UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Fondamenti di Informatica

Ing. Alba Amato, PhD

alba.amato@unina2.it

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

RIEPILOGO ED ESERCITAZIONI

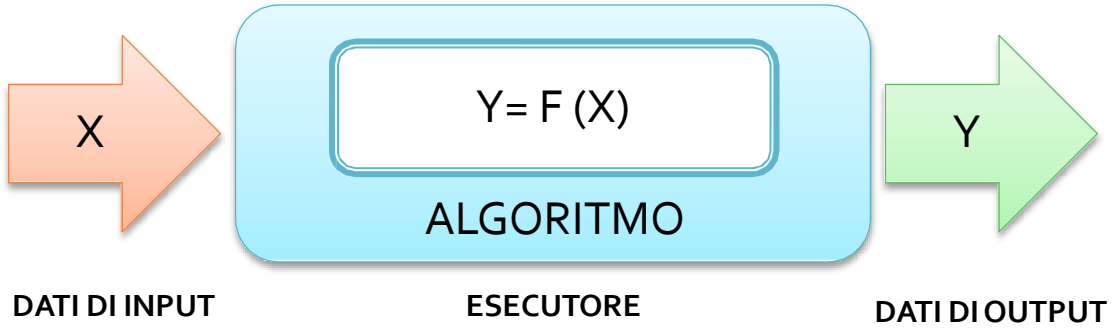
Lucidi tratti dal Web

Programmare

- Definire un insieme di attività che devono svolte secondo un ordine
  - Esempi di programmi
    - Libretto di istruzioni
    - Ricetta di cucina
    - Teorema matematico

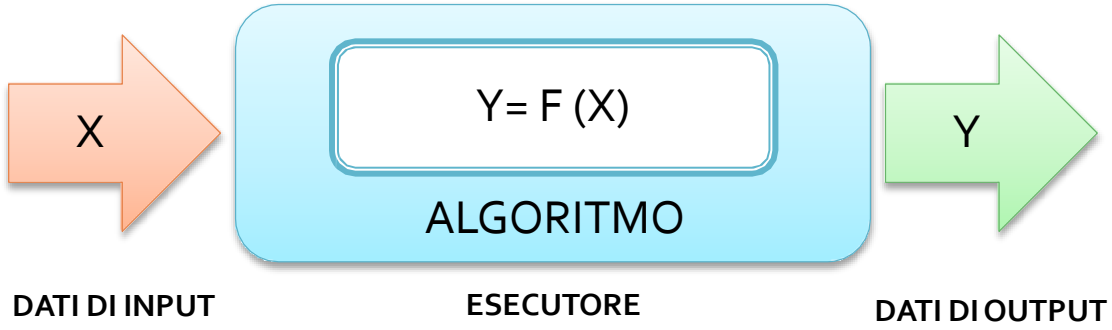
- **Esecutore** di un programma
  - Soggetto che è in grado di
    - Comprenderlo
    - Eseguirlo
  - **Istruzioni**
    - Frasi del programma che istruiscono l'esecutore sul da farsi
  - **Dati**
    - Oggetti di cui l'esecutore si serve per eseguire il programma

- Un qualsiasi programma elabora dati
  - E' una *trasformazione* di *dati di ingresso* in *dati di uscita*



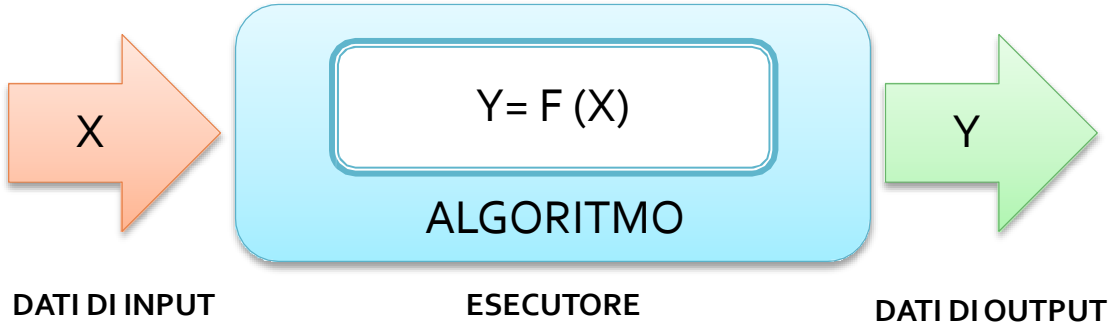
Linguaggio

- Un programma deve essere espresso in un *linguaggio* che è noto all'esecutore
  - Ma i procedimenti sono <u>indipendenti</u> dal linguaggio



Algoritmo

- Sequenza finita di passi da eseguire
  - Procedimento la cui validità è indipendente dal linguaggio



Programma

- *Traduzione* dell'algoritmoprogettato in un linguaggio comprensibileper l'esecutore a cui è destinato

Programmare

- Progettare algoritmi *indipendentemente* dal linguaggio dell'esecutore

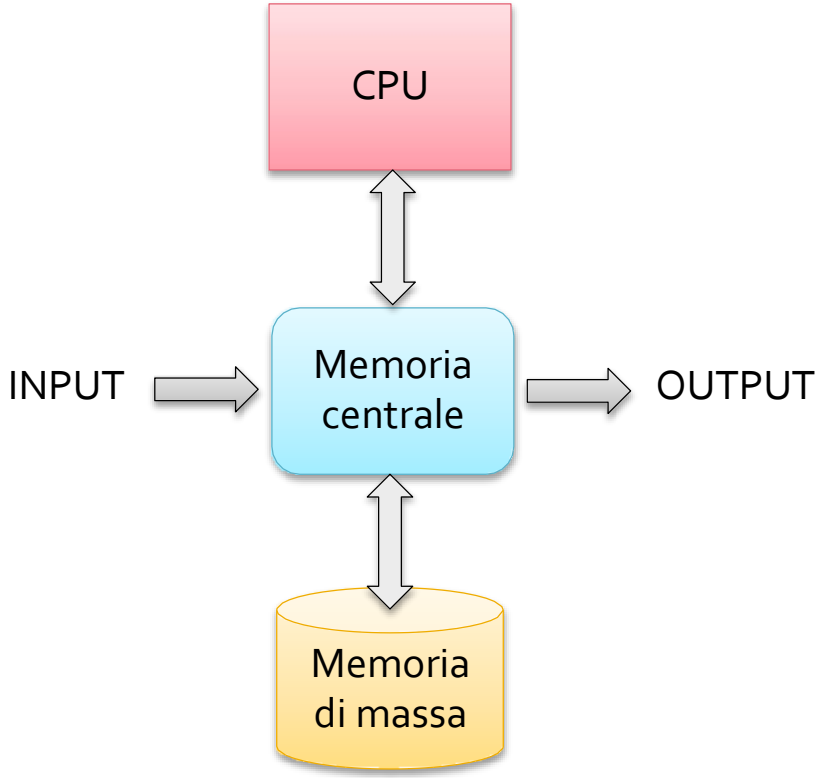
- L'informatica si interessa dello <u>studio degli algoritmi</u>
  - Trasformazioni delle informazioni in tutte quelle realtà che ne fanno uso
    - Non necessariamente devono essere eseguite dagli elaboratori
    - Il vantaggio dell'utilizzo degli elaboratori è dato dalla loro velocità e affidabilità

- Un po' di storia…
  - **Fortran**
    - Il primo linguaggio
      - Sviluppato a partire dal 1954, rilasciato nel 1957
      - Pensato per facilitare la scrittura di formule matematiche
  - **C**
    - Sviluppato nel 1972 da Dennis Ritchie presso i laboratori della AT&T Bell
    - Eredita principi e idee dal linguaggio B che a sua volta aveva ereditato da BCPL e CPL
    - Pensato per operare ad alto livello indipendentemente dalla macchina
    - Standard completato nel 1989 (ANSI C)
  - **C++**
    - E' una estensione del C
    - Formulato da Bjarne Stroustrup all'inizio degli anni '80, sempre presso i laboratori della AT&T Bell
    - Supporta la programmazione orientata agli oggetti (OOP)

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

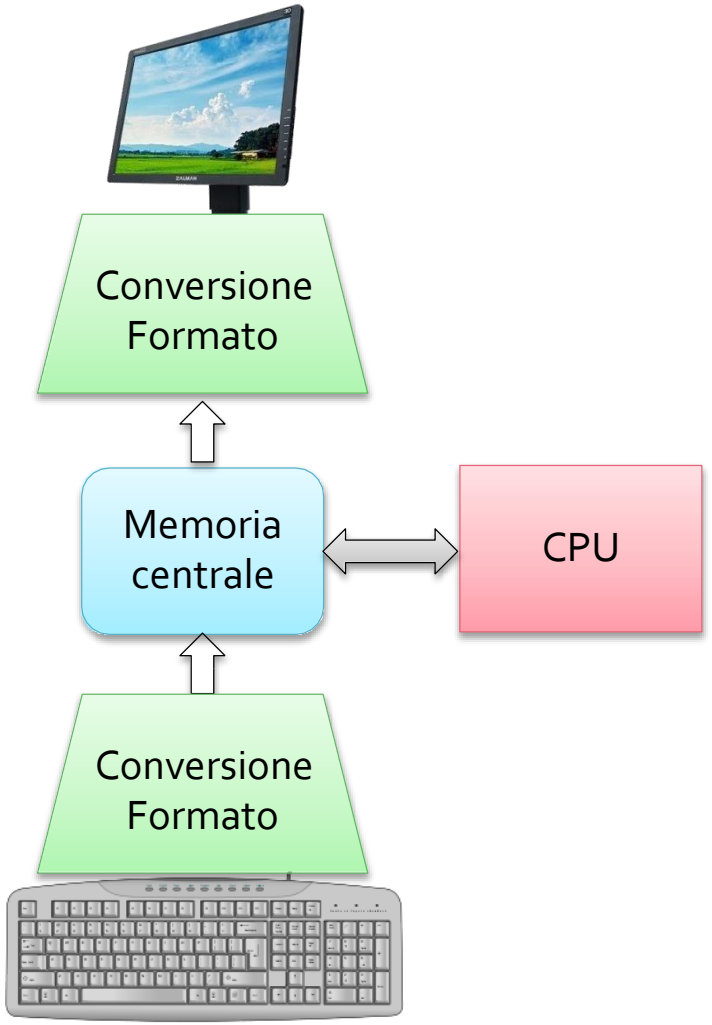
- Componenti fondamentali di un **computer**

  - Unità Centrale di Elaborazione (CPU)
    - Comprende ed esegue le istruzioni del programma
  - Memoria centrale
    - Contiene istruzioni e dati che servono all'esecuzione del programma
  - Dispositivi di input
    - Per inserire dati e istruzioni in memoria
  - Dispositivi di output
    - Per mostrare i risultati
  - Memorie di massa
    - Contiene i dati e le istruzioni che vengono poi inseriti nella memoria centrale durante l'esecuzione del programma
    - Può conservare i risultati prodotti
    - Dispositivi sia di input che di output



Input e output

- Stream di caratteri presa dalla codifica ASCII
  - in numero teoricamente infinito se
    - acquisito da tastiera (input standard)
    - restituito su terminale (output standard)
  - in numero finito se
    - scritto o letto in memorie di massa (file)
- Conversione da stream di caratteri in binario e viceversa



- Il legame tra la progettazione dei programmi ed il linguaggio è meno stretto di quanto si possa pensare
  - La potenza espressiva del linguaggio può essere di aiuto, ma è il fattore meno determinante per apprendere i fondamenti della programmazione
- Programmazione strutturata
  - Insieme di regole da seguire per progettare un programma di qualità e da adottare indipendentemente dal linguaggio di programmazione
- Qualità
  - Un compromesso tra obiettivi diversi:
    - Correttezza
    - Efficienza
    - Robustezza
    - Affidabilità
    - Usabilità
    - Estendibilità
    - Riusabilità
    - Strutturazione
    - Leggibilità
    - Manutenibilità
    - Modificabilità
    - Portabilità

- La progettazione di programmi è un'attività complessa
  - Separazione netta tra
    - Cosa -> Analisi dei requisiti e specifiche funzionali
    - Come -> Progetto a diversi livelli di dettaglio
  - Principi fondamentali
    - Modularità
    - Uso di strutture di controllo one in one out – NO SALTI
    - Approccio top-down e stepwise refinement
      - Dal generale al particolare per raffinamenti successivi
      - Metodo deduttivo più adatto agli esseri umani
    - Approccio bottom-up
      - Da moduli elementari a moduli più complessi passando per integrazioni successive
      - Metodo induttivo

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Struttura Dei Programmi

- Due parti:
  - Definizione di elementi
    - Variabili, tipi, costanti, funzioni
  - Istruzioni da eseguire

- `main()`
  - Deve sempre essere presente
  - La prima istruzione della sequenza main è la prima ad essere eseguita

```
/* hello.c */
#include <stdio.h>
int main() {
printf ("Hello world!\n");
return 0;
}
```

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Frasi di Commento

- Servono «solo» a migliorare la comprensione del programma da parte di un lettore
  - Non sono tradotte dal compilatore
- Tipi di commenti
  - Documentativo
  - Motivazionale (M:) Spiegano il significato delle istruzioni che seguono
  - Asserzionale (A:)
    Valori che devono assumere le variabili perché le istruzioni successive procedano in modo corretto

```
codice
codice /* Questo è un commento in stile C */
codice /* Anche questo è
un commento in stile C */
codice // Questo è un commento in stile C++
```

Identificatore

- Nome che si vuole attribuire ad una variabile
- Valido se composto di lettere, cifre e caratteri underscore ma che non inizia con una cifra

<identificatore> ::=
<lettera>|_{<lettera>|<cifra>|_}

- Il linguaggio è sensibile (case sensitive)
  - e.g. `mela` ≠ `Mela`
- Identificatori riservati
  - `if, main, function, int, …`

```
int a; // Dichiaro una variabile intera chiamata a senza inizializzarla
int b = 3; // Dichiaro una variabile intera b che vale 3
char c = 'q'; // Dichiaro una variabile char che contiene il carattere q
float d = 3.5; // Dichiaro una variabile float d che vale 3.5
a = 2; // Adesso a vale 2
int e = a+b; // e vale la somma di a e b, ossia 5
```

- Un identificatore può essere usato solo dopo essere stato «definito»
  - Per le variabili, va definito il «tipo» con la dichiarazione

```
int a;
```

nome_tipo identificatore_di_variabile

  - Dichiarazioni di più variabili dello stesso tipo

```
int a, b, numeroIntero;
```

  - Assegnazione valore iniziale
    - con = oppure con ()

```
int a = 10;       // a = 10
int b(10);        // b = 10
int c(a+1);       // c = 11
```

Dichiarazioni di Tipi

- La **dichiarazione** di tipo di una variabile
  - Fissa i *valori* che essa può assumere
  - Fissa le *operazioni* che si possono fare con essa
- Tipi atomici
  - Il compilatore fissa caratteristiche e regole d'uso delle variabili ad essi associati
    - `int, float, double, bool, char`
- Tipi non atomici
  - Necessitano di costruttori di tipo con cui se ne definiscono le caratteristiche
    - Es. dimensione degli array, campi delle strutture, ecc.
- Dichiarazione `typedef`
  - Rende più *leggibile* e *modificabile* un programma perché consente di
    - Assegnare un nome ai tipi definiti dal programmatore
    - Assegnare un alias (nome alternativo) ai tipi predefiniti

```
// senza typedef
float a, b, x, y;

// con typedef
float a, b;
typedef float COORDINATA;
COORDINATA x, y;
```

Costanti

- Valori prefissati non alterabili
  - Numerici o alfanumerici

- Costanti intere
  - Con segno (signed)
    - `+300, -12, +12, 12, 1000`
  - Senza segno (unsigned)
    - `12u, 1000u, 55u`

```
#define A 11u
#define B 011u
#define C 0x11u

int main ()
```

ottale

esadecimale

- Per le costanti unsigned, si possono adottare anche le notazioni
  - Ottale (anteponendo la cifra `0`)
  - Esadecimale (anteponendo la sequenza `0x`)

Costanti

Costanti reali

- In virgola fissa
  - `9.55  -1.34  +0.0001`
- In virgola mobile
  - `0.2e-23  15.001E-99  0.1E10`

Costanti carattere

- Un carattere ASCII racchiuso tra una coppia di apici ‘ ’
  - `'A  'a  '1  '*`

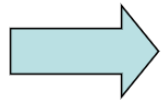
Costanti stringa di caratteri

- Sequenza di caratteri ASCII racchiusa tra una coppia di doppie virgolette ‟ ”
  - `"stringa  "due parole"  "A""`
- Caratteri speciali possono essere inseriti indicandoli con il carattere backslash
  - Sequenze di escape
    - `\n  \r  \'  \"  \?  \\`, ecc.

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

```
#define PI 3.14
const int raggio = 10;

int main() {
    float area;
    area = raggio * raggio
      * PI
    return 0;
}
```



```
int main() {
    float area;
    area = 10 * 10 * 3.14
    return 0;
}
```

Costanti

- Migliora la leggibilità e la parametricità del programma
  - Manutenzione più semplice
    - Modo 1: Direttiva `#define`

```
#define nome_costante valore_costante
```

      - Al nome della costante non viene associata memoria
      - Non deve essere terminata dal ;

    - Modo 2: Prefisso `const`

```
const nome_tipo nome_costante = valore;
```

      - Al nome della costante viene associata memoria
      - Deve essere terminata dal ;

A differenza delle variabili tale valore non cambierà mai durante tutto il programma

Esempi

- #define nmaxp 10
- #define vmax 150.0
- #define FALSE 0
- #define TRUE 1
- #define MIASTRINGA "Hello world!"

- Tipo
  - Intero
  - Reale
  - Booleano o logico
  - Carattere

```
Size of int   = 4
Size of float = 4
Size of bool  = 1
Size of char  = 1

Premere un tasto per continuare . . .
```

- Con essi si specifica:
  - l'insieme di valori che la variabile può assumere
    - dipendente dal tipo di occupazione in memoria prevista dal compilatore usato
  - le operazioni permesse su di essi
- Il tempo di calcolo di una espressione dipende fortemente dalla quantità di memoria coinvolta
  - usare le variabili che rappresentano la realtà da trattare con minore memoria usata rende il calcolo più efficiente
- `sizeof(nome_tipo)`
  - restituisce la occupazione effettiva di memoria di un tipo
    - es. `sizeof(char)` = 1 con un qualsiasi compilatore

- Tipo
  - Anche per il tipo reale la rappresentazione in memoria dipende dallo specifico compilatore
  - In generale si assume però che
    - la precisione di `float` sia minore o a uguale a quella di `double`
    - la precisione di `double` sia minore o uguale a quella di `long double`
  - Es. dichiarazione di variabili reali
    - `float alpha;`
    - `double beta;`

- Le variabili di tipo booleano assumono uno dei due valori di verità
  - TRUE
  - FALSE

- Occupano un solo byte e si dichiarano con bool
  - Es. `bool cond;`

- Con char si dichiarano variabili contenenti uno dei caratteri della tabellina ASCII

```
char carattere = 'A';
```

- Le variabili di tipo carattere occupano un solo byte
- Il linguaggio consente anche di operare direttamente sul codice del carattere considerando la variabile come un intero

valori tra 0 e 255 se dichiarata `unsigned char`
valori tra - 128 e 127 se dichiarata `signed char`

- Operatore `enum`
  - consente di dichiarare un nuovo tipo elencando i valori costanti che lo compongono

```
enum colori {bianco, rosso, verde}
```

  - Il compilatore associa ad ogni costante il valore intero corrispondente alla posizione occupata
    - bianco=0, rosso=1, verde=2
  - Possono anche essere esplicitati i valori interi

```
enum colori {bianco=4, rosso=5, verde=6}
```

- Istruzione che consente di modificare il valore di una variabile

```
variabile = espressione;
```

- Passi
  1. Viene calcolato il risultato dell'espressione
  2. Il risultato viene assegnato alla variabile al primo membro

- In una istruzione di assegnazione l'espressione deve essere di tipo «compatibile» con quello della variabile
  - Sono tipi compatibili
    - I tipi numerici se il valore che si assegna può essere contenuto nella variabile che lo riceve

- **Coercizione (type casting)**
  - Si possono imporre conversioni di tipo con due modalità

```
variabile = (tipo) espressione;
variabile = tipo(espressione);
```

  - Esempio:

```
int   i;
float x = 5.11;

i = (int)x;
cout << "i=" << i << endl;
```

```
i = 5
```

- Aritmetici

  `+  -  *  /  %`

- Logici

  `&&  ||  !`

- Bitwise
  - Operano sui singoli bit della rappresentazione

    `&  |  ^  ~  <<  >>`

- Relazionali
  - Restituiscono TRUE se il confronto tra due valori ha successo, FALSE se fallisce

    `==  !=  >  <  >=  <=`

- Incremento e decremento unario
  - **Prefisso**: incremento/decremento prima dell'uso della variabile

```
++i   --i
```

  - **Postfisso**: prima si usa la variabile, poi la si incrementa/decrementa

```
i++   i--
```

- Composti

```
var = var op espressione;
var op = espressione;
```

Esempio: `b=b*4` equivalente a `b*=4`

- ## Condizionale ?
  - Restituisce il primo risultato se la condizione (di tipo logico) è TRUE, il secondo se la condizione è FALSE

```
c = a<b ? b : a;
```

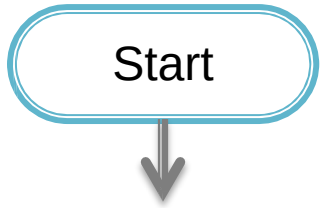
- ## Virgola
  - Consente di elencare più espressioni in un secondo membro di un'istruzione di assegnazione
    - solo il valore dell'espressione più a destra viene assegnato alla variabile

```
a = (b=10, c=b+10, c+10);
```

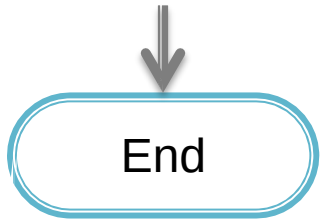
- ## Precedenza operatori
  - In una espressione con diversi operatori vengono eseguite per prime le operazioni con priorità più elevata (si veda tabella sul libro di testo)
  - In presenza di operatori con egual priorità si stabilisce di procedere da sinistra verso destra
  - L'introduzione di parentesi consente di cambiare l'ordine di esecuzione delle istruzioni

- I diagrammi di flusso o flow chart sono un formalismo che consente di rappresentare graficamente gli algoritmi
  - descrivono le azioni da eseguire ed il loro ordine di esecuzione
- ogni azione corrisponde ad un simbolo grafico (blocco)
  - ogni blocco ha un ramo in ingresso ed uno o più rami in uscita

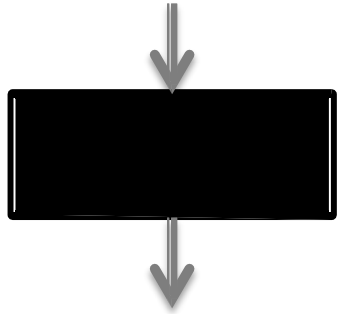
Inizio



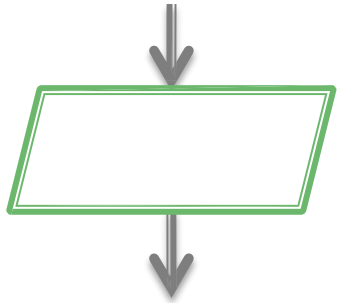
Fine



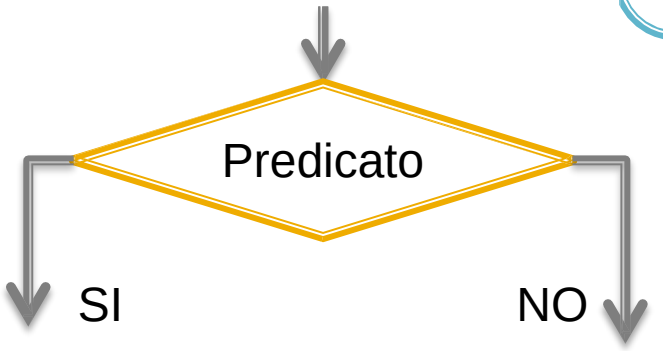
Elaborazione

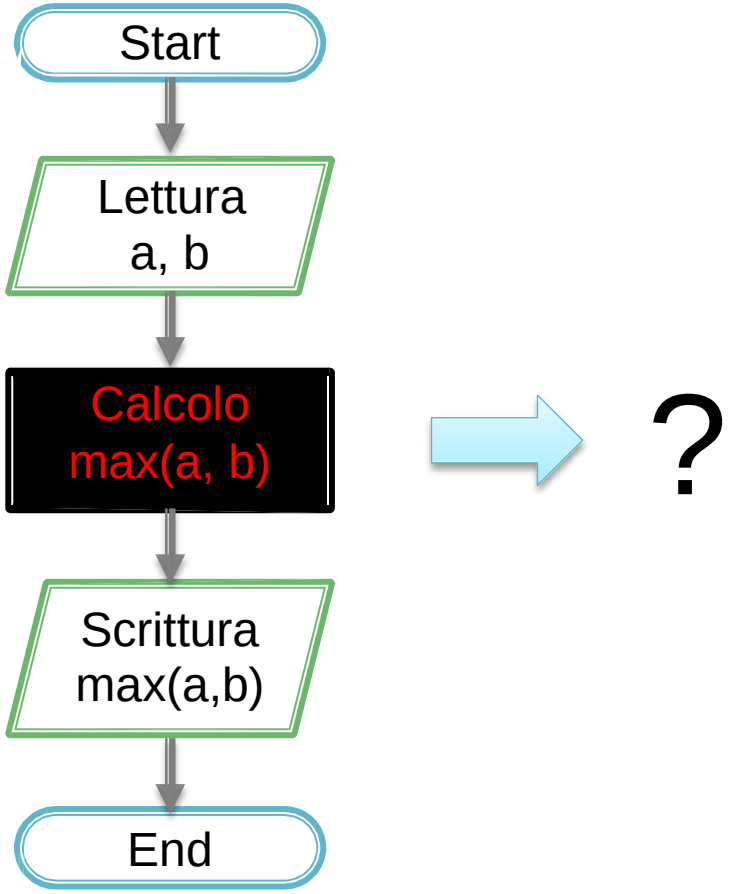


I/O

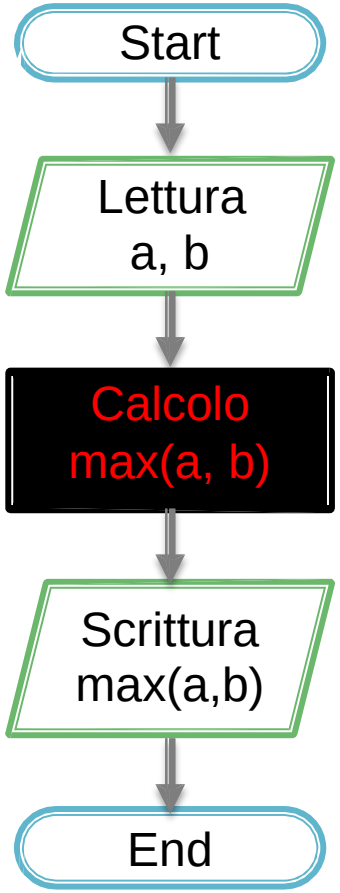


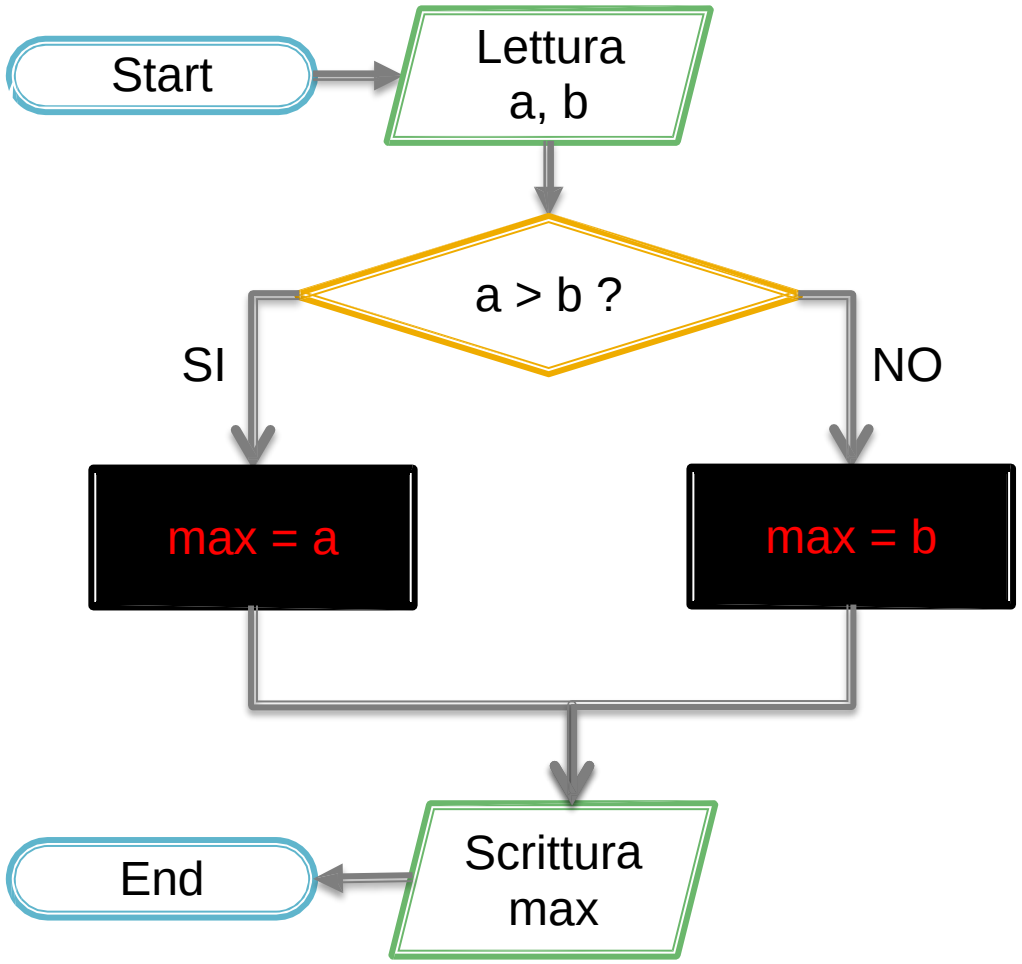
Selezione a due vie

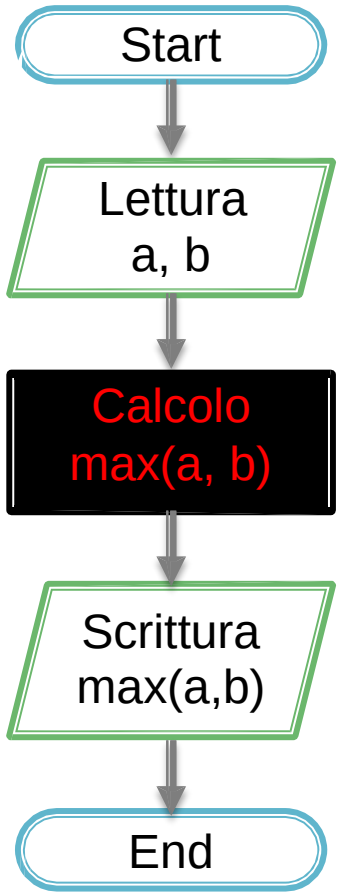


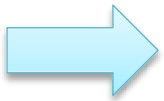

Start
Lettura
a, b
Calcolo
max(a, b)
Scrittura
max(a,b)
End

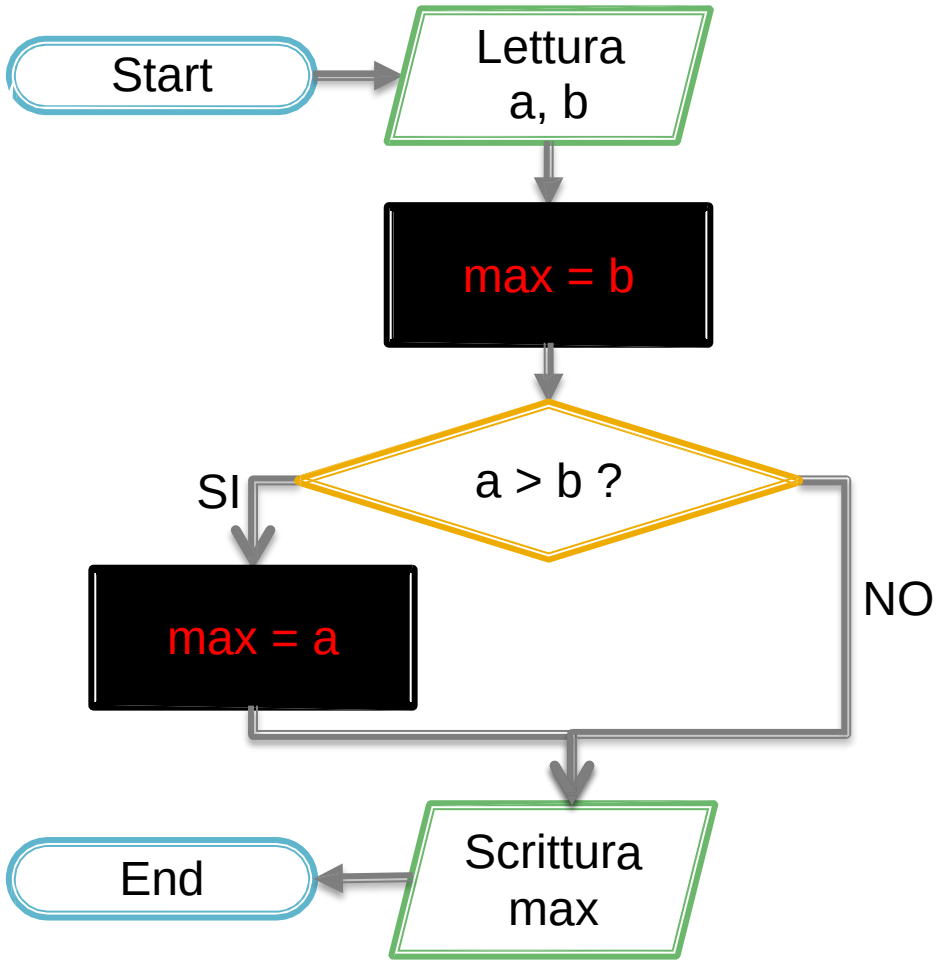
?


Start
Lettura
a, b
Calcolo
max(a, b)
Scrittura
max(a,b)
End


Start
Lettura
a, b
a > b ?
SI
NO
max = a
max = b
Scrittura
max
End

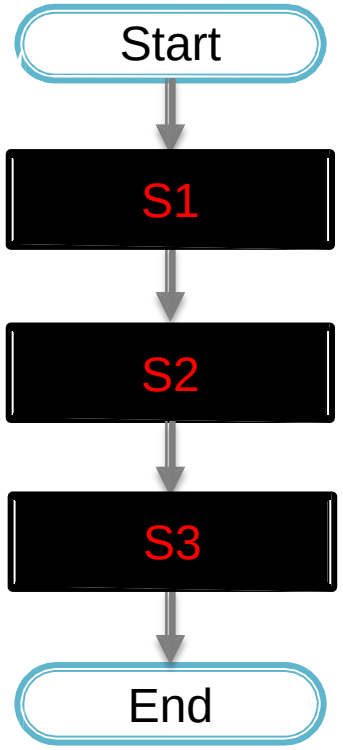

Start
Lettura
a, b
Calcolo
max(a, b)
Scrittura
max(a,b)
End




Start
Lettura
a, b
max = b
a > b ?
SI
NO
max = a
Scrittura
max
End

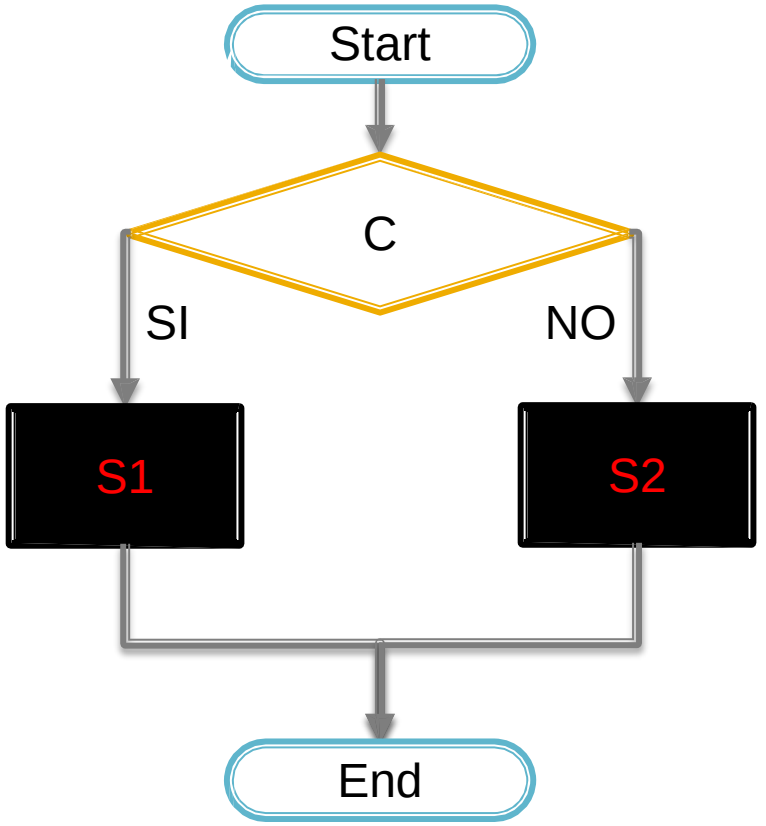
- **Teorema di Böhm-Jacopini** (1966)
  - Qualunque algoritmo può essere implementato utilizzando **tre** sole strutture
    - Sequenza
    - Selezione
    - Iterazione o ciclo

Sequenza



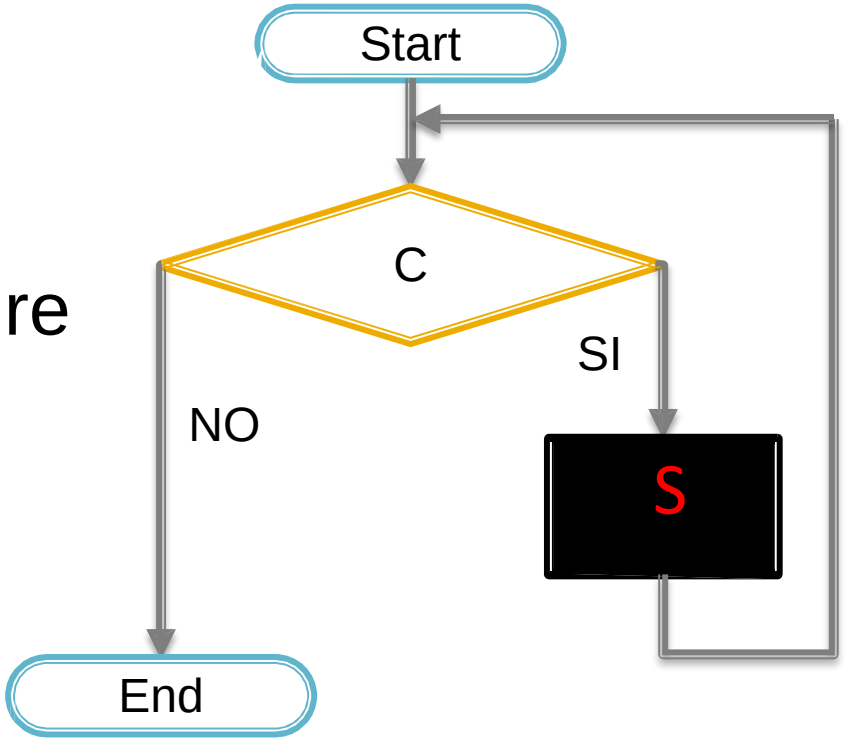
- **Teorema di Böhm-Jacopini** (1966)
  - Qualunque algoritmo può essere implementato utilizzando **tre** sole strutture
    - Sequenza
    - Selezione
    - Iterazione o ciclo

Selezione

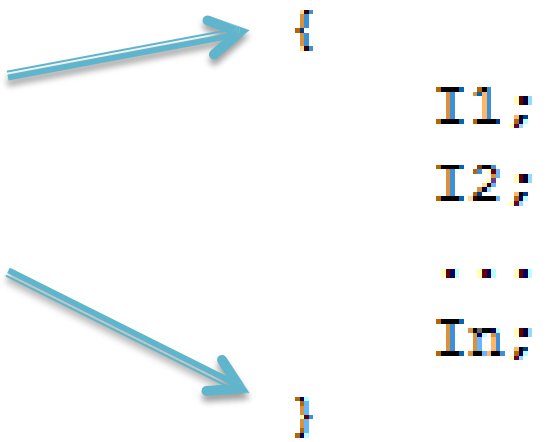


- **Teorema di Böhm-Jacopini** (1966)
  - Qualunque algoritmo può essere implementato utilizzando **tre** sole strutture
    - Sequenza
    - Selezione
    - Iterazione o ciclo

Iterazione



- Elemento base per la costruzione di un programma
- Formato dalla sequenza S di n istruzioni
  - $I_1; I_2; \ldots I_n$
- L'inizio del blocco è indicato da una **parentesi graffa aperta**
- La fine del blocco è indicata da una **parentesi graffa chiusa**
- Le istruzioni vengono eseguite secondo l'ordine di lettura
  - Dall'alto verso il basso

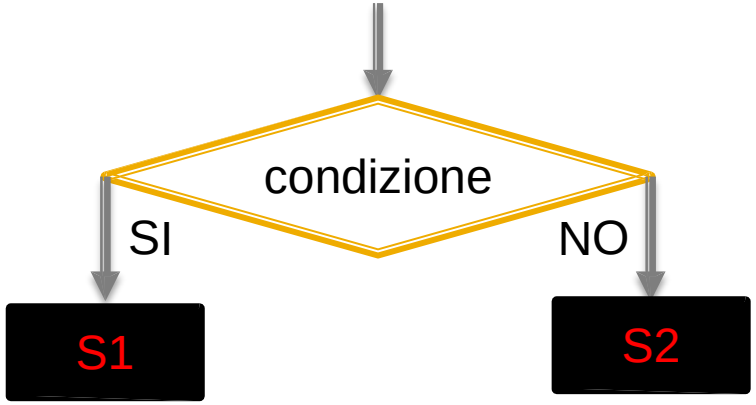


- Un blocco può contenerne altri

- Incolonnamento
  - Evidenzia la presenza di blocchi innestati
    - Migliora la leggibilità del codice

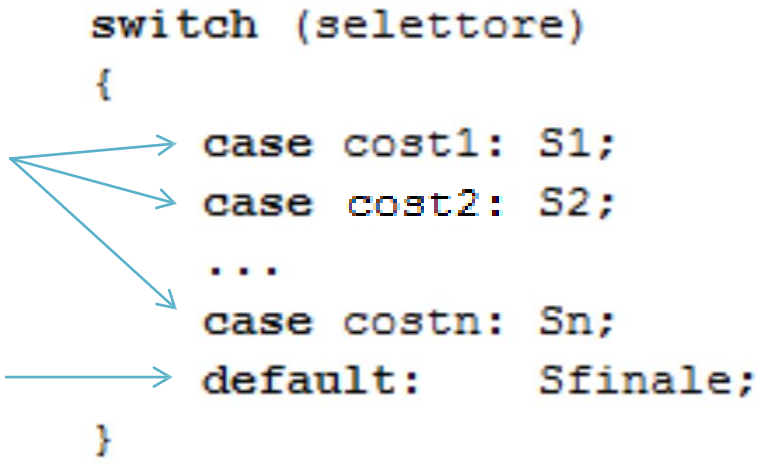
```
{
    Blocco1-I1;
    Blocco1-I2;
    {
        Blocco2-I1;
        Blocco2-I2;
        {
            Blocco3-I1;
            Blocco3-I2;
        }
        Blocco2-I3;
    }
    Blocco1-I3;
}
```

- Con `if … else` si effettua la scelta tra due blocchi di istruzioni
- La condizione è un'espressione di tipo logico
- Il ramo `else` può non esistere




```
if (condizione)
{
   S1;
}
else
{
   S2;
}
```

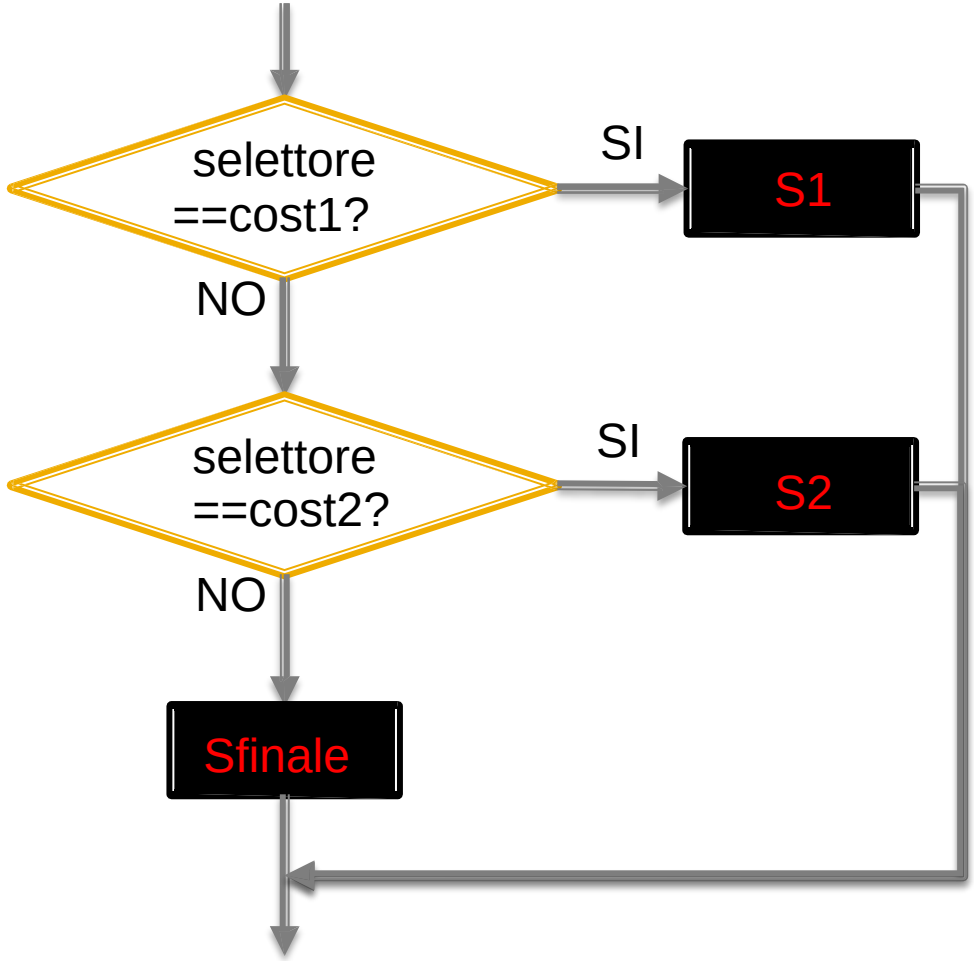
- Con il costrutto `switch` si sceglie l'istruzione di inizio della esecuzione in una sequenza di istruzioni
- La scelta avviene:
  - Calcolando il valore dell'espressione *selettore*
  - Confrontando tale valore con una serie di valori costanti indicati con la parola chiave `case`
- Può comprendere una condizione finale di `default`
  - viene eseguita quando il valore del selettore è diverso dalle costanti riportate nelle frasi case



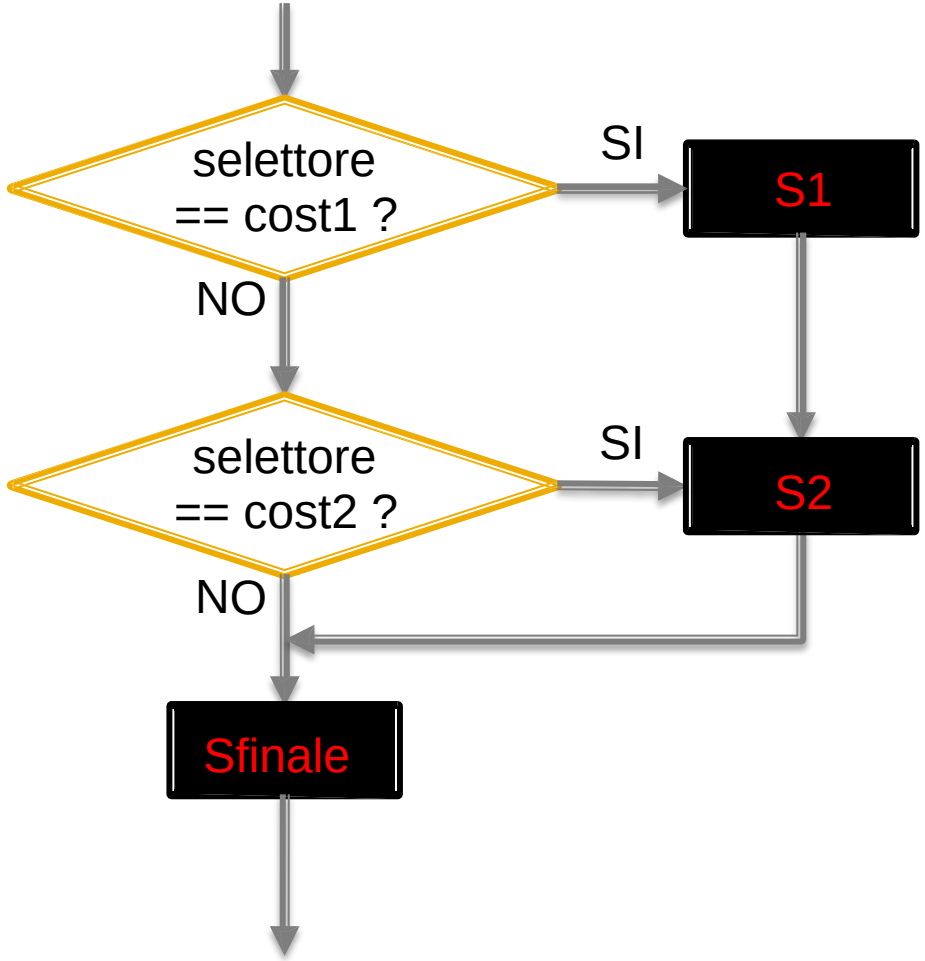
- La struttura `switch` diventa un <u>selettore di blocchi</u> se
  - tutte le sequenze etichettate con le frasi `case` si concludono con l'istruzione `break`
    - che produce come effetto l'uscita dallo switch

```
switch (selettore)
{
   case cost1: S1;
               break;
   case cost2: S2;
               break;
   ...
   case costn: Sn;
               break;
   default:    Sfinale;
}
```

```
switch (selettore)
{
    case cost1: S1;
                break;
    case cost2: S2;
                break;
    default:    Sfinale;
}
```



```
switch (selettore)
{
    case cost1: S1;
                break;
    case cost2: S2;
                break;
    default:    Sfinale;
}
```



- Impone che l'esecuzione del blocco di istruzioni sia ripetuta fino a quando la condizione non diventa FALSE

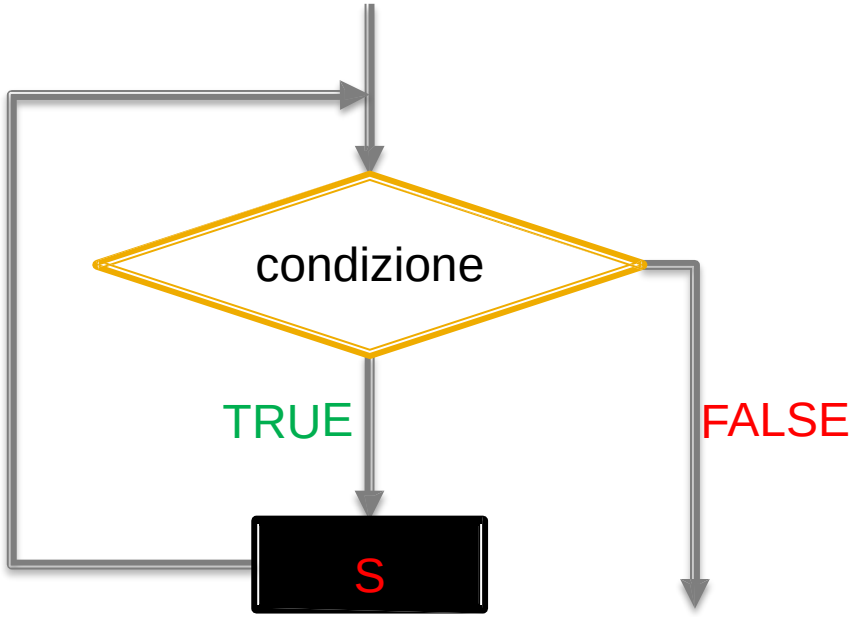
  - Viene calcolata la *condizione*
    - Se FALSE la sequenza S non viene eseguita
    - Se TRUE si esegue S
      - al suo termine si ricalcola la condizione e si riesegue S se è ancora TRUE
        - Si continua fino a che la condizione non diventa FALSE

Struttura iterativa
A CONTROLLO INIZIALE

Numero di esecuzioni:
[0,n]

```
while (condizione)
{
    S;
}
```

```
while (condizione)
{
    S;
}
```



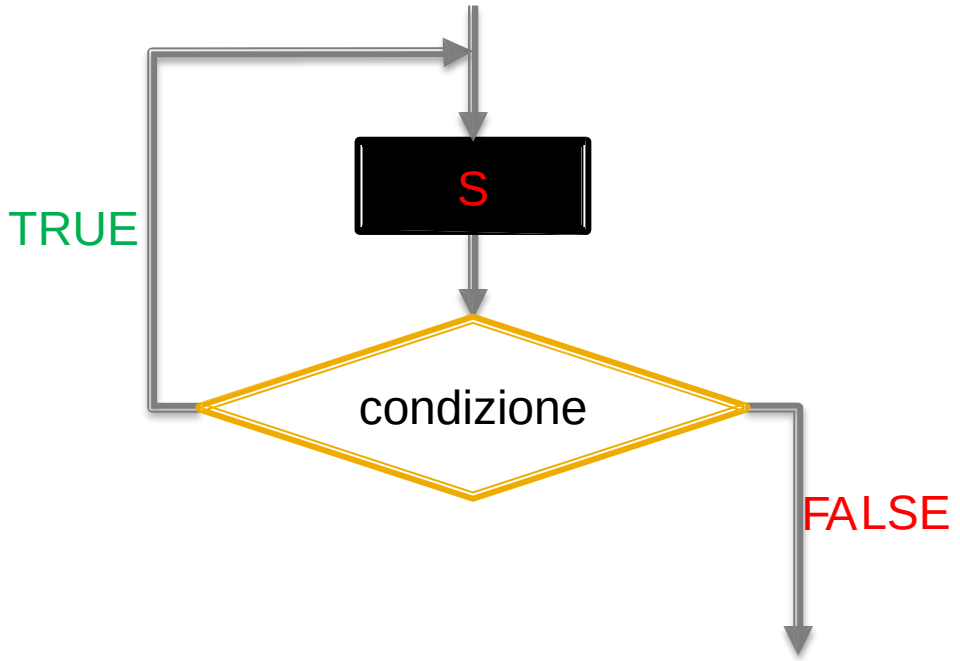
- Come il while, ma la condizione viene scritta dopo la sequenza S
  - L'esecuzione del blocco avviene almeno una volta
  - Si esegue la sequenza S
  - Si calcola la *condizione*
    - Se TRUE si riesegue S
      - Si continua fino a che la condizione non diventa FALSE

Struttura iterativa
A CONTROLLO FINALE

Numero di esecuzioni:
[1,n]

```
do
{
    S;
}
while (condizione)
```

```
do
{
    S;
}
while (condizione)
```



- Il ciclo `for` consente di esprimere
  - una o più istruzioni di inizializzazione per il ciclo
  - una o più istruzioni di variazione delle condizioni di iterazione

```
for (inizializzazioni; condizione; variazioni)
{
    S;
}
```

- ciclo `for` prescrive
  - L'esecuzione delle istruzioni di inizializzazione
  - Il calcolo della condizione
  - L'esecuzione della sequenza S se è vera la condizione
    - In caso contrario l'esecuzione termina
  - L'esecuzione delle istruzioni di variazione al termine di S
  - La rivalutazione della condizione con il ripetersi dei passi precedenti fino alla determinazione della falsità della condizione

```
for (inizializzazioni; condizione; variazioni)
{
    S;
}
```

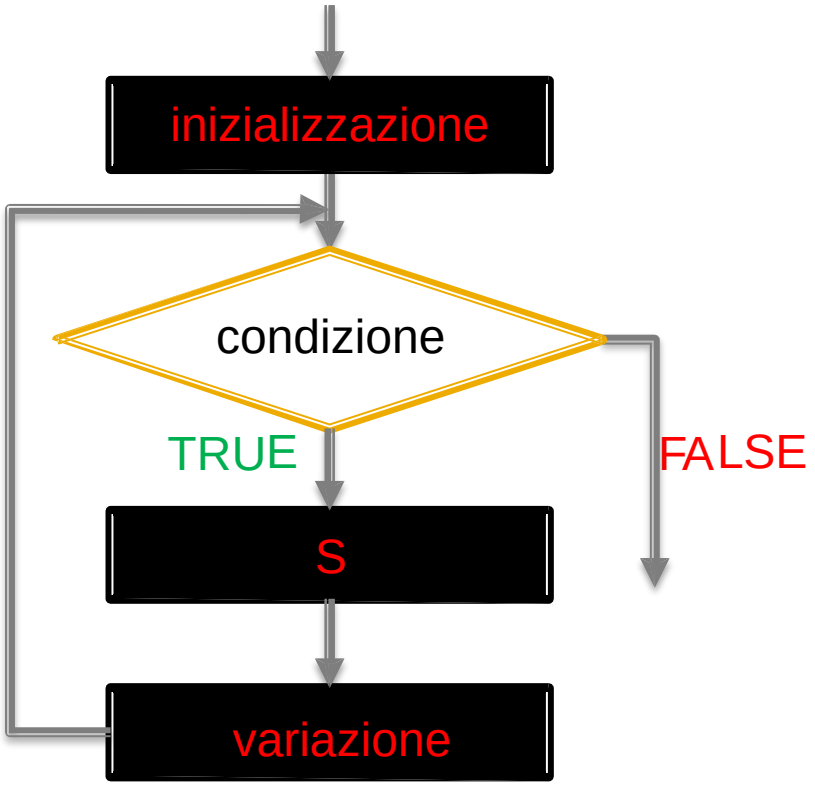
- Le istruzioni di inizializzazione e di variazione non sono obbligatorie
  - mentre la condizione è obbligatoria

- Se inizializzazione e variazione vengono espresse, i cicli `for` e `while` non sono equivalenti

```
for (inizializzazioni; condizione; variazioni)
{
    S;
}
```

- In generale, i cicli for vengono utilizzati quando il numero delle operazioni richieste è ben definito

```
for (inizializzazioni; condizione; variazioni)
{
    S;
}
```



- Struttura di controllo **innestata** in
  - Contenuta totalmente un'altra
- Struttura di controllo **in sequenza**
  - Il suo punto di ingresso è collegato al punto di uscita di un'altra struttura

```
if (condizioneA)
{
   if (condizioneB)
      istruzione1;
   else
      istruzione2;
}
else
   istruzione3;
```

```
if (condizioneA)
   istruzione1;
else
   istruzione2;
if (condizioneB)
   istruzione3;
else
   istruzione4;
```

- Le istruzioni non strutturate sono istruzioni di salto che possono violare i principi della programmazione strutturata
  - Sono da evitare

  - `goto`
    - Provoca il trasferimento incondizionato del flusso di controllo del programma all'istruzione identificata dall'etichetta <label>

```
goto <label>;
label: istruzione;
```

  - `break`
    - Comporta l'uscita da `while`, `do-while`, `switch`, `for`
    - E' normalmente usata solo per lo `switch`

  - `continue`
    - Può essere usata nel
      - `while` e `do-while`, in cui è equivalente al salto alla verifica della condizione
      - `for`, in cui è equivalente al salto alla variazione delle condizioni del ciclo

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

if-then-else

- if (a >b)
- r = a + b;
- else
- r = a – b
- …
- [Altre Istruzioni]

Verifica Condizione

Condizione Vera

Condizione Falsa

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Espressioni logiche

- Operatori di confronto

Simbolo	Significato	Utilizzo
==	uguale a	a == b
!=	diverso da	a != b
<	minore	a < b
>	maggiore	a > b
<=	minore o uguale	a <= b
>=	maggiore o uguale	a >= b

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Operatori logici

Simbolo	Significato	Utilizzo
&&	AND logico	a && b
||	OR logico	a || b

AND logico	true	false
true	***true***	***false***
false	***false***	***false***

OR logico	true	false
true	***true***	***true***
false	***true***	***false***

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Teorema di De Morgan

- Il negato di un espressione booleana si ottiene:
- negando tutti I termini
- Sostituendo ogni operatore logico con il suo duale

- Es:
- !((a > 0) &&(b >=0))
- (a <=0) || (b<0))

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

If annidati

- L'**else** si riferisce sempre all'ultimo **if** !!!!

- if(temperatura < 20)
- if(temperatura < 10) printf("Metti il cappotto!n");
- else printf(" Basta mettere una felpa");

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Parentesi e formattazione

```
if(temperatura < 20)
{
  if(temperatura < 10)
  {
    printf("Metti il cappotto!");
  }
  else
  {
    printf(" Basta mettere una felpa");
  }
}
```

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Solo formattazione

```
if(temperatura < 20)
  if(temperatura < 10) printf("Metti il cappotto!n");
  else printf(" Basta mettere una felpa");
```

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Calcolare il massimo tra tre numeri

- Esercizio …

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

While

```
int main()
{
    int a;
    a = 5;
    while (a>0)
        a = a -1;
    printf ("a=%d",a);
}
```

Condizione Vera

```
int main()
{
    int a;
    a = 5;
    while (a>0)
        {
            a = a -1;
        }
    printf ("a=%d",a);
}
```

Costrutto Iterativo

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Realizzare il Diagramma di Flusso dell'Algoritmo per il calcolo del Massimo Comun Divisore

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Massimo Comun Divisore

Teorema di Euclide:

"ogni divisore comune di a e b è divisore di a, b e del resto r della divisione tra a e b (a *mod* b), se questo non è nullo"

Soluzione di Euclide - Costruzione dell'algoritmo

Dati due numeri ***a***,***b*** assumiamo che **a** sia sempre il maggiore

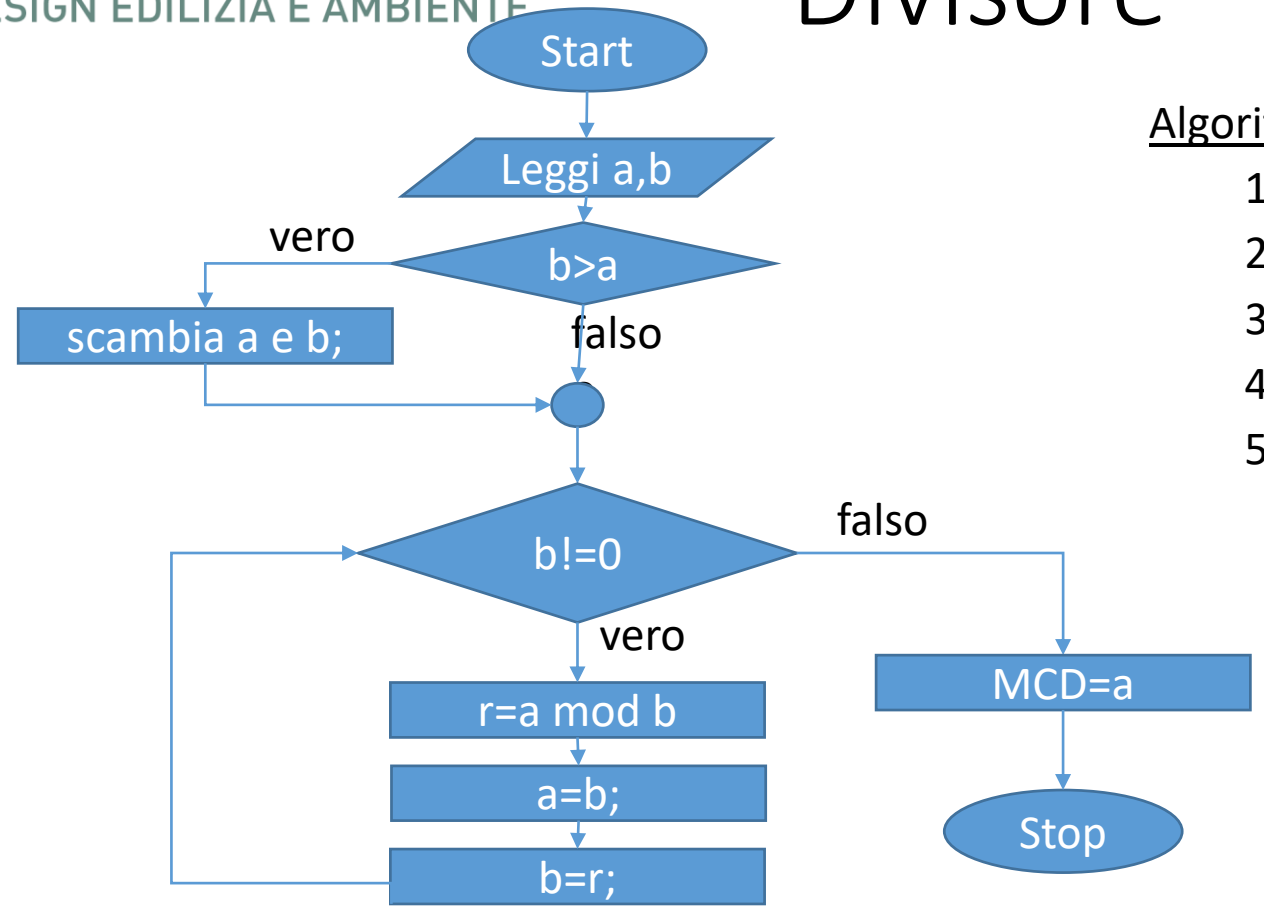
Effettuiamo l'operazione r= a *mod* b se r=0 allor b è MCD di a

Altrimenti calcoliamo r'= b mod r, se r'=0 allora r è divisore di b ed è MCD di a (non lo era b)

Possiamo quindi procedere così fino a trovare il caso in cui l'operazione dia zero.

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Massimo Comun Divisore



Algoritmo:

1. acquisire due numeri a,b
2. se b>a scambiare a con b
3. se b= 0 MCD(a,b)=a e termina
4. r= a *mod* b
5. sostituire a con b, b con r ed andare al passo 3

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

Algoritmo

- Problema: Calcolo del Massimo Comun Divisore tra due numeri a,b : MCD(a,b)
- Soluzione di Euclide: “ogni divisore comune di a e b è divisore di a, b e del resto r della divisione tra a e b (a *mod* b), se questo non è nullo”
- Algoritmo:
- 1. acquisire due numeri a,b
- 2. se b>a scambiare a con b
- 3. se b= 0 MCD(a,b)=a a andare al passo 6
- 4. r= a *mod* b
- 5. sostituire a con b, b con r ed andare al passo 3
- 6. Fine

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE

MCD Soluzione

```
#include <stdio.h>

int main()
{
int a=24;
int b=20;
int temp;
int mcd;
int r;
```

```
if (b>a) {  //scambio di 2 variabili
      temp=a;
      a=b;
      b=temp;
     }
while (b!=0) {
      r = a % b;  //operazione modulo
        a=b;
        b=r;

    if (b==0)
       mcd=a;

}// end while
```

```
printf("mcd = %d",mcd);

}
```

- Esercizio 1: Problema

  Calcolare area e perimetro di una figura geometrica fornita in input. Le possibili figure geometriche sono cerchio, triangolo, rettangolo e quadrato

- Esercizio 1: Analisi

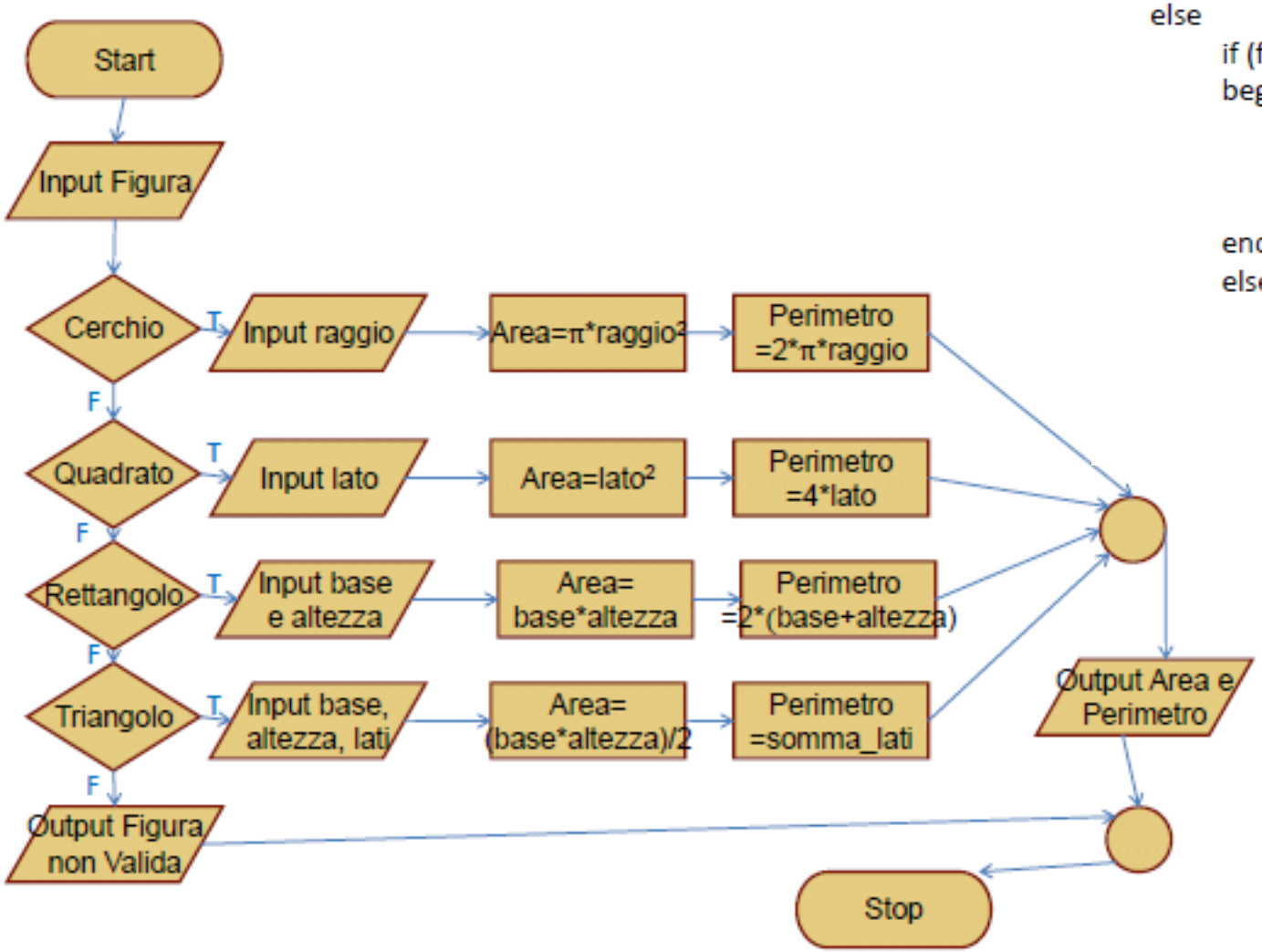
  Il problema del calcolo di area e perimetro delle quattro figure geometriche può essere decomposto in 4 sottoproblemi, ciascuno finalizzato al calcolo di area e perimetro di ognuna delle figure

  – Ciascuno dei 4 sottoproblemi può ulteriormente essere decomposto in 2 sotto-sottoproblemi:

  - calcolo del perimetro e calcolo dell'area

Calcolo di perimetro e area è ulteriormente decomposto in problemi elementari (controllo consistenza dati in input e calcoli aritmetici elementari)

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE




```
Begin
    leggi figura
    if (figura == cerchio)
    begin
        leggi raggio;
        area=π*raggio²;
        perimetro= 2*π*raggio;
    end
    else
        if (figura == quadrato)
        begin
            leggi lato;
            area=lato²;
            perimetro= 4*lato;
        end
        else
            if (figura == rettangolo)
            begin
                leggi base;
                leggi h;
                area=base*altezza;
                perimetro= 2*(base+h);
            end
            else
                if (figura == triangolo)
                begin
                    leggi b, h;
                    leggi l1, l2, l3;
                    area=(b*h)/2;
                    perimetro= l1+ l2 + l3;
                end
                else
                    scrivi "Figura Non Valida";
        scrivi area;
        scrivi perimetro;
    End
```

- Esercizio 2: Problema

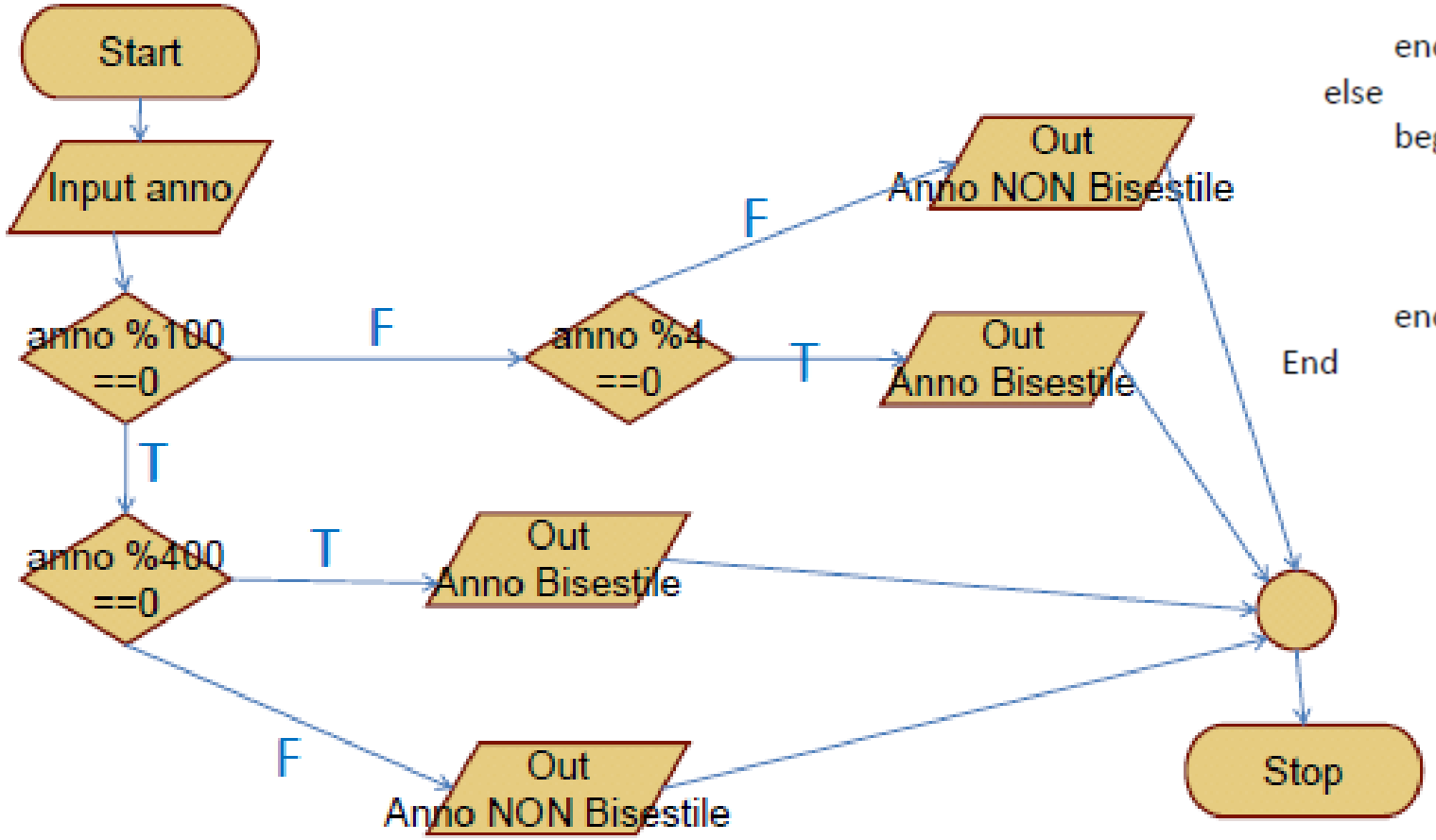
  Decidere se un anno è bisestile

  sono bisestili:

  – gli anni non secolari multipli di 4
  – gli anni secolari multipli di 400

- Esercizio 2: Analisi
- Il problema può essere decomposto in due sottoproblemi
  - Analisi degli anni non secolari
  - Analisi degli anni secolari
- In entrambi i casi l'analisi avviene applicando istruzioni elementari, rispettivamente
  - Resto della divisione dell'anno per 4
  - Resto della divisione dell'anno per 400
- Limitiamo il calcolo agli anni d.c.
  - I valori che esprimono gli anni sono solo interi >0

UNIVERSITÀ DEGLI STUDI DELLA CAMPANIA
LUIGI VANVITELLI
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA CIVILE
DESIGN EDILIZIA E AMBIENTE




```
Begin
    input anno
    if ((anno % 100)==0)
    then
        begin
            if ((anno%400)==0)
                then output “Anno Bisestile”;
                else output “Anno NON Bisestile”;
        end
    else
        begin
            if ((anno%4)==0)
                then output “Anno Bisestile”;
                else output “Anno NON Bisestile”;
        end
End
```