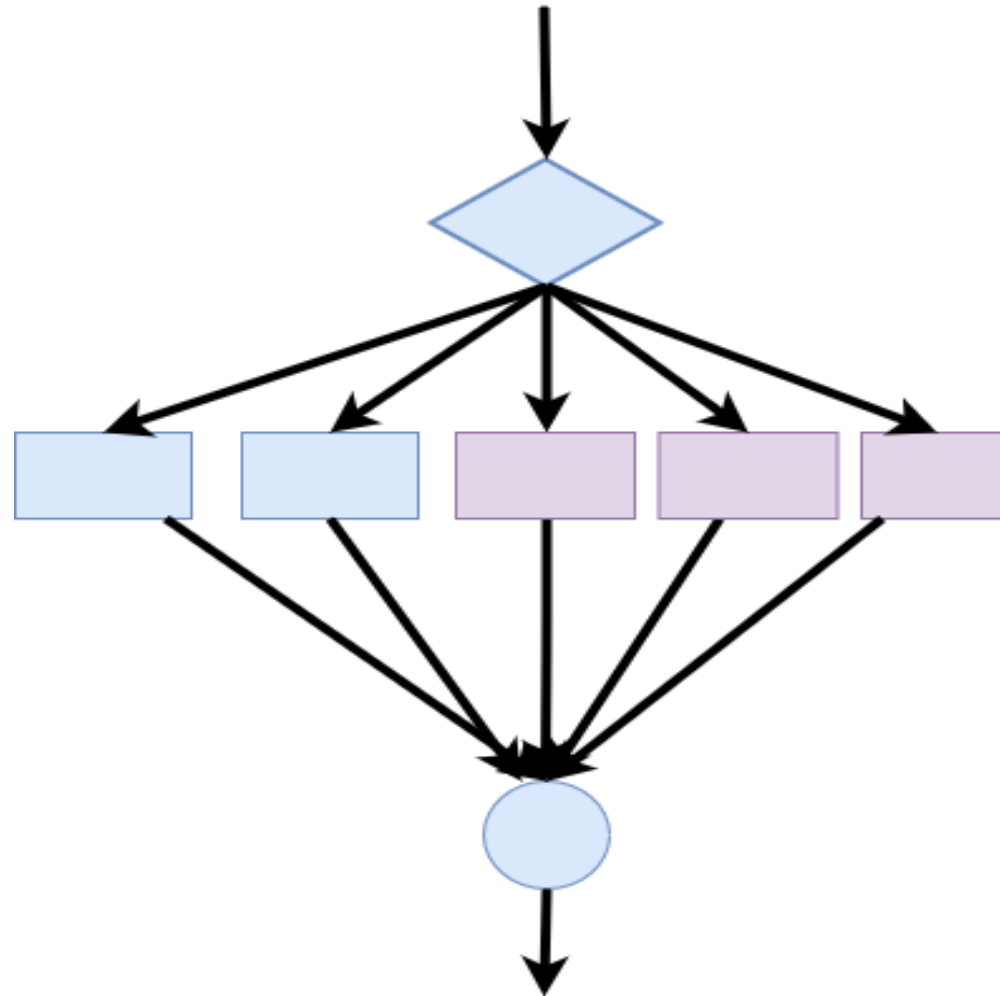
Blocchi strutturati – Selezioni – else-if

- Questa sequenza di istruzioni **if** è il modo più generale per realizzare una **scelta plurima**;
- Le espressioni sono analizzate nell'ordine in cui si presentano. Se una di esse risulta **vera**, l'istruzione ad essa associata viene **eseguita**, e ciò **termina** l'intera catena;
- Questi costrutti possono diventare molto **complicati** se nella scelta si devono valutare molte espressioni

Blocchi strutturati – Selezioni – else-if

```
var espr1 = 0;  
var istr1 = string.Empty;  
if (espr1 == 1)  
    istr1 = "primo step";  
else if (espr1 == 2)  
    istr1 = "secondo step";  
else if (espr1 == 3)  
    istr1 = "terzo step";  
else  
    istr1 = "step finale";
```

Blocchi strutturati – Selezioni – switch



Blocchi strutturati – Selezioni – switch

- L'istruzione switch è una struttura di **scelta plurima** che controlla se un'espressione assume un **valore** all'interno di un **insieme** di costanti intere;
- **L'espressione** da valutare e i **singoli elementi** sono espressioni in genere di tipo int o char;
- Non sono ammessi più **case** con lo stesso valore di **element**;
- **default** può comparire solo una volta;

Blocchi strutturati – Selezioni – switch

- Funzionamento dell'istruzione switch:
- L'espressione (expression) viene valutata e confrontata con le espressioni costanti (element_n);
- Se il valore di una (e quindi solo una) delle espressioni costanti è uguale al valore dell'espressione intera allora:
 - Si esegue la prima istruzione in posizione successiva ai due punti che seguono l'espressione costante;
- Altrimenti:
 - Se è presente default, si esegue la prima istruzione in posizione successiva ai due punti seguenti default;
- Tutte le istruzioni seguenti fino all'istruzione break vengono eseguite in sequenza;

Blocchi strutturati – Selezioni – switch

```
var expr1 = 20;
var result = string.Empty;
switch (expr1)
{
    case 10:
        result = "risultato 10";
        break;
    case 20:
        result = "risultato 20";
        break;
    default:
        result = "risultato default";
        break;
}
```

Espressioni condizionali

- I linguaggi derivanti C forniscono l'operatore condizionale (?:) che è strettamente correlato al comando if-else.
- Quello condizionale è l'unico operatore ternario del C, ovvero accetta in input tre parametri.
- Gli operandi insieme all'operatore ternario formano una espressione condizionale.
- `espressione_1 ? espressione_2 : espressione_3`
- Consente di scrivere codice in modo molto sintetico

Espressioni condizionali

```
var x = 0;
```

```
If (a > b)
```

```
    x = a;
```

```
else
```

```
    x = b;
```

```
x = (a > b) ? a : b;
```

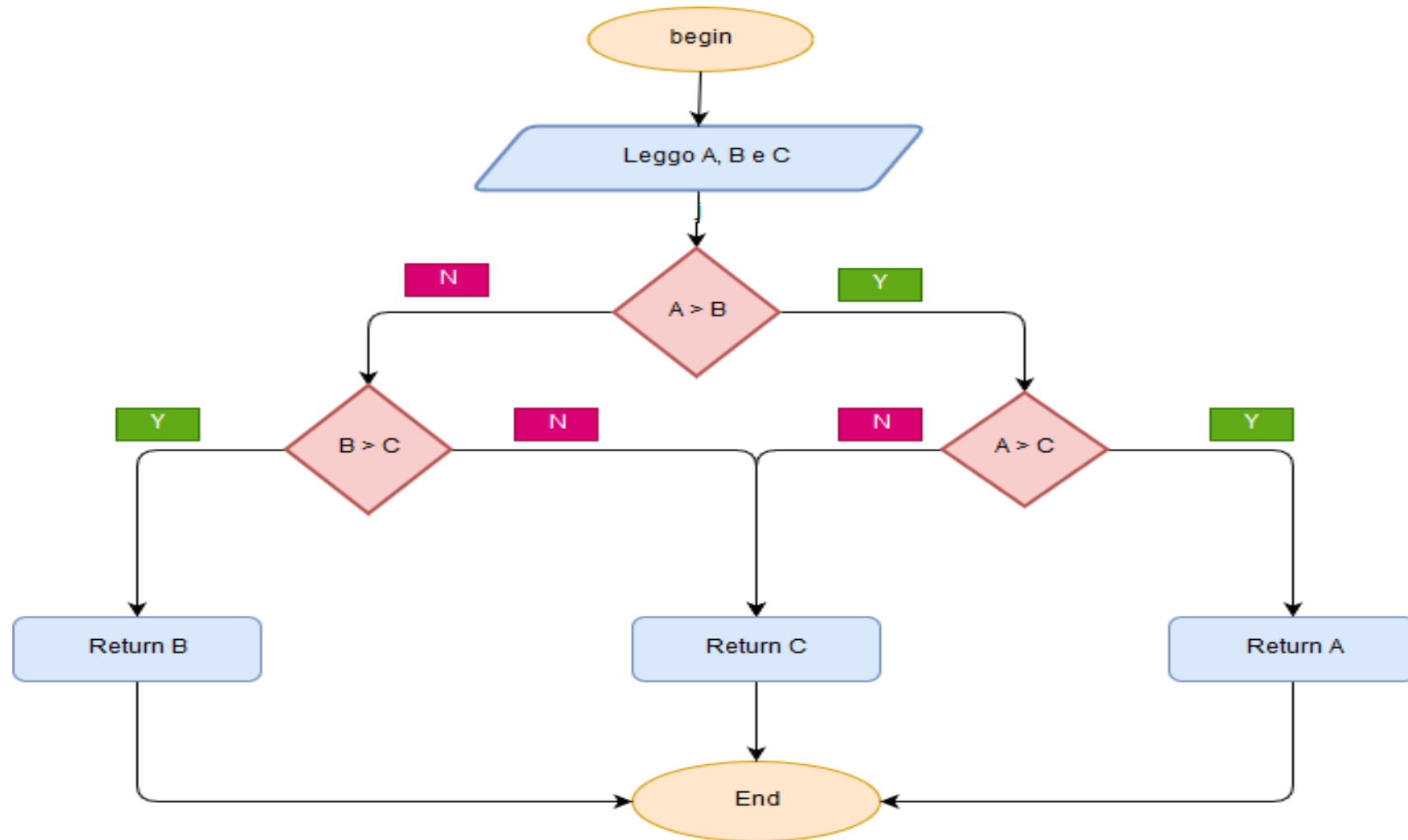
Esercitazione

1. Scrivere un programma che dati in input tre numeri visualizzi il numero maggiore;
2. Scrivere un programma che converte un carattere da maiuscolo in minuscolo utilizzando un'espressione condizionale;
3. Scrivere un programma che letto in input un numero intero positivo minore di 3 visualizzi il numero in caratteri (leggo 3 scrivo tre);

Esercitazione 1 - algoritmo

1. Leggo A B e C
2. Controllo **se** $A > B$
 1. Controllo **se** $A > C$
 - a. Mostro A
 2. **altrimenti**
 - a. Mostro C
3. **altrimenti**
 1. controllo **se** $B > C$
 - a. Mostro B
 2. **altrimenti**
 - a. Mostro C

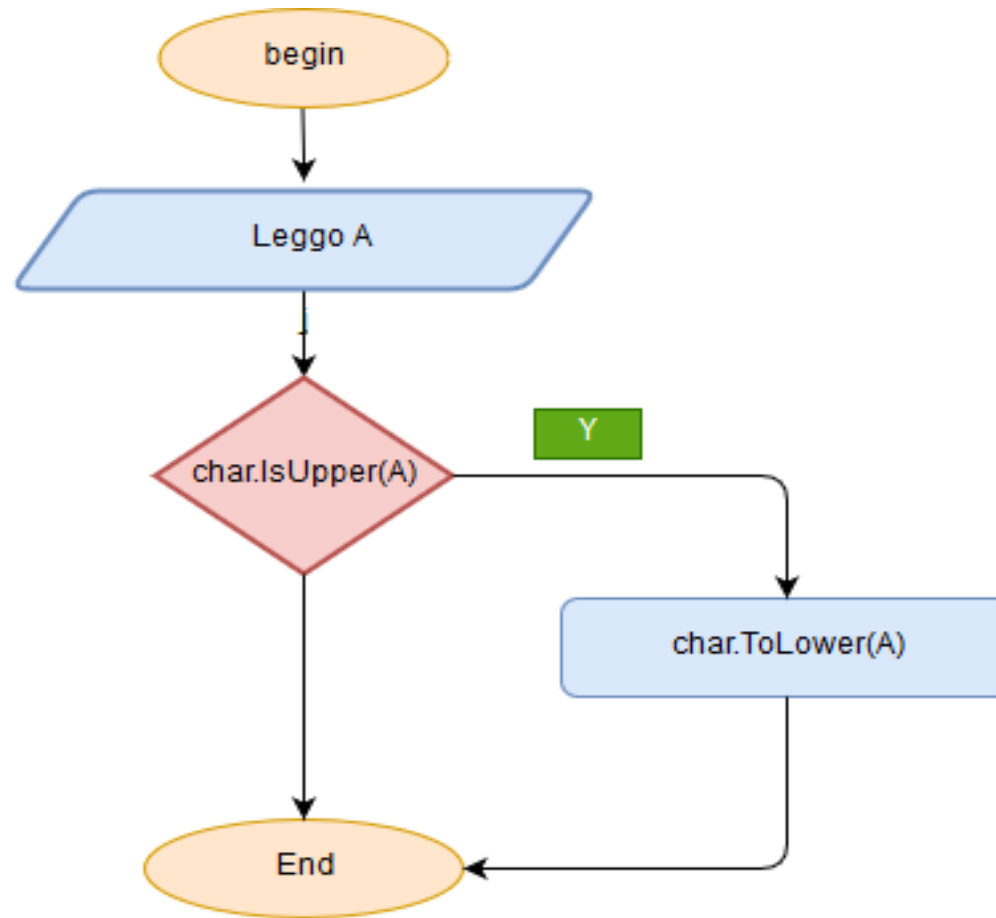
Esercitazione 1 - blocchi



Esercitazione 2 - algoritmo

1. Leggo **A**
2. **se** A è maiuscolo
3. **converti** A in minuscolo

Esercitazione 2 - blocchi



Blocchi strutturati – Iterazione/ciclo

- L'**iterazione** serve a fare in modo che un'istruzione o un blocco di istruzioni vengano eseguite **ripetutamente** per un certo numero di volte.
- Il **numero** dei cicli può essere:
- **Noto a priori**, essendo uguale al valore di una costante o di una variabile che **non muta** durante l'iterazione.
- Determinato da **condizioni** dipendenti **dall'iterazione**
- La difficoltà sta nel capire in che circostanze è più **opportuno** usare **l'uno o l'altro**.

Blocchi strutturati – Iterazione/ciclo

Elementi di un costrutto ciclico:

- **Inizializzazione**: le variabili usate nella **condizione** del ciclo, devono essere **inizializzate** prima della **valutazione** della condizione
- **Test**: Deve essere prevista una valutazione della condizione di permanenza nel ciclo, che **determini** la ripetizione o la terminazione del ciclo
- **Modifica**: almeno una delle variabili della condizione deve essere **modificata** all'interno del ciclo, in modo che **prima** o **poi** la condizione di ripetizione diventi **falsa**, causando la **terminazione** del ciclo

Blocchi strutturati – Iterazione/ciclo

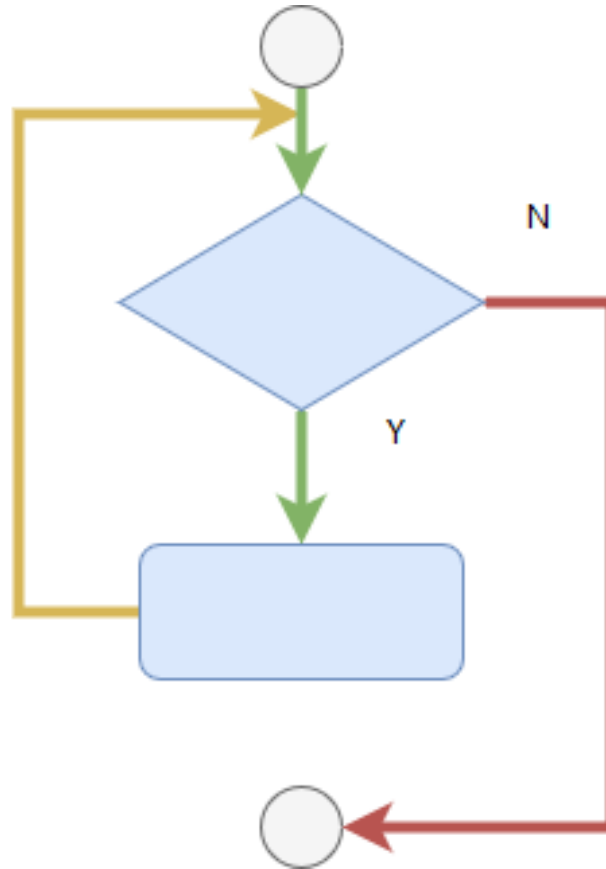
Possono essere di due tipi:

- Ciclo di ripetizione su **condizione**
 - Ciclo con condizione in **testa** (while)
 - Ciclo con condizione in **coda** (do-while)
- Ciclo di ripetizione su **contatore** (for)

Blocchi strutturati – Iterazione/ciclo

- Cicli con **contattore**
 - **Garanzia** che la **condizione** diventa **falsa** in un **numero noto** di passi;
 - Il ciclo **termina** in un tempo finito;
- Cicli con **condizione**
 - **Nessuna garanzia** su quando la condizione diventi **falsa**;
 - Il ciclo potrebbe **non terminare** mai;
- Per la correttezza dell'algoritmo è importante **prevedere** che il ciclo **termini**

Blocchi strutturati – Iterazione/ciclo - while



Blocchi strutturati – Iterazione/ciclo - while

- Si basa sul costrutto **mentre predicato esegui istruzioni**;
- In inglese **while** significa **mentre**;
- Il ciclo è **pre-condizionato** o con **condizione in testa**;
- Le istruzioni del blocco vengono **ripetute** fintanto che la **condizione** (espressione booleana) risulta essere **vera**;
- Se la condizione è **falsa** il ciclo si blocca;
- Se la condizione è **falsa prima** ancora di iniziare il ciclo, non viene eseguita neanche una iterazione;

Blocchi strutturati – Iterazione/ciclo - while

- La parola chiave **while** è seguita da una **condizione** tra **parentesi** tonde
- La condizione viene valutata e se è **vera** viene **eseguito** il blocco di istruzioni subito **seguito** alla **parentesi** chiusa
- Alla fine dell'esecuzione del blocco, la **condizione** viene valutata **ancora** e se è **vera** il blocco viene **eseguito** un'altra volta
- E così via
- Il blocco di istruzioni viene **eseguito** fino a che la **condizione** non diventa **falsa**

Blocchi strutturati – Iterazione/ciclo - while

```
int i=0;
while (i<10)
{
    Console.WriteLine("i={0}", i);
    i++;
}
Console.WriteLine("fine while");
```

Blocchi strutturati – Iterazione/ciclo - while

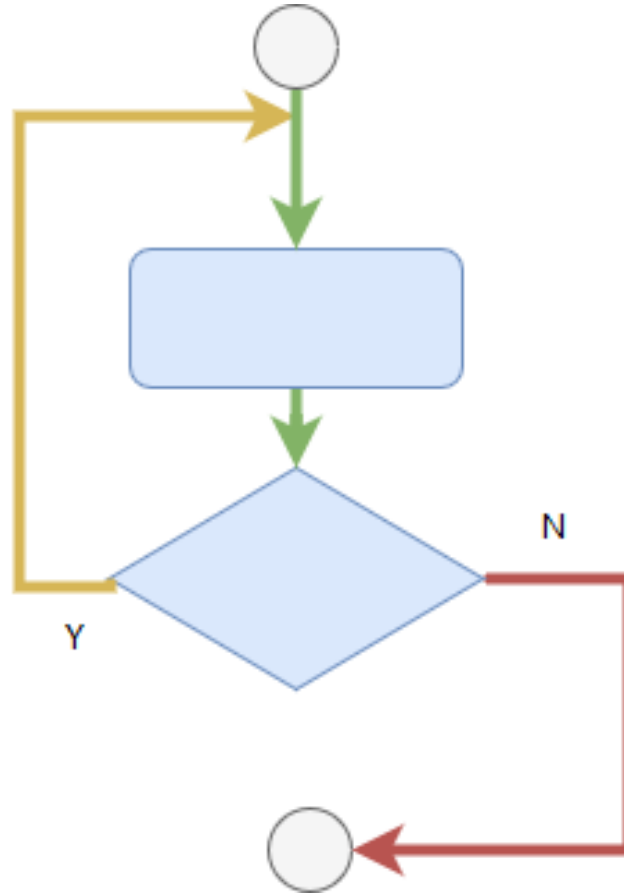
Loop

- L'introduzione dei cicli in un programma potrebbe **violare** la proprietà di **terminazione** di un programma
- Bisogna essere certi che **esista almeno** un caso in cui la condizione di permanenza diventi **falsa** per fare in modo che l'algoritmo **termini**
- Altrimenti il **programma** potrebbe entrare in stato di **loop**

Blocchi strutturati – Iterazione/ciclo - while

```
var a = 1;
while (a > 0)
{
    Console.WriteLine("I'm creating a loop! a ==
    {0}", a);
    a++;
}
```

Blocchi strutturati – Iterazione/ciclo – do-while



Blocchi strutturati – Iterazione/ciclo – do-while

- Si basa sul costrutto **esegui istruzioni mentre predicato;**
- Il ciclo è **post-condizionato** o con **condizione in coda;**
- Le istruzioni del blocco vengono **ripetute** fintanto che la **condizione** (espressione booleana) risulta essere **vera;**
- Il blocco viene **eseguito sempre** almeno una volta, anche se la condizione dovesse risultare **falsa** a priori
- Per l'iterazione while valgono le stesse considerazioni viste in precedenza

Blocchi strutturati – Iterazione/ciclo – do-while

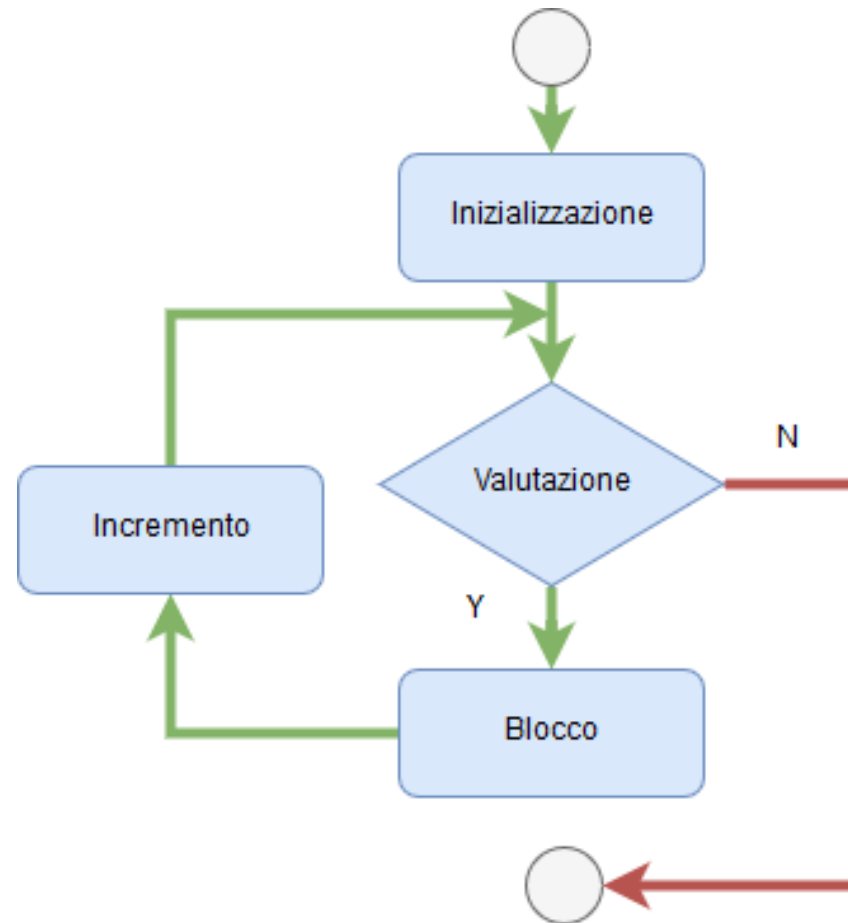
Quando conviene usarlo:

- Quando ha senso testare la condizione solo dopo aver già eseguito una prima volta il ciclo
- Quando c'è un obiettivo da raggiungere mediante uno o più tentativi

Blocchi strutturati – Iterazione/ciclo – do-while

```
char c;  
do  
{  
    Console.WriteLine("premi q per uscire");  
    c = Console.ReadKey().KeyChar;  
}  
while (c != 'q');
```

Blocchi strutturati – Iterazione/ciclo – for



Blocchi strutturati – Iterazione/ciclo – for

- Utilizzato ogni volta che è **noto** a priori il **numero** di cicli da eseguire:
 - Nel **while** la condizione di uscita è determinata da una **condizione generica**;
 - Nel ciclo **for** la condizione di uscita è basata su un **contatore**;
- Il contatore è una variabile intera che memorizza il **numero** di volte che il ciclo è stato **eseguito**;
- Gli elementi del ciclo for sono:
 - **Inizializzazione**: il contatore viene inizializzato;
 - **Test**: Il contatore viene confrontato con il limite superiore (inferiore);
 - **Modifica**: il contatore viene incrementato (decrementato);

Blocchi strutturati – Iterazione/ciclo – for

- Esso racchiude tutti e **tre** gli elementi sopra descritti:

```
for (inizializzazione; condizione; incremento)
{
    bloccoIstruzioni
}
```

Blocchi strutturati – Iterazione/ciclo – for

```
for(int i=0; i<10; i++)  
{  
    Console.WriteLine("hello world");  
}
```

Blocchi strutturati – confronto while/for

while

```
var i = 0;
while (i<10)
{
    Console.WriteLine"
    hello world");
    i = i + 1;
}
```

for

```
var i;
for(i=0; i<10; i++)
{
    Console.WriteLine"
    hello world");
}
```

Blocchi strutturati – cicli infiniti

while

```
int i=0;
while (true)
{
    Console.WriteLine("i
    = {0}", i);
    i++;
    if (i==10)
        break;
}
```

for

```
for (;;)
{
    Console.WriteLine("
    hello world ");
    if (condizione)
        break;
}
```


Blocchi strutturati – istruzione di salto

- L'istruzione **break** provoca l'uscita **incondizionata** da un ciclo for, while, do while;
- **Non valuta** le istruzioni **successive** né le condizioni di controllo;
- La lettura dei programmi risulta **difficile**;
- Si **sconsiglia** l'uso in particolare se esistono **alternative** più semplici;

Esercitazione

1. Calcolare i cubi degli interi da 1 a 10;
2. Scrivere un programma che calcoli il fattoriale di un numero intero;

Esercitazione

Replicare le operazioni di prelievo o deposito effettuate da una persona al bancomat.
ANALISI dell'algoritmo:

1. deve chiedere in Ingresso la password;
2. deve controllare la password;
3. deve chiedere in Ingresso il tipo di operazione e importo;
4. in caso di deposito deve aggiungere l'importo al saldo;
5. in caso di prelievo deve controllare il saldo:
 - a. Se l'importo è $>$ del saldo, deve formulare un messaggio di errore;
 - b. Se l'importo è $<$ del saldo, deve sottrarre l'importo al saldo e fornire il saldo aggiornato
6. deve chiedere se si vuole procedere con altra operazione;
7. altrimenti deve terminare.

Variabili in Ingresso: password, tipo di operazione, importo. Variabili in Uscita: errore o contante.