Calcolo numerico e **programmazione**
Risoluzione di problemi

Tullio Facchinetti
<tullio.facchinetti@unipv.it>

16 aprile 2012

14:37

http://robot.unipv.it/toolleeo

Risoluzione di problemi

per problema si intende un compito che si vuole far risolvere automaticamente a un calcolatore

per risolvere un problema bisogna svolgere le seguenti attività:

- comprendere il problema
- definire un procedimento risolutivo (algoritmo) per il problema
- implementare l'algoritmo in un linguaggio di programmazione

problemi parametrici: dipendono da dati i cui valori non sono noti al momento in cui si vuole affrontare e risolvere il problema; i dati sono i parametri da fornire alla procedura di risoluzione al momento in cui questa viene eseguita

Esempi di problemi

- dati due numeri trovare il maggiore
- dati a e b, risolvere l'equazione ax+b=0
- calcolare il massimo comun divisore fra due numeri dati
- dato un insieme di parole, metterle in ordine alfabetico
- dato un elenco di nomi e relativi numeri di telefono trovare il numero di telefono di una determinata persona
- dato l'archivio dell'anagrafe comunale, trovare tutti i nuclei familiari composti da più di 4 persone
- dato l'archivio dei dipendenti di un'azienda, calcolare lo stipendio di ogni dipendente dell'azienda

la descrizione del problema non fornisce (in genere) un metodo per calcolare il risultato

Problemi non risolvibili

non tutti i problemi sono risolvibili attraverso l'uso del calcolatore

esistono classi di problemi per le quali la soluzione automatica non è possibile:

- se il problema presenta infinite soluzioni
- per alcuni dei problemi non è stato trovato un metodo risolutivo
- per alcuni problemi è stato dimostrato che non esiste un metodo risolutivo automatizzabile

considereremo problemi che ammettono un metodo risolutivo

L'algoritmo

algoritmo: insieme di passi elementari (istruzioni) che, eseguiti secondo un ordine prestabilito, permettono di arrivare ai risultati a partire dai dati del problema

esempi sono: una ricetta di cucina, le istruzioni per far funzionare un elettrodomestico, le istruzioni per installare un programma

L'algoritmo

un algoritmo deve essere espresso in termini delle istruzioni di un esecutore automatico (calcolatore), cioè:

- ciascuna istruzione deve poter essere eseguita dall'esecutore in tempo finito
- l'intera sequenza di istruzioni deve poter essere eseguita in tempo finito, per ogni possibile insieme di ingresso che soddisfa la pre-condizione del problema

Proprietà di un algoritmo

per un algoritmo devono essere valide una serie di condizioni:

1. realizzabilità: ogni operazione prevista dall'algoritmo deve essere eseguibile con le risorse a disposizione
2. non ambiguità: ogni operazione deve essere univocamente interpretabile dall'esecutore, cioè deve essere descritta in modo preciso; il risultato non deve cambiare al variare dell'esecutore (macchina/persona) dell'algoritmo (carattere deterministico);
3. finitezza: il numero totale di istruzioni da eseguire, per ogni insieme di dati di ingresso, è finito e le operazioni da esse specificate devono essere eseguite un numero finito di volte

Proprietà di un algoritmo

un algoritmo deve

- essere costituito da operazioni appartenenti ad un determinato insieme di operazioni fondamentali (sistema formale)
- essere applicabile a qualsiasi insieme dei dati di ingresso appartenenti al dominio di definizione dell'algoritmo

un algoritmo possiede due qualità fondamentali:

1. la correttezza, ovvero l'algoritmo deve permettere effettivamente di risolvere il problema
2. l'efficienza, cioè l'esecuzione dell'algoritmo deve richiedere un uso limitato di risorse

Proprietà di un algoritmo

tipiche risorse che devono essere salvaguardate sono il tempo di esecuzione e la memoria utilizzata

altre qualità di un algoritmo sono:

- la leggibilità, in quanto l'algoritmo essere quanto più facilmente comprensibile possibile
- la modificabilità: l'algoritmo deve essere facilmente modificabile, a fronte di (piccole) modifiche nelle specifiche del problema risolto dall'algoritmo
- la parametricità, cioè l'algoritmo deve dipendere da dati i cui valori non sono noti al momento in cui si vuole affrontare e risolvere il problema
- la riusabilità

Esempi di algoritmo: prodotto di matrici

problema: calcolare il prodotto di due matrici A e B di dimensione rispettivamente $m \times n$ e $n \times p$.

dati di ingresso: gli $m \cdot n$ valori che compongono A, che saranno indicati con a_{ij}; gli $n \cdot p$ valori che compongono B, che saranno indicati con b_{ij}.

pre-condizione: deve essere verificato che il numero di colonne di A deve essere uguale al numero di righe di B.

dati di uscita: una matrice C di dimensione $m \times p$.

specifica dell'algoritmo:

- l'elemento c_{ij} della matrice C viene calcolato come

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \ldots + a_{in} b_{nj}$$

Classificazione degli algoritmi

- a seconda del tipo di implementazione
- a seconda del paradigma di design
- a seconda del campo di studio
- a seconda della complessità

Tipo di implementazione

- ricorsivo: una funzione chiama se' stessa fino al raggiungimento di una condizione di uscita
- iterativo: uso di istruzioni ripetute ciclicamente
- seriale, parallelo, distribuito: rispettivamente, viene eseguito sequenzialmente su un processore, più istruzioni sono eseguite contemporaneamente da più processori della stessa macchina, o da più processori su macchine diverse interconnesse da una rete di comunicazione
- deterministico o non-deterministico (euristico): a seconda che, ad ogni passo, possa essere presa una decisione univoca o meno sul passo successivo da intraprendere
- esatto o approssimato: a seconda che si possa trovare la soluzione esatta del problema oppure una sua approssimazione
- quantistico: usa un modello di processore quantistico

Paradigma di design

- ricerca esaustiva (brute force): verifica tutte le possibili soluzioni per identificare la migliore
- divide et impera: riduce il problema in sotto-problemi successivi sempre più semplici
- metodi greedy: le soluzioni dei sotto-problemi sono approssimate e vengono raffinate quando nuove informazioni utili si rendono disponibili
- programmazione lineare: il problema è espresso da un insieme di disuguaglianze (vincoli), e risolto ottimizzando (min o max) una funzione lineare dei parametri
- ricerca: problemi rappresentabili mediante grafi e risolvibili mediante la ricerca all'interno del grafo

Campo di studio

ogni campo di ricerca richiede algoritmi per la risoluzione di problemi specifici

alcuni esempi

- ricerca di un elemento in un insieme
- ordinamento degli elementi di un insieme
- fusione degli elementi di due o più insiemi
- calcolo numerico: es. calcolo dello zero di una funzione
- crittografia
- compressione dati: riduzione della memoria necessaria per rappresentare un'informazione
- data mining: identificazione di dati interessanti all'interno di una grossa mole di dati
- scheduling: decisione di assegnamento ad una entità di una risorsa limitata

Complessità

la complessità è legata al numero di operazioni da eseguire a seconda dei dati in ingresso

se il problema da risolvere deve elaborare un insieme di n dati:

- $O(1)$: complessità che non dipende dalla quantità di dati in ingresso
- $O(n)$: complessità lineare
- $O(n^x)$: complessità polinomiale
- $O(\log n)$: complessità logaritmica
- $O(x^n)$: complessità esponenziale (non polinomiale)

Operazioni fondamentali

le operazioni base per la realizzazione di un algoritmo sono 4:

1. trasferimento di informazioni: acquisizione dati, visualizzazione risultati intermedi, scrittura risultati finali
2. esecuzione di calcoli
3. assunzione di decisioni: scelta della successiva operazione da compiere sulla base di risultati intermedi
4. esecuzione di iterazioni: ripetizione di sequenze di operazioni

- per rappresentare (descrivere) un algoritmo non è possibile utilizzare il linguaggio naturale in quanto questo può presentare ambiguita' e causare false interpretazioni
- è necessario, pertanto, utilizzare linguaggi sintetici e standardizzati in modo da consentire all'esecutore una interpretazione univoca dei passi da svolgere

Rappresentazione degli algoritmi: diagrammi di flusso

- per la descrizione degli algoritmi si utilizzano particolari rappresentazioni grafiche denominate diagrammi di flusso, schemi a blocchi o flowchart

> è un formalismo che consente di rappresentare graficamente gli algoritmi

- questa descrizione costituisce un efficace strumento per la descrizione degli algoritmi, più valido di una esposizione di tipo discorsivo (generica e ambigua)
- qualsiasi algoritmo può essere decomposto in poche strutture elementari

Diagrammi di flusso

1. flowchart o schema a blocchi: qualsiasi combinazione di blocchi operativi e decisionali; ogni arco uscente da un blocco va in ingresso a un solo blocco

2. flusso di controllo: ordine di percorrenza dei blocchi individuato da un flowchart

3. struttura di controllo: flowchart parziale da assumere come modello di computazione, con un ingresso e un'uscita

4. sequenza di computazione: successione di blocchi operativi e decisionali prodotta dall'esecuzione di un flowchart per un certo insieme di dati in ingresso

Programmazione strutturata

è una tecnica di programmazione che ha lo scopo di semplificare la struttura di un algoritmo disciplinando l'uso delle strutture di controllo utilizzabili all'interno di uno schema blocchi

prevede l'uso di un numero limitato di strutture di controllo fondamentali, con un ingresso ed una uscita:

1. sequenza
2. selezione
3. iterazione

Programmazione strutturata

la programmazione strutturata vincola quindi l'utilizzo delle strutture di controllo, ma offre i seguenti vantaggi:

- rende possibile una progettazione di tipo top-down
- permette la definizione di algoritmi più leggibili, essendo più facile individuare i moduli corrispondenti alle varie parti di cui si compone l'algoritmo
- test, correzione e manutenzione del programma sono perciò più semplici, anche se per il test del sistema completo bisogna attendere di assemblare tutti i componenti

Progettazione top-down

si parte dall'obiettivo finale del problema e si esplicitano e raffinano ricorsivamente le parti che lo compongono

- la strategia per risolvere il problema viene originata dall'obiettivo del problema stesso
- ad ogni passo vengono identificati dei sotto-blocchi logici correlati che vengono rifiniti sempre piu' (decomposizione, specializzazione, specificazione)
- il processo di rifinizione termina quando si raggiunge il livello di dettaglio sufficiente per l'applicazione da risolvere
- tecnica adatta a problemi complicati di tipologie differenti
- contrapposta alla progettazione bottom-up

Rappresentazione degli algoritmi: diagrammi di flusso

- un diagramma di flusso (o flowchart) descrive le azioni da eseguire ed il loro ordine di esecuzione
- ogni azione elementare corrisponde ad un simbolo grafico (blocco) diverso (sono convenzioni non universali)
- ogni blocco ha uno o più rami in ingresso ed un ramo in uscita (ad esclusione del blocco di decisione che ne ha due); collegando tra loro i vari blocchi attraverso i rami, si ottiene un diagramma di flusso
- un diagramma di flusso appare, quindi, come un insieme di blocchi di forme diverse che contengono le istruzioni da eseguire, collegati fra loro da linee orientate che specificano la sequenza in cui i blocchi devono essere eseguiti (flusso del controllo di esecuzione o sequenza di computazione)

Diagrammi di flusso: variabili

l'insieme dei dati di ingresso e dei risultati vengono rappresentati attraverso dei nomi simbolici, detti variabili

ad esempio: a, b, c, somma, ecc.

talvolta può essere necessario introdurre delle variabili temporanee, necessarie alla risoluzione del problema: tali variabili vengono anch'esse rappresentate da nomi simbolici

ad esempio: delta, somma_parziale, ecc.

Blocchi (o istruzioni)

ogni azione è rappresentata in un flowchart da un blocco grafico

- blocchi semplici: esecuzione di operazioni sui dati
- blocco condizione: in base al verificarsi di una condizione, permette di differenziare il comportamento dell'algoritmo, mediante la scelta tra due strade alternative

Valori e grandezze

valori:

- numerici: interi e reali (es. 10 23.4)
- logici: Vero e Falso
- alfanumerici, o stringhe (es. “AAAA”, “C.Colombo”)

grandezze:

- variabili: rappresentate da un nome simbolico cui è assegnato un valore che può cambiare durante l’esecuzione dell’algoritmo
- costanti: quantita’ note a priori, il cui valore non cambia durante l’esecuzione

Espressioni

sequenze di variabili e costanti combinate fra loro mediante operatori
(es. operatori aritmetici: +, -, *, /)

ad esempio

s + r *5
A
100

nella valutazione di una espressione, si sostituisce ad ogni variabile il suo valore attuale e si eseguono le operazioni secondo un ordine prestabilito da regole di precedenza (possono comparire parentesi)

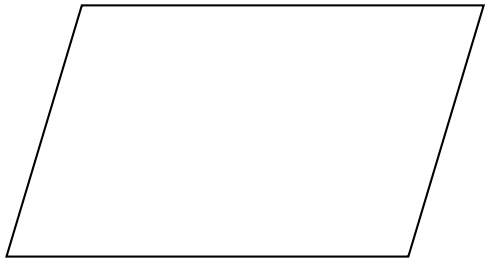
Espressioni

a tutte le variabili che compaiono nell'espressione deve essere stato associato un valore prima della valutazione dell'espressione

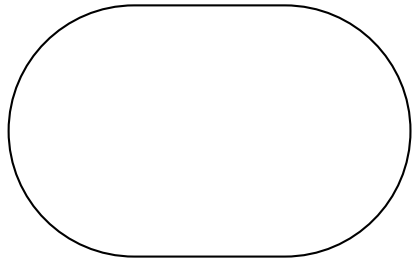
tipi di espressioni:

- espressioni aritmetiche: danno come risultato un valore aritmetico (ad esempio, un intero, un reale, etc.): +, -, *, /, ...
- espressioni relazionali e logiche: danno come risultato vero o falso ($>$, $<$, $=$, $\geq$, $\leq$, $\neq$)

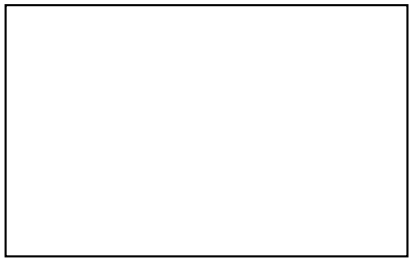
Blocchi



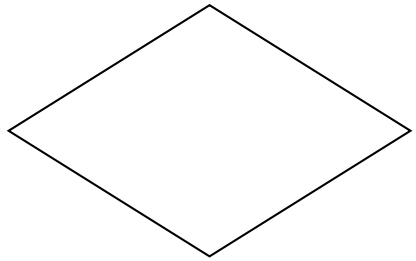
ingresso/uscita



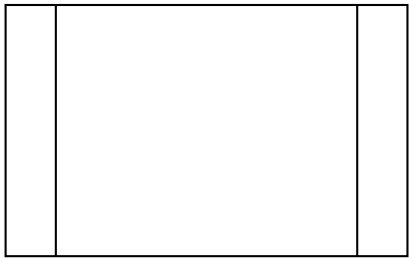
inizio/fine



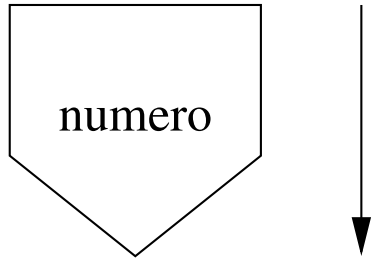
elaborazione/calcolo



decisione



elaborazione predefinita

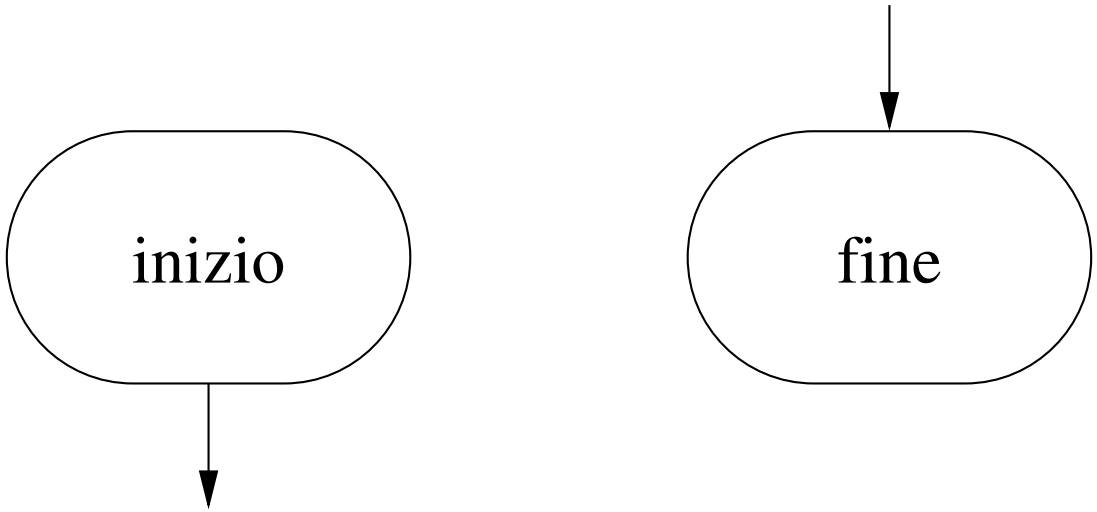


connessioni

Inizio/fine

inizio e fine di un algoritmo

- inizio è il blocco da cui deve iniziare l'esecuzione (uno solo)
- il blocco fine fa terminare l'esecuzione dell'algoritmo (almeno uno)

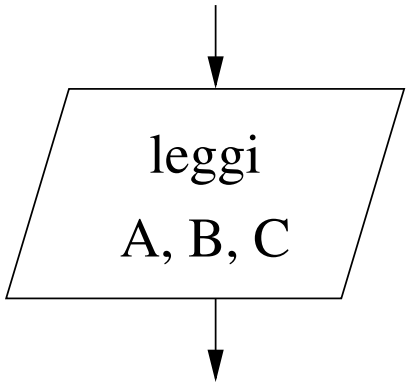


Ingresso/lettura

esecuzione dell'istruzione:

- si ricevono dall'unita' di ingresso (per esempio, la tastiera) tanti valori quante sono le variabili specificate all'interno del blocco, e si assegnano nello stesso ordine alle variabili
- A, B, C sono nomi di variabili

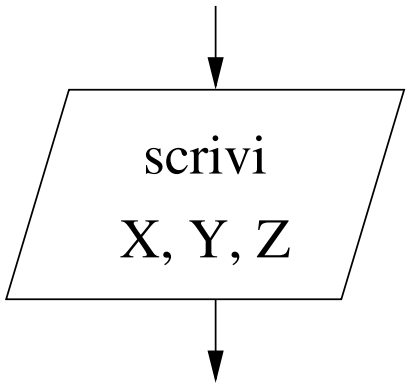
es. "Leggi i tre valori dati in ingresso, ed assegnali rispettivamente alle variabili A, B, e C"



Uscita/stampa

si calcolano i valori delle espressioni e si trasmettono all'unita' di uscita (ad esempio, il video)

- X, Y, Z possono essere variabili
- se X, Y, Z sono espressioni → "calcola i valori delle espressioni X, Y e Z, e trasmettili in uscita"



N.B.: i valori di X, Y, Z non vengono comunque alterati dall'esecuzione del blocco

Assegnamento

si calcola il valore dell'espressione a destra del simbolo "="
e lo si assegna alla variabile indicata a sinistra del simbolo "=",
con eventuale perdita del valore precedente di V

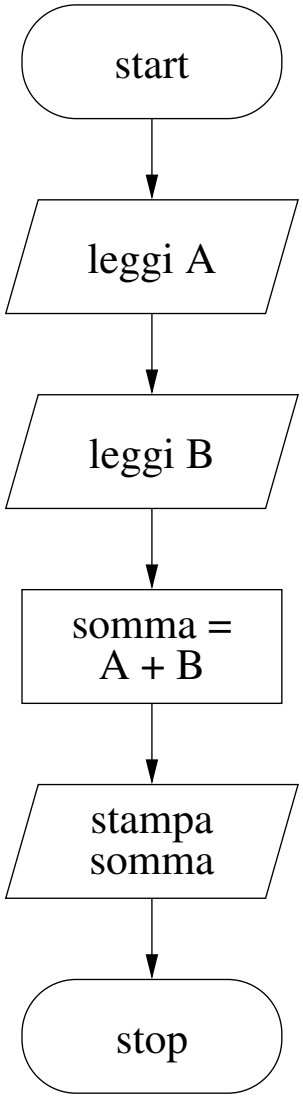
$$V = E$$

dove

- V è il nome di una variabile
- E è una espressione

si interpreta come: *"calcola il valore dell'espressione E e assegnalo alla variabile V"*

Esempio: somma di due numeri



Strutture di controllo

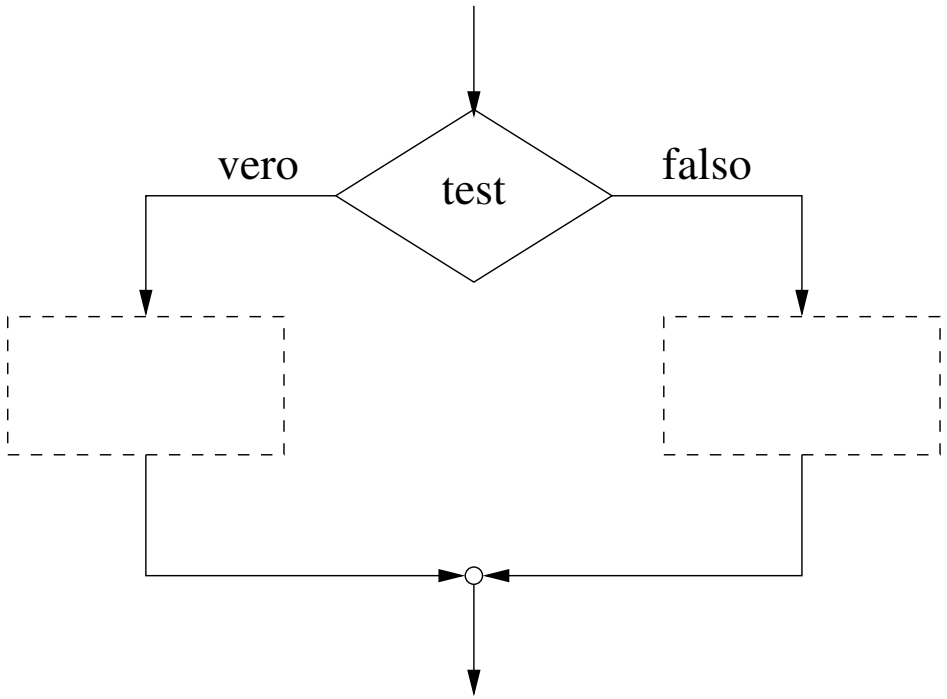
mediante i blocchi fondamentali, è possibile costruire delle strutture tipicamente utilizzate per il controllo del flusso di esecuzione dell'algoritmo

le strutture di controllo in questione sono

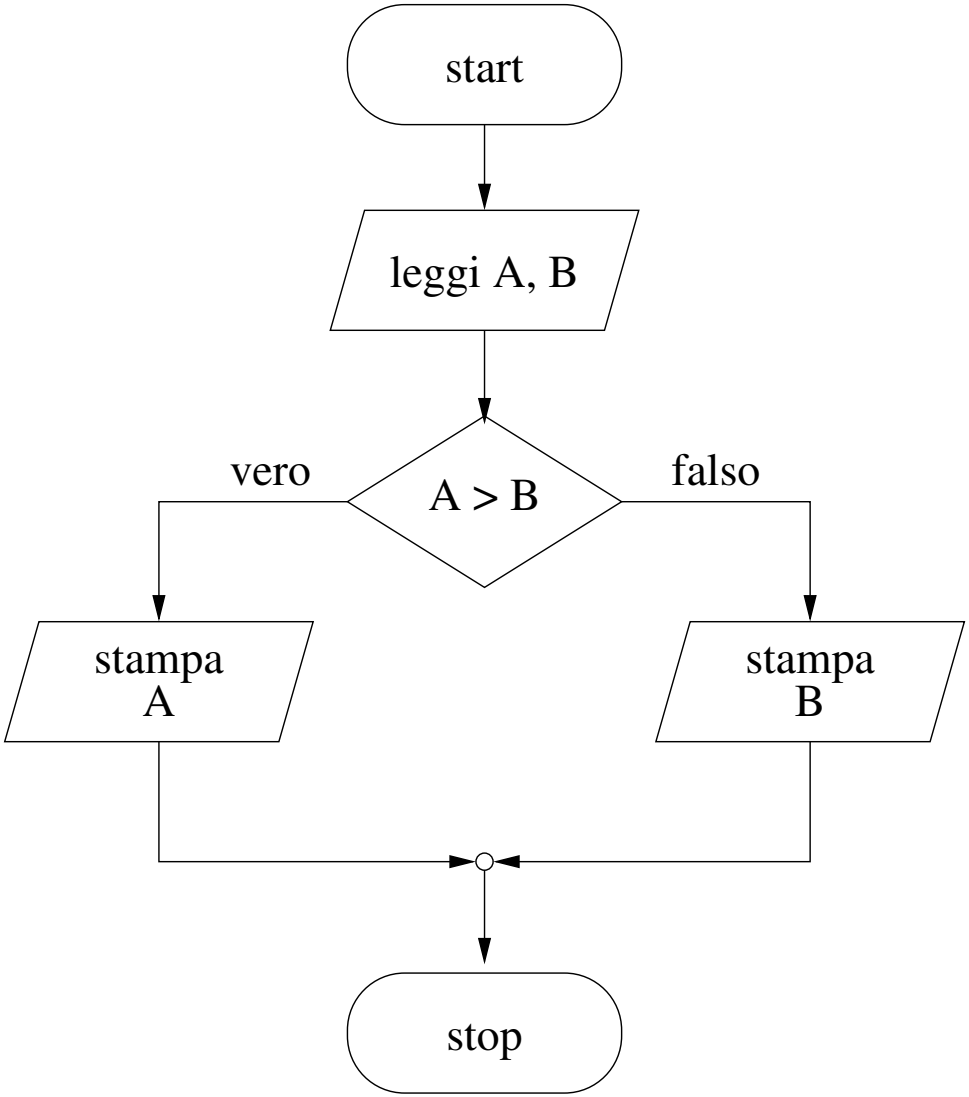
- selezione
- iterazione

Strutture di controllo: selezione

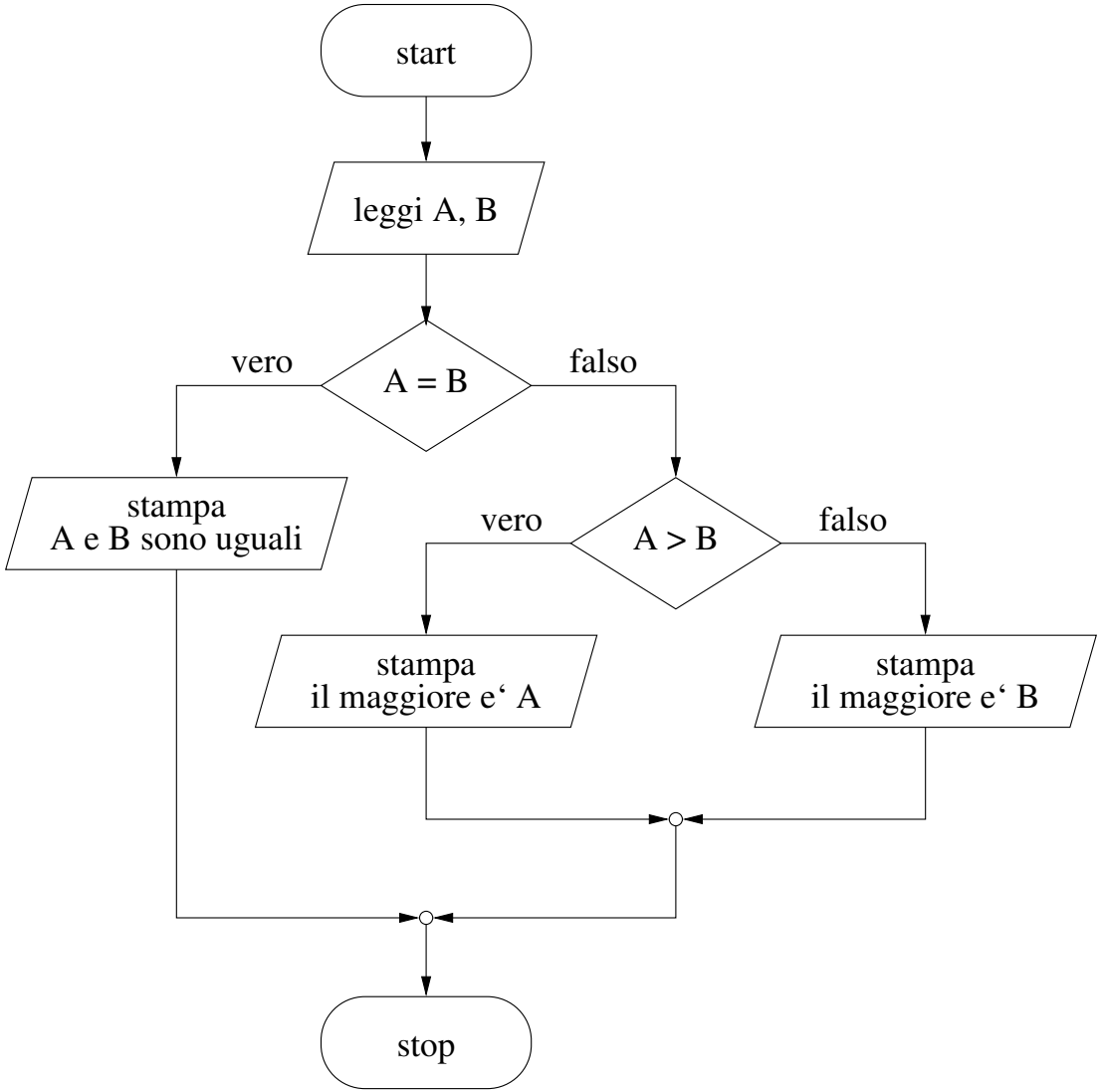
esprime la scelta tra due possibili azioni, o sequenze di azioni, mutuamente esclusive



Stampa il massimo tra due numeri



Confronta due numeri



Strutture di controllo: iterazione

esprime la ripetizione di una sequenza di istruzioni

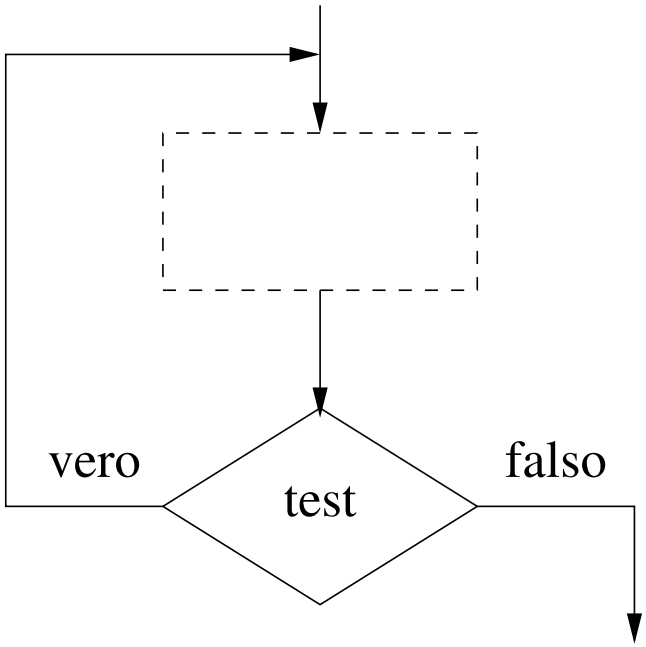
nel caso piu' generale, è costituita da:

- inizializzazione: assegnazione dei valori iniziali alle variabili caratteristiche del ciclo (viene eseguita una sola volta)
- corpo: esecuzione delle istruzioni fondamentali del ciclo che devono essere eseguite in modo ripetitivo
- modifica: modifica dei valori delle variabili che controllano l'esecuzione del ciclo (eseguito ad ogni iterazione)
- controllo: determina, in base al valore delle variabili che controllano l'esecuzione del ciclo se il ciclo deve essere ripetuto o meno.

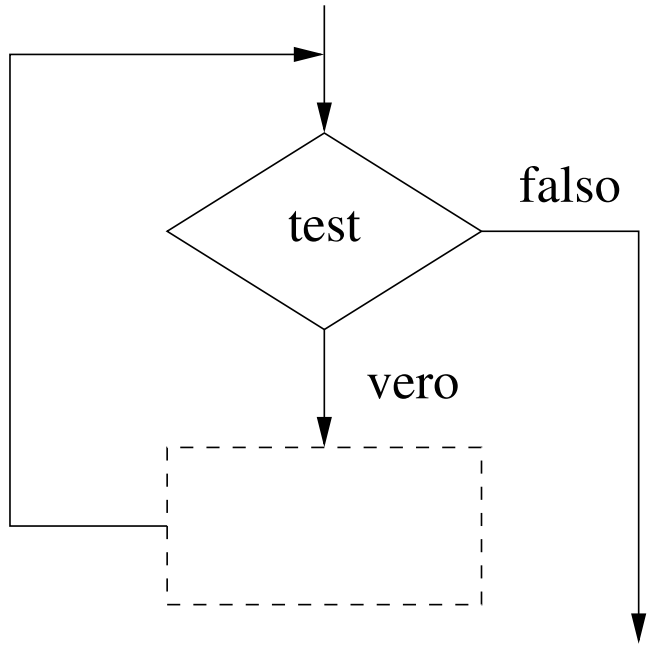
Strutture di controllo: iterazione

sono possibili due configurazioni:

- repeat-until
- while-do

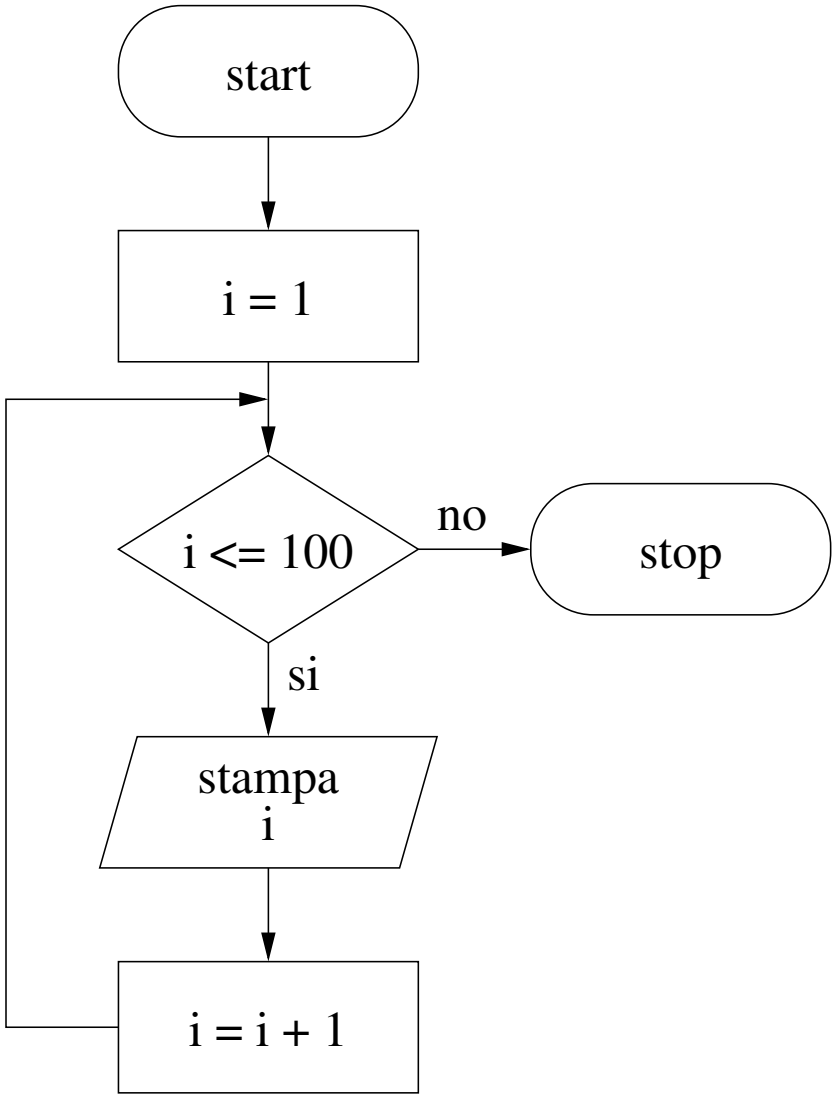


repeat–until

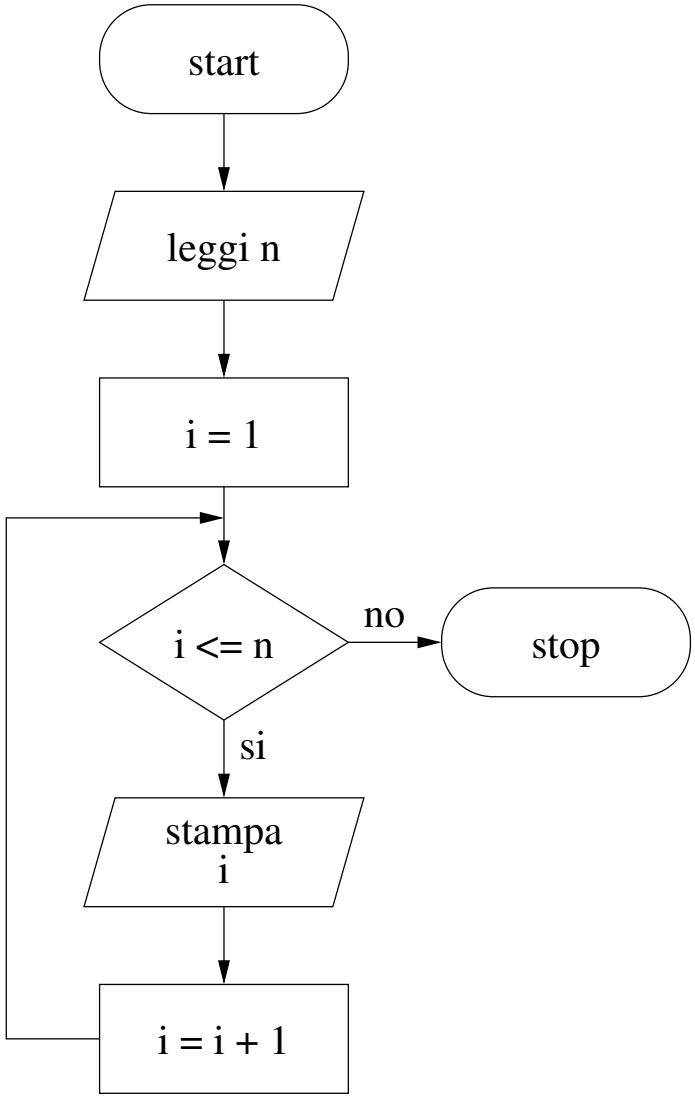


while–do

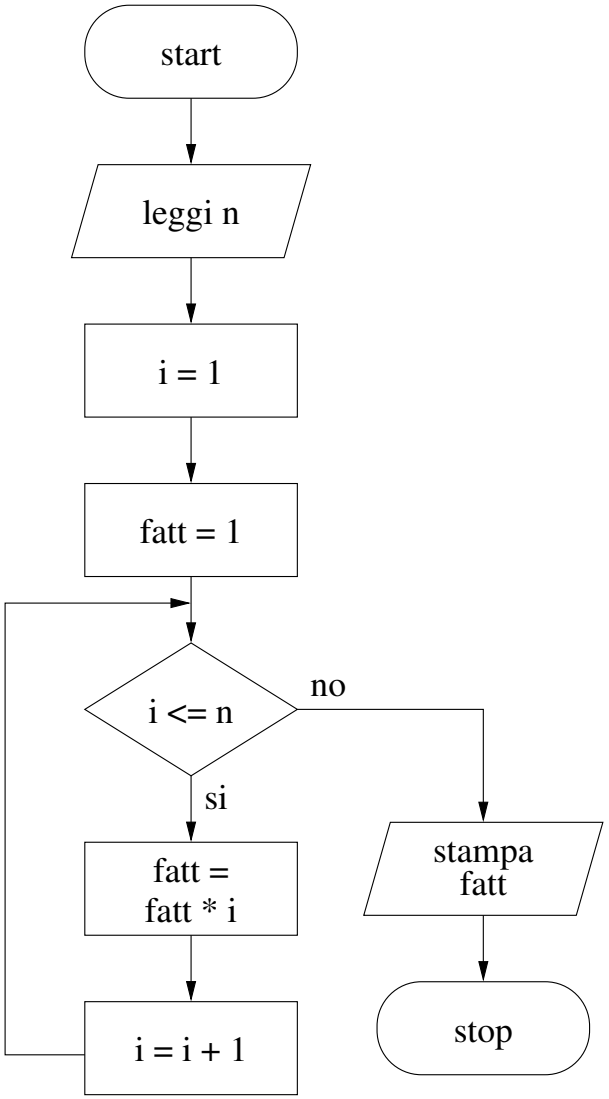
Esempio: stampa dei primi 100 numeri



Esempio: stampa dei primi n numeri



Esempio: fattoriale di n



Equivalenza tra diagrammi di flusso

Equivalenza debole

due flowchart (algoritmi) sono debolmente equivalenti se, per ogni insieme di dati in ingresso, generano gli stessi dati in uscita

Equivalenza forte

due flowchart sono fortemente equivalenti se sono debolmente equivalenti e le rispettive sequenze di computazione sono uguali, per ogni insieme di dati in ingresso

Equivalenza fortissima

due flowchart sono fortissimamente equivalenti se sono fortemente equivalenti ed inoltre in essi compaiono lo stesso numero di volte gli stessi blocchi elementari

Programmazione strutturata

esiste un problema fondamentale:

è sicuro che usando solo le strutture di controllo fondamentali non si limita la capacità di realizzare algoritmi?

che equivale a domandarsi

esistono problemi non risolubili per mezzo delle sole strutture di controllo fondamentali?

la risposta è data dal teorema di Jacopini-Böhm che asserisce la possibilità di realizzare qualunque algoritmo con le sole strutture di controllo fondamentali

Teorema di Jacopini-Böhm

siano

- $\mathcal{P}$ l'insieme di tutti gli algoritmi realizzabili
- $\mathcal{D}$ l'insieme di tutti gli algoritmi realizzabili facendo uso esclusivo delle tre strutture di controllo fondamentali (sequenza, selezione e iterazione)

allora

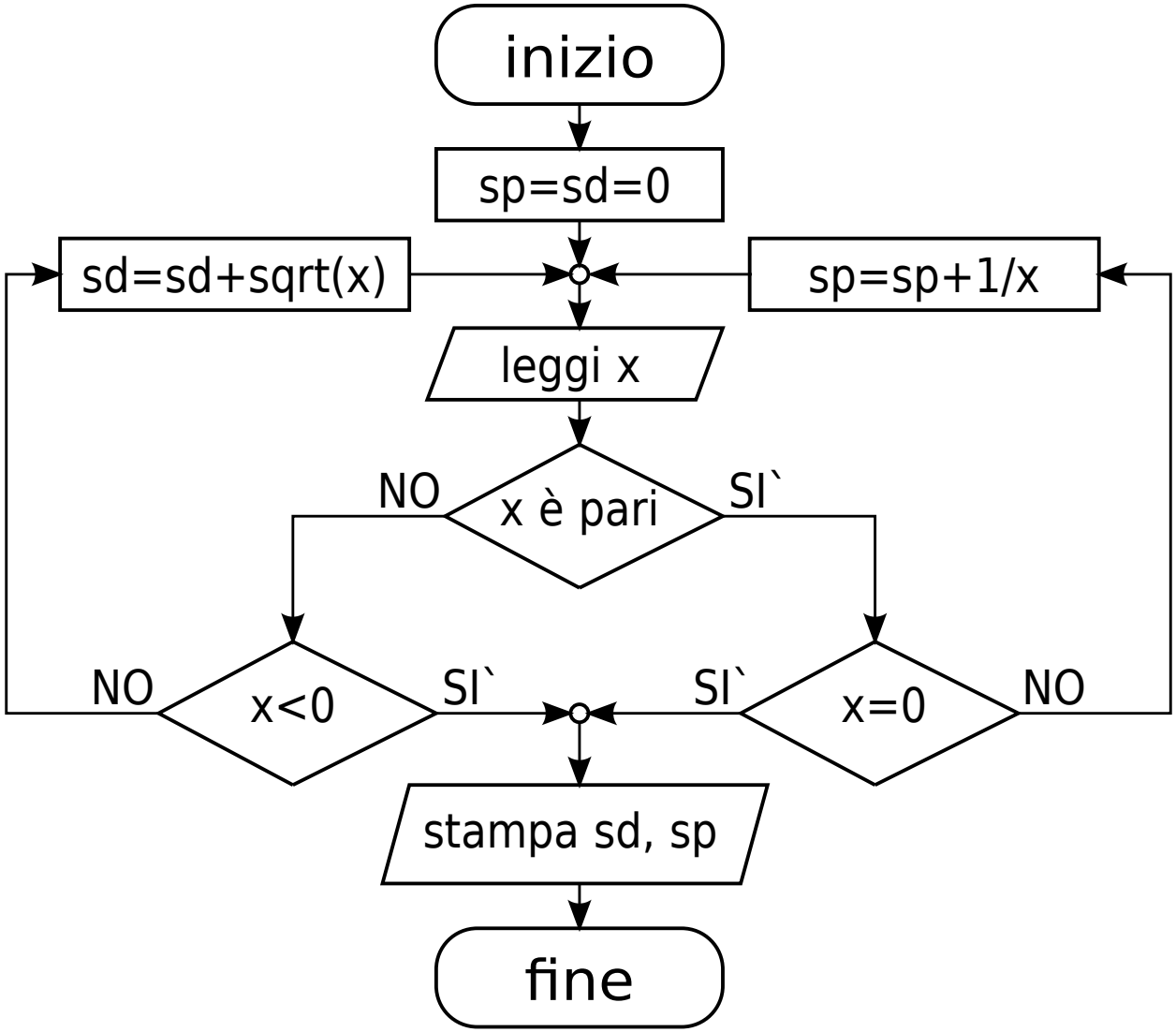
dato un programma $p \in \mathcal{P}$
esiste sempre un programma $d \in \mathcal{D}$
che risulta debolmente equivalente a p

Esempio di flowchart

realizzare diagramma di flusso che risolva il seguente problema:

- continui a leggere dei valori numerici
- effettuare la somma delle radici quadrate di tutti i numeri dispari inseriti
- calcolare la somma dell'inverso di tutti i numeri pari
- il programma termina quando viene inserito un valore che non permette di effettuare correttamente il calcolo nel dominio dei numeri reali
- prima di terminare, stampare i valori calcolati

Esempio di flowchart



Cos'è lo pseudo-codice

descrizione informale di alto livello adatta a rappresentare un algoritmo o programma

- usa le strutture di un linguaggio di programmazione
- adatto ad essere letto dall'uomo
- non adatto per la specifica formale di programmi (non direttamente comprensibile ad una macchina)
- omette dettagli non essenziali per la comprensibilità (es. dichiarazione di variabili)

Caratteristiche

- la descrizione può essere completata da testo in linguaggio naturale
- tipicamente utilizzato in libri di testo, in articoli scientifici, nella pianificazione dello sviluppo di programmi
- non esiste uno standard di rappresentazione
- metodo alternativo a flowchart e UML
- più compatto di questi ultimi

Istruzioni condizionali

if $i \geq maxval$ **then**
    $i \leftarrow 0$
else
    if $i + k \leq maxval$ **then**
        $i \leftarrow i + k$
    end if
end if

Ciclo `for`

es.: ciclo che stampa a video i primi 10 numeri naturali

for $i = 1, i \leq 10$ **do**
    stampa i
    $i \leftarrow i + 1$
end for

Ciclo `while-do`

es.: ciclo che stampa a video i primi 10 numeri naturali

$i = 1;$
while $i \leq 10$ **do**
    stampa i
    $i = i + 1$
end while

Ciclo `repeat-until`

es.: ciclo che stampa a video i primi 10 numeri naturali

$i = 1;$
repeat
    stampa i
    $i = i + 1$
until $i \leq 10$

Esempio di pseudo-codice

scrivere lo pseudo-codice che risolva il seguente problema:

- continui a leggere dei valori numerici
- effettuare la somma delle radici quadrate di tutti i numeri dispari inseriti
- calcolare la somma dell'inverso di tutti i numeri pari
- il programma termina quando viene inserito un valore che non permette di effettuare correttamente il calcolo nel dominio dei numeri reali
- prima di terminare, stampare i valori calcolati

Esempio di pseudo-codice

$sd \leftarrow 0$
$sp \leftarrow 0$
repeat
    leggi val
    if val pari **then**
        if val $\neq 0$ **then**
            $sp \leftarrow sp + (1/val)$
        end if
    else
        if val > 0 **then**
            $sd \leftarrow sd + \sqrt{val}$
        end if
    end if
until (val pari e val $\neq 0$) oppure (val dispari e val > 0)
stampa sd e sp