Elementi Di Informatica E Programmazione

Prof. Andrea Loreggia

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

Sommario

UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

- Analisi e programmazione
- Gli algoritmi
  - Proprietà ed esempi
- I linguaggi per la formalizzazione di algoritmi
  - Diagrammi a blocchi
  - Pseudocodifica
- Gli algoritmi ricorsivi

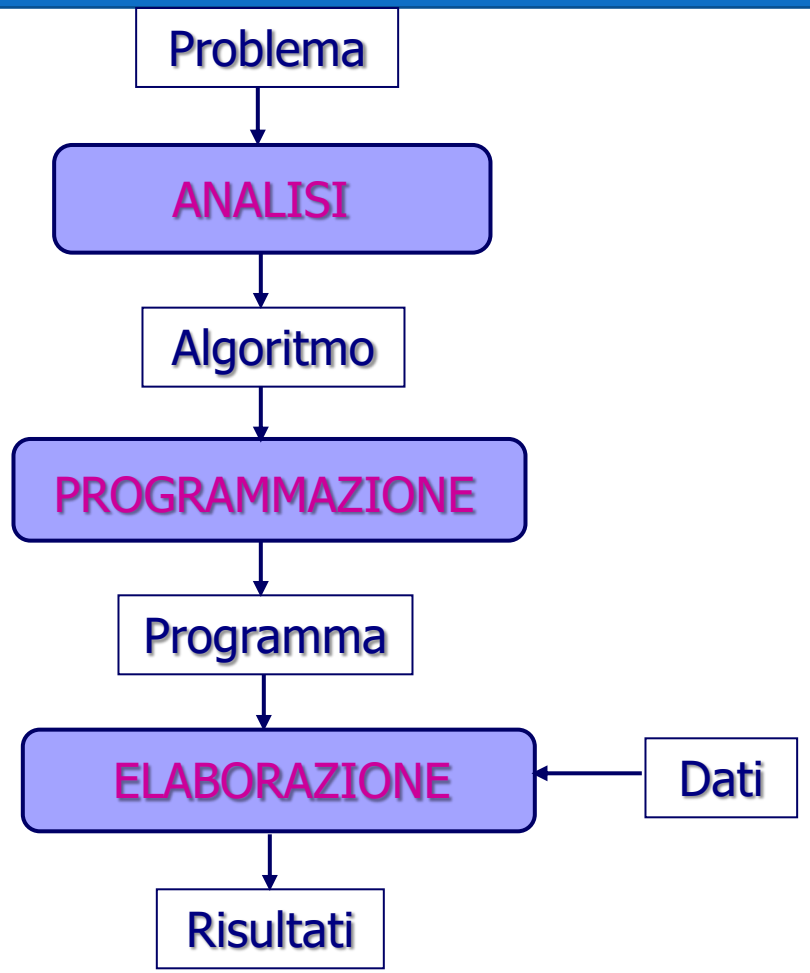
Analisi e programmazione

- Tramite un elaboratore si possono risolvere problemi di varia natura: emissione di certificati anagrafici, gestione dei c/c di un istituto di credito, prenotazioni aeree…
- Il problema deve essere formulato in modo opportuno, perché sia possibile utilizzare un elaboratore per la sua soluzione
- Per analisi e programmazione si intende l'insieme delle attività preliminari atte a risolvere problemi utilizzando un elaboratore, dalla formulazione del problema fino alla predisposizione dell'elaboratore
  - Scopo dell'analisi ⇨ definire un algoritmo
  - Scopo della programmazione ⇨ definire un programma

Analisi e programmazione

- Algoritmo: elenco finito di istruzioni, che specificano le operazioni eseguendo le quali si risolve una classe di problemi
  - Un particolare problema della classe viene risolto tramite l'apposito algoritmo sui dati che lo caratterizzano
  - Un algoritmo non può essere eseguito direttamente dall'elaboratore
- Programma: ricetta che traduce l'algoritmo ed è direttamente comprensibile, pertanto eseguibile, da parte di un elaboratore
- Linguaggio di programmazione: linguaggio rigoroso che permette la formalizzazione di un algoritmo in un programma

Analisi e programmazione: Le Fasi



Definizione di algoritmo

UNIVERSITÀ DEGLI STUDI DI BRESCIA

- Algoritmo deriva dal nome del matematico uzbeco Mohammed ibn–Musa Al–Khuwarizmi, vissuto nel IX secolo d.C. (dalla cui opera è nata l'algebra moderna), e significa procedimento di calcolo
- Un algoritmo è una successione di istruzioni o passi che definiscono le operazioni da eseguire sui dati per ottenere i risultati; un algoritmo fornisce la soluzione ad una classe di problemi
- Lo schema di esecuzione di un algoritmo specifica che i passi devono essere eseguiti in sequenza, salvo diversa ed esplicita indicazione

Proprietà degli algoritmi

- Affinché una ricetta, un elenco di istruzioni, possa essere considerato un algoritmo, devono essere soddisfatti i seguenti requisiti:
  - **Finitezza**: ogni algoritmo deve essere finito, cioè ogni singola istruzione deve poter essere eseguita in tempo finito ed un numero finito di volte
  - **Generalità**: ogni algoritmo deve fornire la soluzione per una classe di problemi; deve pertanto essere applicabile a qualsiasi insieme di dati appartenenti all'insieme di definizione o dominio dell'algoritmo e deve produrre risultati che appartengano all'insieme di arrivo o codominio
  - **Non ambiguità**: devono essere definiti in modo univoco i passi successivi da eseguire; devono essere evitati paradossi, contraddizioni ed ambiguità; il significato di ogni istruzione deve essere univoco per chiunque esegua l'algoritmo

Algoritmi

- I linguaggi naturali non soddisfano questi requisiti, infatti...
  - …sono ambigui: la stessa parola può assumere significati diversi in contesti differenti (pesca è un frutto o un'attività sportiva; ancora: ambiti, principi, vera, venti, avanzato, mora, amare…)
  - …sono ridondanti: lo stesso concetto può essere espresso in molti modi diversi, ad esempio "somma 2 a 3", "calcola 2+3", "esegui l'addizione tra 2 e 3"

Algoritmi, Linguaggi ed Esecutori

- Gli Algoritmi sono espressi attraverso un ***Linguaggio*** che permette di descrivere i passi che compongono l'algoritmo.

- Un ***Esecutore*** effettua I passi dell'algoritmo per produrre I risultati attesi.

Linguaggi

Rappresentazione di un algoritmo:

- Linguaggi di programmazione
  - Linguaggio di programmazione C, Python, C++, Java
  - Linguaggio Assembler
  - Linguaggio macchina
- Diagrammi di flusso
- Pseudo codice
- Linguaggi visuali

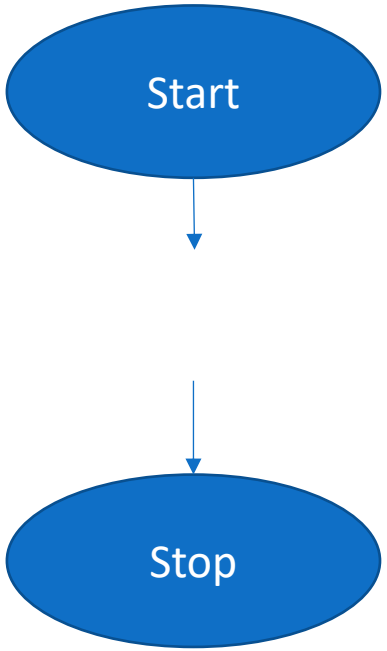
Chi è l'esecutore?

Diagrammi di Flusso

- I ***Diagrammi di flusso*** sono un *linguaggio grafico* per la rappresentazione di un algoritmo
  - Il diagramma è composto da blocchi che rappresentano i passi dell'algoritmo
  - I blocchi hanno forme diverse in base al significato
  - I blocchi contengono testo che dscrive il passo da eseguire
- ***CHI è l'esecutore?***
  - Utile per descrivere un algoritmo più che per eseguirlo…

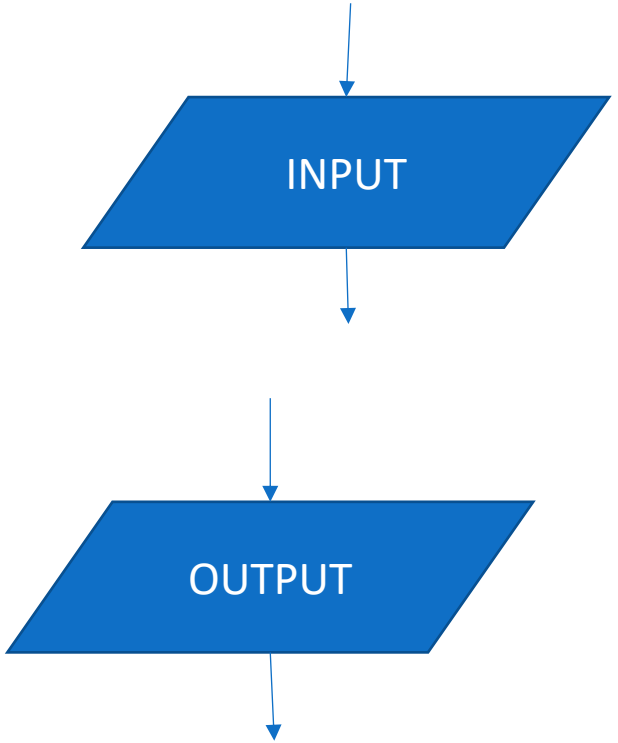
Diagrammi di Flusso

- Blocchi ***Start*** e ***Stop*** vengono utilizzati per indicare da dove il diagramma inizia e quando il diagramma termina.



Diagrammi di Flusso

- I Blocchi ***Input*** e ***Output*** vengono utilizzati per indicare la richiesta di dati all'utente



Diagrammi di Flusso

- I Blocchi ***Azione*** vengono utilizzati per indicare una operazione di elaborazione elementare

Diagrammi di Flusso

- I Blocchi ***Condizione*** vengono utilizzati per indicare una operazione di scelta
- Ogni ramo di uscita rappresenta una scelta differente

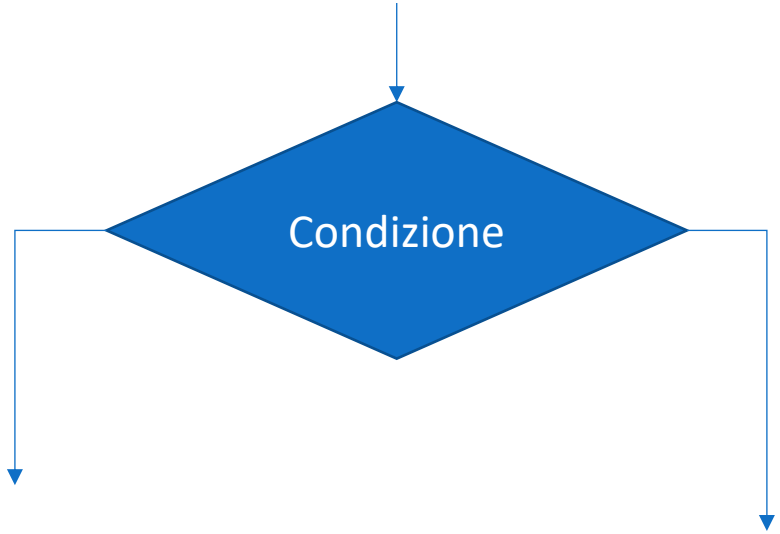


Diagramma di Flusso: Somma di due numeri

UNIVERSITÀ DEGLI STUDI DI BRESCIA

Passo 1: Leggi A

Passo 2: Leggi B

Passo 3: Somma A e B

Passo 4: Stampa il risultato

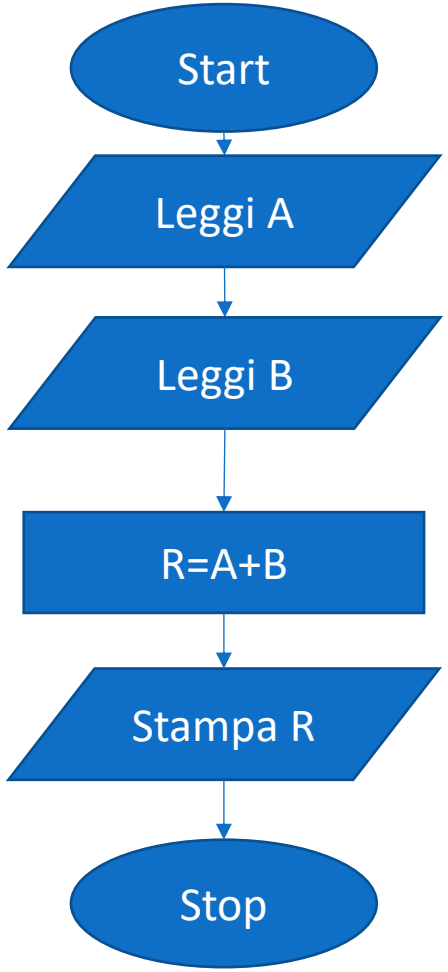


Diagramma di Flusso: Stampa il maggiore di due numeri

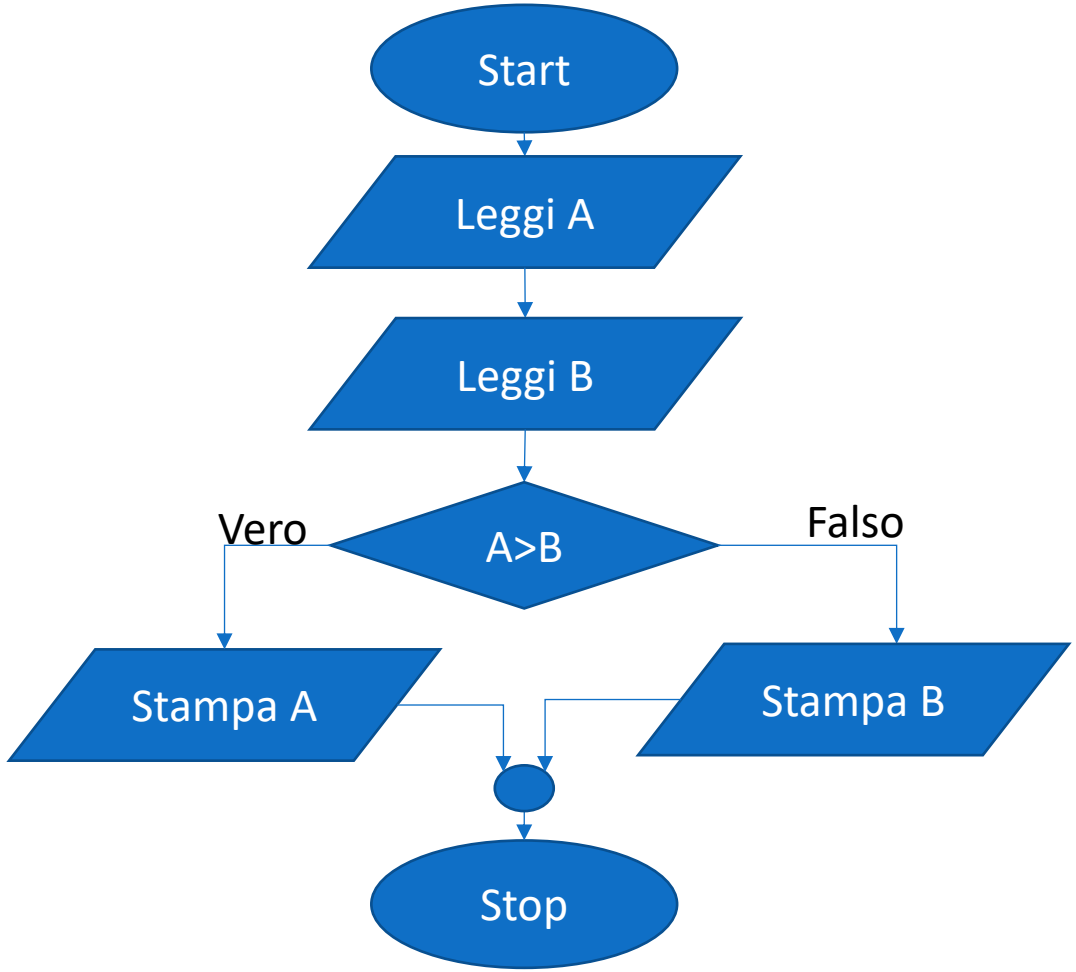
UNIVERSITÀ DEGLI STUDI DI BRESCIA

Passo 1: Leggi A

Passo 2: Leggi B

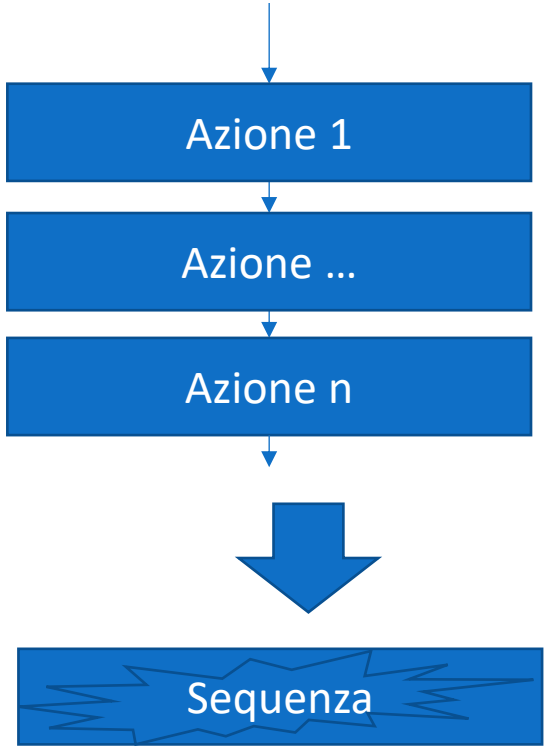
Passo 3: confronta A e B

Passo 4: Se A>B stampa A, altrimenti stampa B



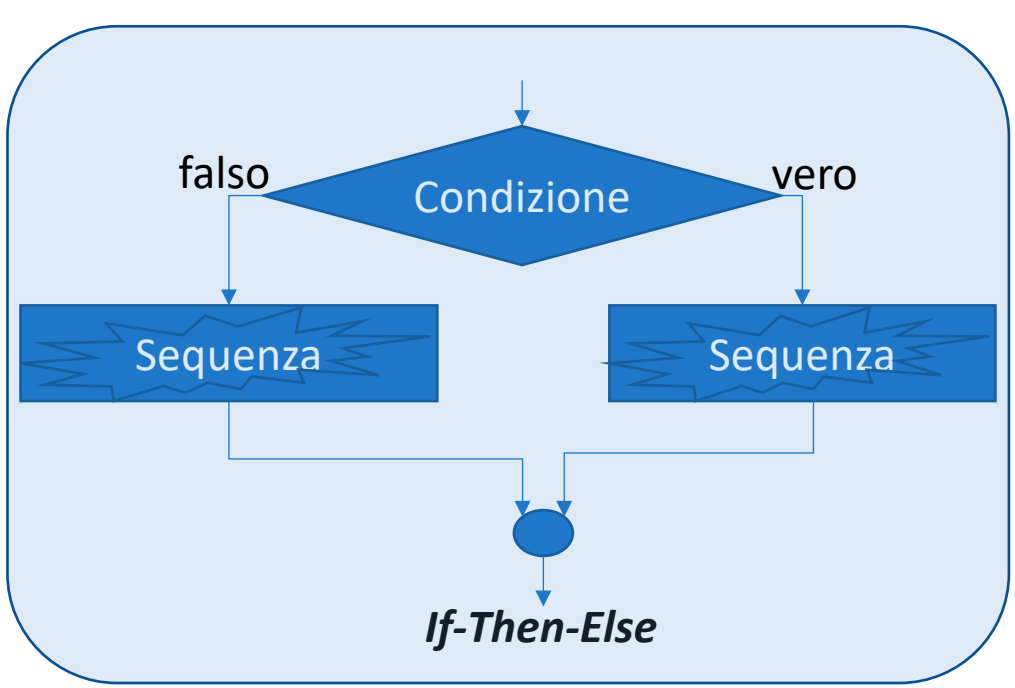
Diagrammi di Flusso: Strutture Avanzate

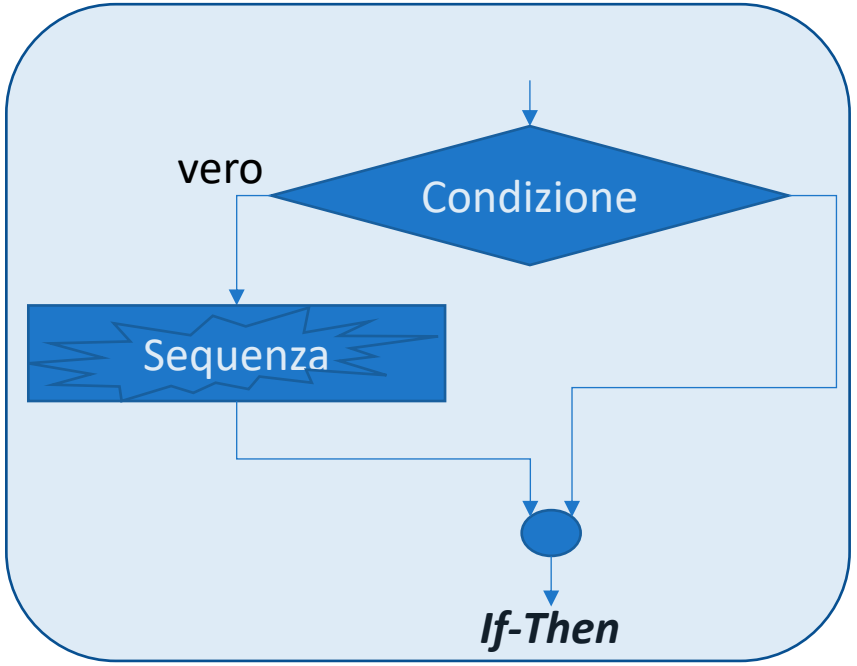
- Il blocco ***sequenza*** viene utilizzato per indicare l'esecuzione consecutiva di blocchi.



Diagrammi di Flusso: Strutture Avanzate

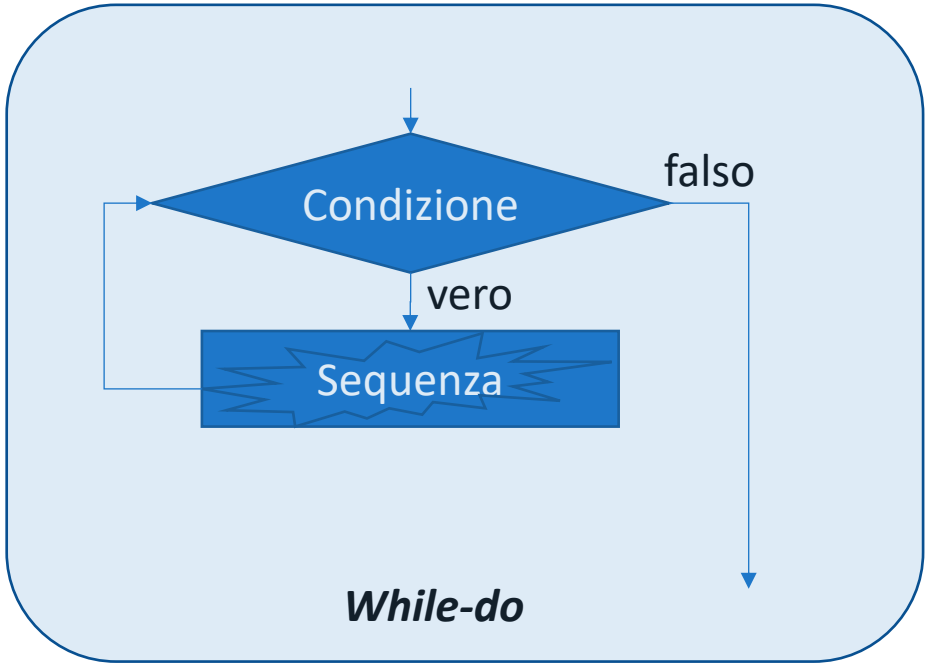
- I blocchi ***If-Then*** e ***IF-Then-Else*** vengono utilizzati per I costrutti di selezione





Diagrammi di Flusso: Strutture Avanzate

- I blocchi ***while-do*** e ***repeat-until*** vengono utilizzati per le operazioni cicliche



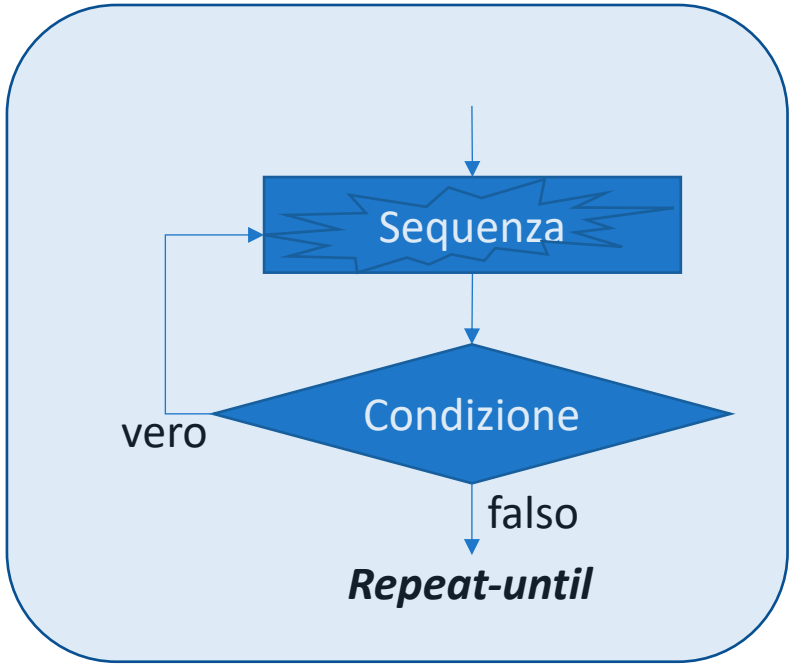


Diagramma di Flusso non strutturato

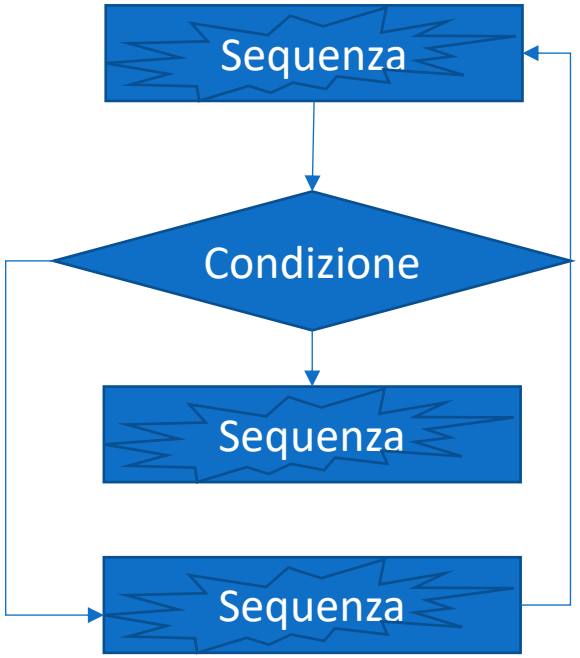
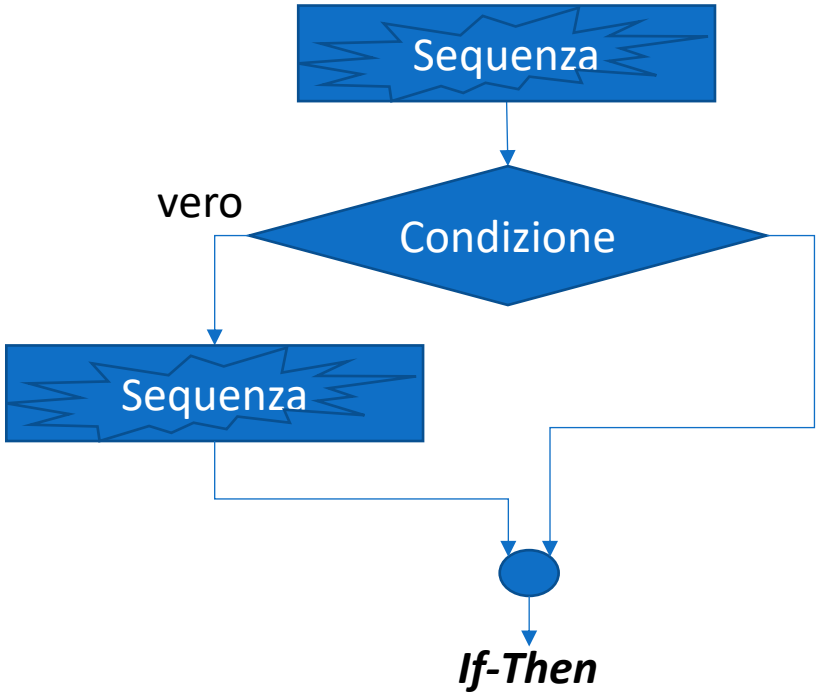


Diagramma di Flusso strutturato



Diagrammi di Flusso: Strutture Avanzate

Diagramma di Flusso non strutturato

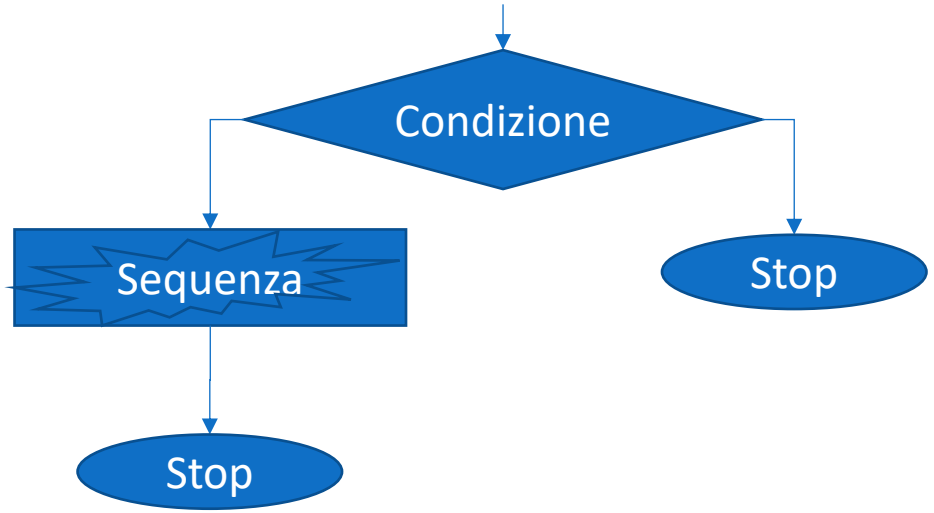
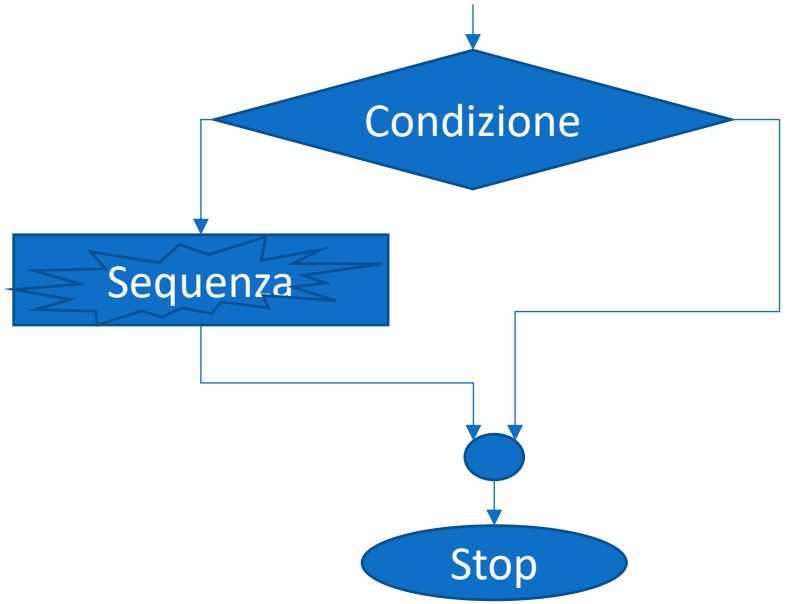


Diagramma di Flusso strutturato



Teorema di Böhm-Jacopini

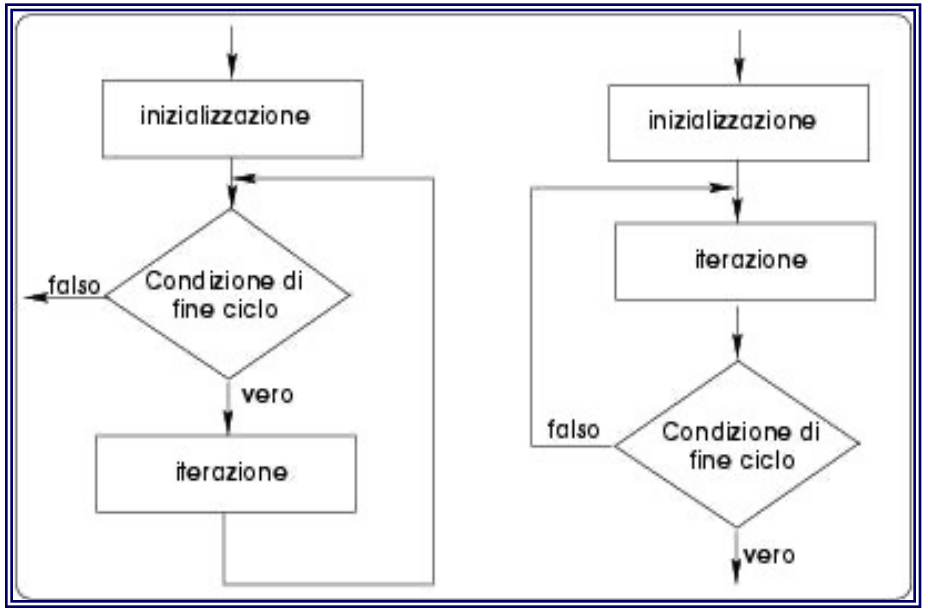
Qualunque algoritmo può essere implementato utilizzando tre sole strutture: la sequenza, la selezione e l'iterazione.

- In altre parole è possibile realizzare ogni algoritmo utilizzando solo le strutture avanzate
- I Diagrammi strutturati hanno il vantaggio di essere molto più intuitivi e comprensibili.

Gli algoritmi iterativi

- Il ciclo o *loop* è uno schema di flusso per descrivere, in modo conciso, situazioni in cui un gruppo di operazioni deve essere ripetuto più volte
- La condizione di fine ciclo viene verificata ogni volta che si esegue il ciclo; se la condizione assume valore vero (falso), le istruzioni vengono reiterate, altrimenti si esce dal ciclo
- La condizione di fine ciclo può essere verificata prima o dopo l'esecuzione dell'iterazione
- Le istruzioni di inizializzazione assegnano valori iniziali ad alcune variabili (almeno a quella che controlla la condizione di fine ciclo)

Ciclo con controllo in testa



Ciclo con controllo in coda

Gli algoritmi iterativi

- Un ciclo è definito quando è noto a priori quante volte deve essere eseguito; un ciclo definito è detto anche enumerativo
- Un contatore del ciclo tiene memoria di quante iterazioni sono state effettuate; può essere utilizzato in due modi:
  - incremento del contatore: il contatore viene inizializzato ad un valore minimo (ad es. 0 o 1) e incrementato ad ogni esecuzione del ciclo; si esce dal ciclo quando il valore del contatore eguaglia il numero di iterazioni richieste
  - decremento del contatore: il contatore viene inizializzato al numero di iterazioni richiesto e decrementato di uno ad ogni iterazione; si esce dal ciclo quando il valore del contatore raggiunge 0 (o 1)

Gli algoritmi iterativi

- Un ciclo è indefinito quando non è possibile conoscere a priori quante volte verrà eseguito
- La condizione di fine ciclo controlla il valore di una o più variabili modificate da istruzioni che fanno parte dell'iterazione
- Comunque, un ciclo deve essere eseguito un numero finito di volte, cioè si deve sempre verificare la terminazione dell'esecuzione del ciclo

La pseudocodifica

- La pseudocodifica è un linguaggio per la descrizione di algoritmi secondo le regole della programmazione strutturata
- La descrizione di un algoritmo in pseudocodifica si compone di due parti...
  - la dichiarazione delle variabili usate nell'algoritmo
  - la descrizione delle azioni dell'algoritmo

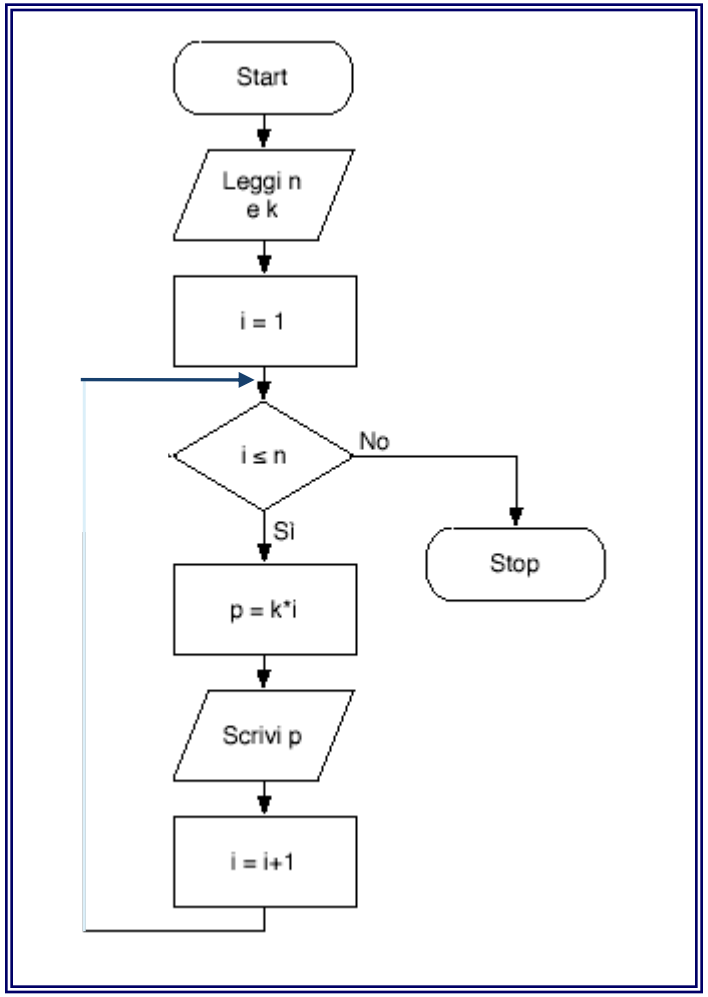
La pseudocodifica

- Esempio: Algoritmo per il calcolo delle radici di equazioni di 2° grado

```
var x1, x2, a, b, c, delta: real.
begin
    read a, b, c;
    delta ← b²–4ac;
    if  delta<0
        then write "non esistono radici reali"
        else if delta=0
            then x1 ← –b/2a;
                 x2 ← x1
            else  x1 ← (–b + √delta)/2a;
                  x2 ← (–b – √delta)/2a
            endif
            write x1, x2
    endif
end
```

Un esempio "comparativo"

- Letti due interi n e k, entrambi maggiori di zero, stampare i primi n multipli di k




```
var  i, n, k, p: integer.
begin
    read n;
    read k;
    for i from 1 to n do
        p ← k×i;
        write p
    endfor
end
```

```
i ← 1;
while i ≤ n
    p ← k×i;
    write p;
    i ← i+1;
endwhile
```

```
#include <stdio.h>
main()
{
   int i,n,k,p;
   scanf("%d",&n);
   scanf("%d",&k);
   for(i=1;i<=n;i++)
     {
       p = k*i;
       printf("%d",p);
     }
    exit(0);
}
```

Considerazioni finali

- Attenzione alla scelta di un “buon” algoritmo…
  - Due algoritmi si dicono equivalenti quando:
    - hanno lo stesso dominio di ingresso
    - hanno lo stesso dominio di uscita
    - in corrispondenza degli stessi valori nel dominio di ingresso producono gli stessi valori nel dominio di uscita
  - Due algoritmi equivalenti forniscono lo stesso risultato, ma possono avere diversa efficienza e possono essere profondamente diversi

Considerazioni finali

UNIVERSITÀ DEGLI STUDI DI BRESCIA

- Un esempio di due algoritmi equivalenti, ma con diversa efficienza, per la moltiplicazione fra interi è…

Algoritmo 1	Algoritmo 2 (somma e shift)
Somme successive: 12x12 = 12+12+…+12=144	12x 12= 24 12= 144

Considerazioni finali

- L'efficienza di un algoritmo si valuta in base alla sua complessità
  - … cioè in base all'analisi delle risorse impiegate dall'algoritmo per risolvere un dato problema, in funzione della dimensione e dal tipo dell'input
  - Risorse:
    - Tempo impiegato per completare l'esecuzione
    - Spazio, ovvero quantità di memoria utilizzata

Esercitazione

Esempio: ordinamento del mazzo di carte

- Problema: Sia dato un mazzo da 40 carte da ordinare in modo che i cuori precedano i quadri, che a loro volta precedono fiori e picche; le carte di uno stesso seme sono ordinate dall'asso al re

Esempio: ordinamento del mazzo di carte

- Problema: Sia dato un mazzo da 40 carte da ordinare in modo che i cuori precedano i quadri, che a loro volta precedono fiori e picche; le carte di uno stesso seme sono ordinate dall'asso al re
- Algoritmo:
  - Si suddivida il mazzo in 4 mazzetti, ciascuno costituito da tutte le carte dello stesso seme
  - Si ordinino le carte di ciascun mazzetto dall'asso al re
  - Si prendano nell'ordine i mazzetti dei cuori, quadri, fiori e picche

Esempio: ordinamento del mazzo di carte

- Problema: Si vuole ricercare, all'interno di un mazzo di chiavi, quella che apre un dato lucchetto

Esempio: ordinamento del mazzo di carte

- Problema: Si vuole ricercare, all'interno di un mazzo di chiavi, quella che apre un dato lucchetto
- Algoritmo:
  1. Si seleziona una chiave dal mazzo
  2. Si marca con un pennarello la chiave e si tenta di aprire il lucchetto con la chiave appena marcata; se funziona, si va al passo 4)
  3. Altrimenti, si controlla la chiave successiva
     1. Se non è marcata, la si marca e si torna al passo 2)
     2. Viceversa, si prende atto che nel mazzo non è presente la chiave che apre il lucchetto
  4. Fine della ricerca

Esempio: radici delle equazioni di 2° grado

- Problema: Calcolo delle radici reali di ax2+bx+c=0

Esempio: radici delle equazioni di 2° grado

- Problema: Calcolo delle radici reali di $ax^2+bx+c=0$
- Algoritmo:
  1. Acquisire i coefficienti a,b,c
  2. Calcolare $\Delta = b^2-4ac$
  3. Se $\Delta<0$ non esistono radici reali, eseguire l'istruzione 7)
  4. Se $\Delta=0$, $x1=x2=-b/2a$, poi eseguire l'istruzione 6)
  5. $x1=(-b+\sqrt{\Delta})/2a$, $x2=(-b-\sqrt{\Delta})/2a$
  6. Comunicare i valori x1, x2
  7. Fine

Esempio: calcolo del M.C.D.

- **Problema**: Calcolare il M.C.D. di due interi a,b, con a>b
- **Algoritmo**: Formalizzato da Euclide nel 300 a.C., si basa sul fatto che ogni divisore comune ad a e b è anche divisore del resto r della divisione intera di a per b, quando a>b e r≠0; se r=0, b è il M.C.D.
  - MCD(a,b) = MCD(b,r), se r≠0
  - MCD(a,b) = b, se r=0
- Nota
- L'algoritmo garantisce la determinazione del M.C.D. senza il calcolo di tutti i divisori di a e b

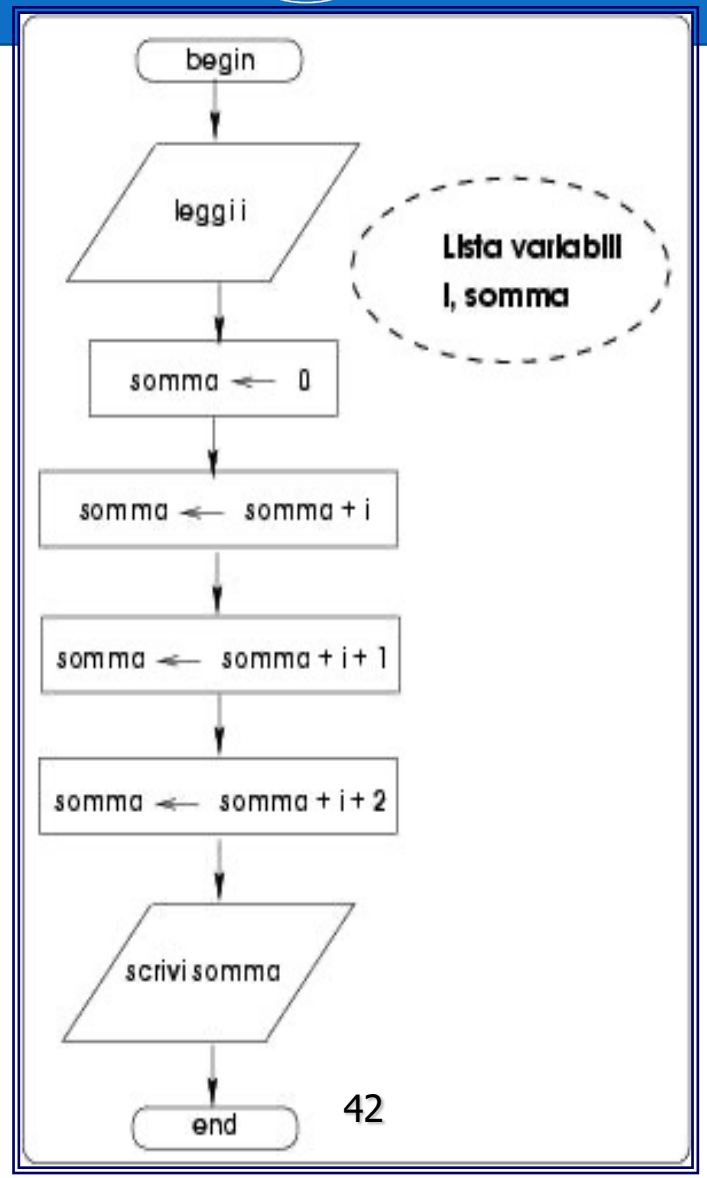
Esempio: calcolo del M.C.D.

- Acquisire i valori di a e b
- Se b>a, scambiare i valori di a e b
- Calcolare il resto r della divisione intera di a per b
- Se r=0, MCD(a,b)=b; comunicare il risultato all'esterno; eseguire l'istruzione 6)
- Se r≠0, sostituire il valore di a con il valore di b ed il valore di b con il valore di r; tornare al passo 3)
- Fine

Gli algoritmi iterativi

Problema: Calcolare la somma di tre interi consecutivi

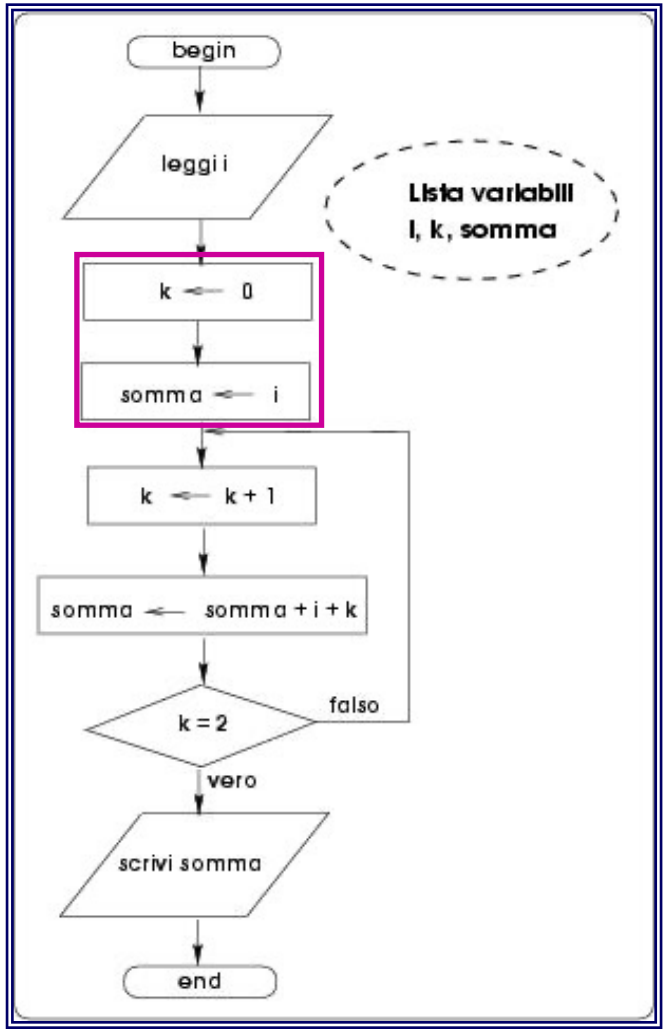
- **Note:**
  - La variabile **somma** è un contenitore di somme parziali, finché non si ottiene la somma totale richiesta
  - La soluzione del problema viene raggiunta eseguendo azioni simili per un numero opportuno di volte



Gli algoritmi iterativi

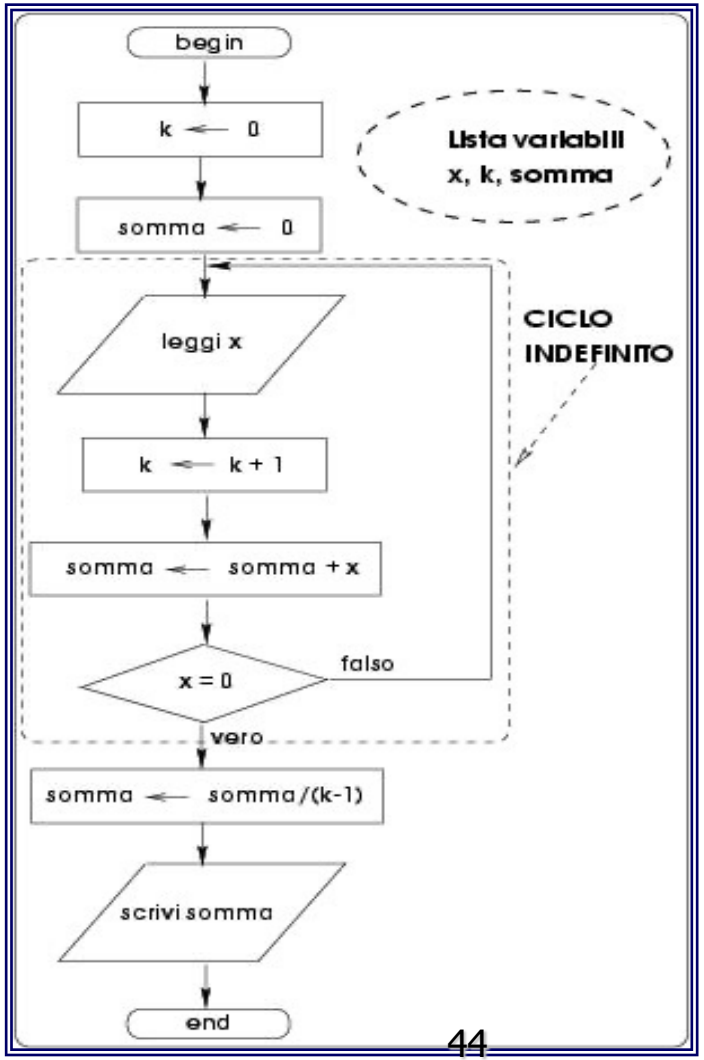
- **Problema:** Calcolare la somma di tre interi consecutivi

- **Note:**
  - La fase di inizializzazione riguarda la somma e l'indice del ciclo
  - Il controllo di fine ciclo viene effettuato in coda (iterazione per falso)



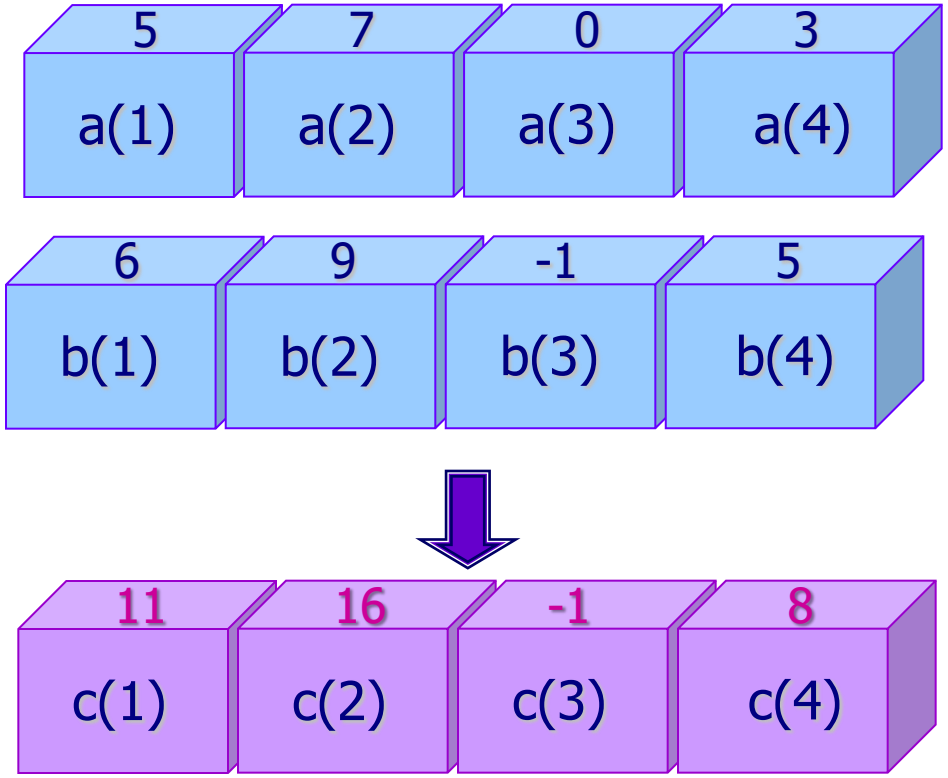
Gli algoritmi iterativi

- Problema: Calcolo della media di un insieme di numeri; non è noto a priori quanti sono i numeri di cui deve essere calcolata la media
  - I numeri vengono letti uno alla volta fino a che non si incontra un x=0, che segnala la fine dell'insieme



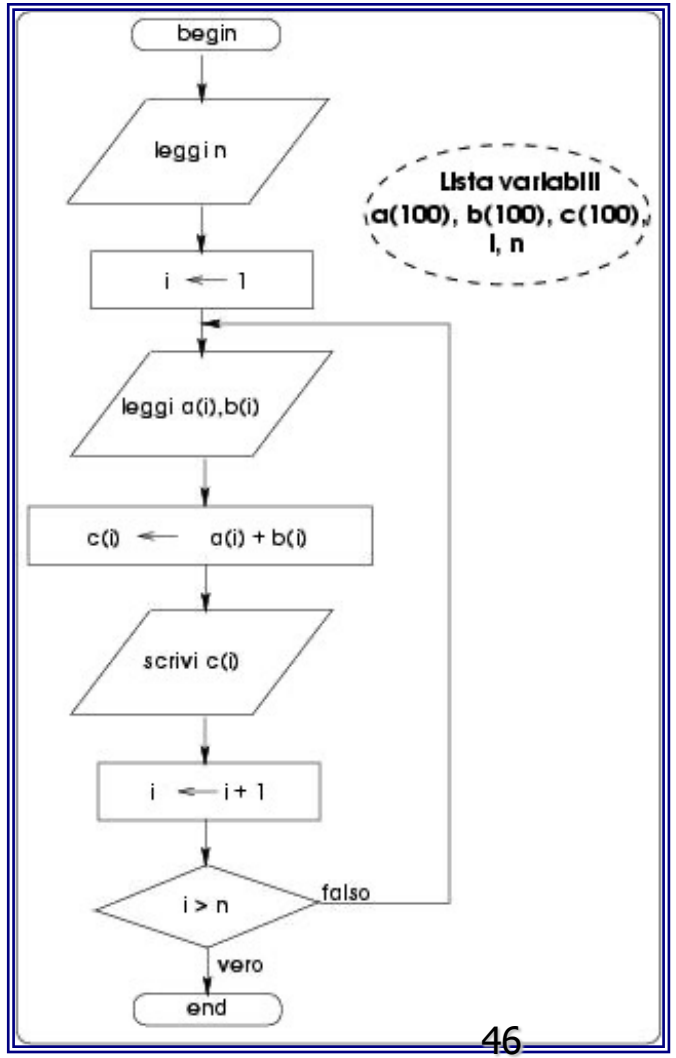
Gli algoritmi iterativi

- **Problema:** Calcolare il vettore somma di due vettori di uguale dimensione *n*



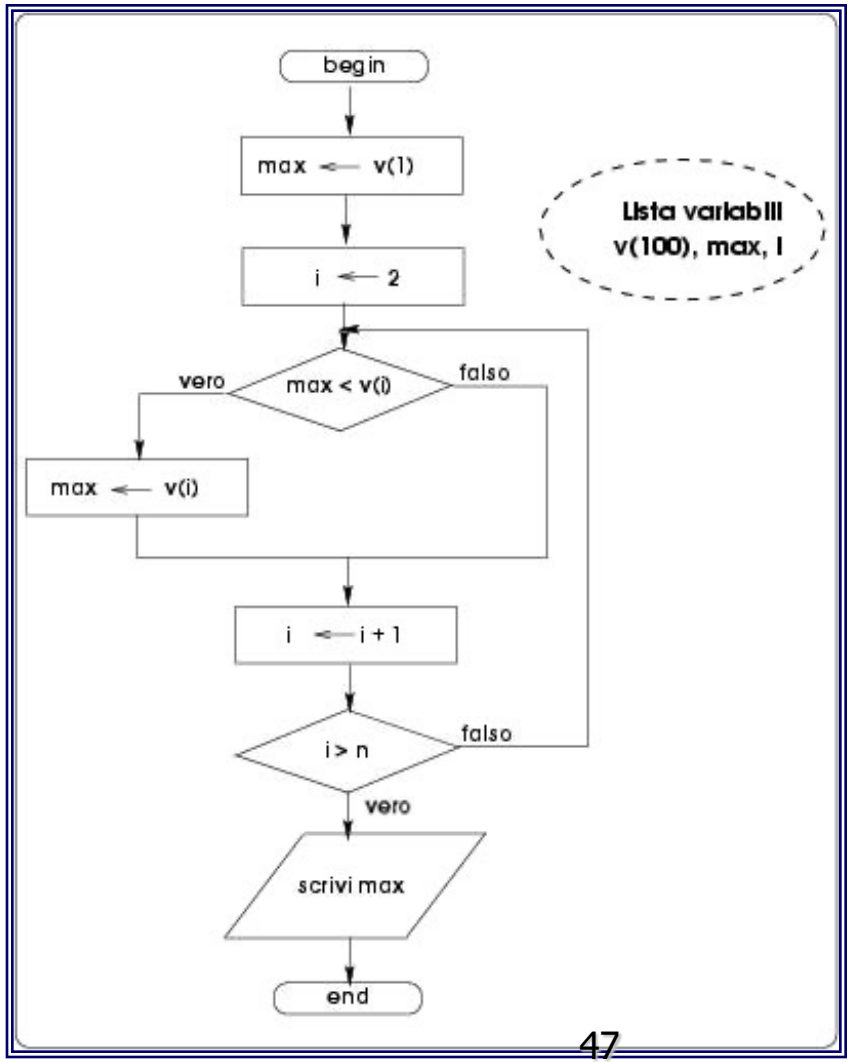
Gli algoritmi iterativi

- L'utilità dei vettori consiste nel poter usare la tecnica iterativa in modo da effettuare la stessa operazione su tutti gli elementi del vettore
- Usando la variabile contatore di un ciclo come indice degli elementi di un vettore è possibile considerarli tutti, uno alla volta, ed eseguire su di essi l'operazione desiderata



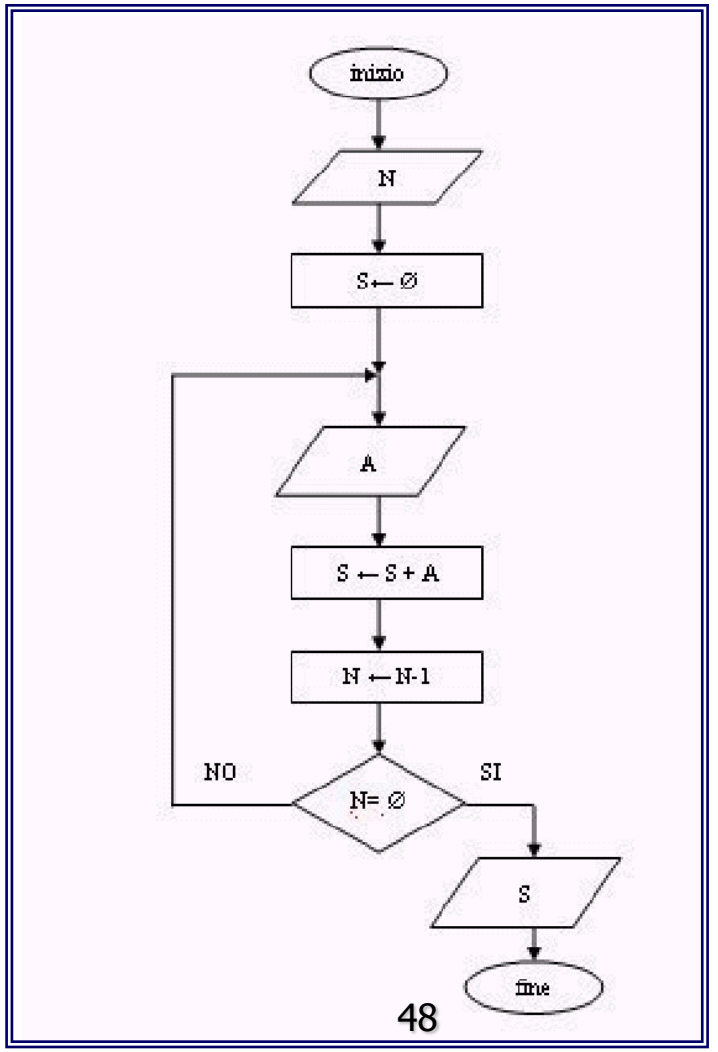
Gli algoritmi iterativi

- **Problema:** Calcolo del massimo elemento di un vettore



Ancora esempi…

- **Problema:** Somma di una sequenza di numeri
  - Indicando con a_i il generico elemento da sommare, la formula generale è
    - $S = a_1 + a_2 + \ldots + a_n$
  - La variabile *n* conta quante volte si ripete l'iterazione: *n* viene decrementata di 1 ad ogni iterazione ed il ciclo termina quando *n* vale 0
  - La variabile A è usata per l'input degli a_i, S per le somme parziali e totale



Gli algoritmi ricorsivi

UNIVERSITÀ DEGLI STUDI DI BRESCIA

- Un algoritmo si dice ricorsivo quando è definito in termini di se stesso, cioè quando una sua istruzione richiede una nuova esecuzione dell'algoritmo stesso
- La definizione ricorsiva di un algoritmo è suddivisa in due parti:

a) la base della ricorsione, che stabilisce le condizioni iniziali, cioè il risultato che si ottiene per i dati iniziali (in generale per 0 e/o 1)

b) la regola di ricorsione, che definisce il risultato per un valore n, diverso dal valore (/i) iniziale per mezzo di un'espressione nella quale si richiede il risultato dell'algoritmo calcolato per $n-1$

Gli algoritmi ricorsivi

- Gli algoritmi ricorsivi sono particolarmente utili per eseguire compiti ripetitivi su un insieme di input variabili
- L'algoritmo ricorsivo richiama se stesso, generando una sequenza di chiamate che ha termine al verificarsi della condizione di terminazione (che in genere si ha con particolari valori d'ingresso)
- Gli algoritmi ricorsivi sono eleganti e sintetici, ma non sempre efficienti, dato che la ricorsione viene implementata mediante l'uso di chiamate di funzione annidate

Gli algoritmi ricorsivi

- Le chiamate di funzione annidate generano una quantità enorme di overhead, occupando lo stack per un numero di istanze pari alle chiamate della funzione che è necessario effettuare per risolvere un dato problema
  - Possibile lo *stack overflow*
  - Costi computazionali dovuti all'impegno del processore per popolare e distruggere lo stack

Gli algoritmi ricorsivi

- **Esempio:** Prodotto di numeri interi

$$a \times b = \begin{cases} 0 & \text{se } b=0 \text{ (base della ricorsione)} \\ a \times (b-1) + a & \text{se } b \neq 0 \text{ (regola di ricorsione)} \end{cases}$$

- Secondo la definizione ricorsiva si ha:

$$3 \times 2 = 3 \times 1 + 3 = 3 \times 0 + 3 + 3 = 0 + 3 + 3 = 6$$

- L'esecuzione di un algoritmo ricorsivo termina sempre: la regola di ricorsione prevede nuove esecuzioni su dati decrescenti, fino ad ottenere i dati di inizio ricorsione

Gli algoritmi ricorsivi

- Esempio: Calcolo del fattoriale di un numero intero
  - Il fattoriale di n è il prodotto di tutti gli interi da 1 ad n, cioè

$$n! = n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1$$

  - Per definizione, $0! = 1$

```
begin fattoriale(n)
    if  n = 0
        then r ← 1
        else r ← n × fattoriale(n–1)
    endif
end
```

Esercizio 1

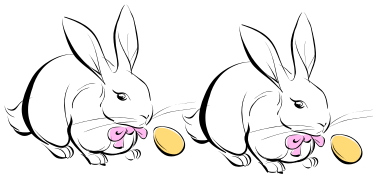
- La successione di Fibonacci
  - Leonardo Pisano, detto Fibonacci, pose il seguente quesito:
    - Una coppia di conigli giovani impiega una unità di tempo a diventare adulta; una coppia adulta impiega una unità di tempo a riprodursi e generare un'altra coppia di conigli (chiaramente giovani); i conigli non muoiono mai
    - Quante coppie di conigli abbiamo al tempo t generico se al tempo t=0 non abbiamo conigli e al tempo t=1 abbiamo una coppia di giovani conigli?

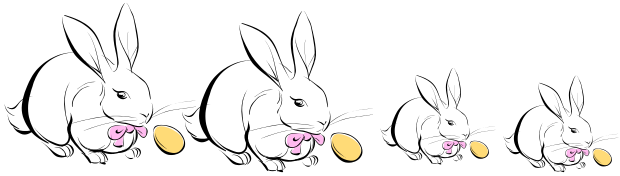
Esercizio 2

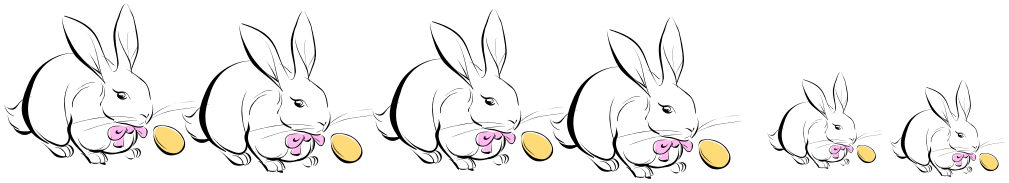
t=0

t=1 

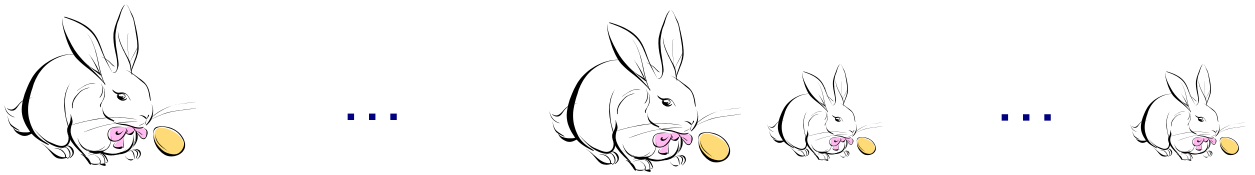
La prima coppia di conigli è data

t=2 

t=3 

t=4 

…

t=N  ?

Esercizio 3

- La successione di Fibonacci
  - Il calcolo di F_n (numero di coppie di conigli), per qualsiasi tempo t, genera la successione dei numeri di Fibonacci
  - La relazione di ricorsione è

$$F_0=0,\ F_1=1,$$
$$F_n = F_{n-1} + F_{n-2}$$

Esercizi

- Scrivere un algoritmo per ogni punto, e rappresentarlo tramite diagramma a blocchi, per la soluzione dei seguenti problemi:
  - calcolare l'area del triangolo
  - trovare il massimo fra due numeri
  - moltiplicare due interi (usando solo l'operazione di somma)
- Formalizzare, tramite diagramma a blocchi, l'algoritmo per…
  - …calcolare le radici reali di equazioni di 2° grado
  - …calcolare il M.C.D. di due numeri con il metodo di Euclide