Università degli Studi di Bari “Aldo Moro”
Corso di Laurea in Informatica

Corso di Programmazione

Prof.ssa Teresa Roselli
teresa.roselli@uniba.it

Programma del corso di
PROGRAMMAZIONE

- Problemi ed Algoritmi
- Linguaggi di Programmazione
- Dati e Istruzioni
- Scomposizione di Problemi
- Tipi Semplici
- Tipi Strutturati
- Record e Insiemi
- Programmazione Modulare
- Sequenze e File di Testo
- Progettazione di Programmi
- Algoritmi Fondamentali

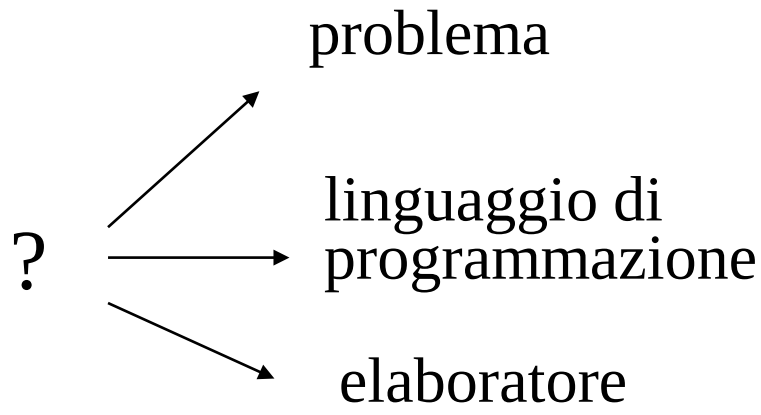
Testi consigliati

- N. Wirth - Principi di programmazione strutturata - ISEDI (UTET Libreria)
  (testo di riferimento per la programmazione strutturata)
- G. Dromey - Algoritmi Fondamentali - Jackson Libri
  (raccolta di tutti gli algoritmi di base e della relativa documentazione)
- Materiale on line fornito dal docente

Corso di Programmazione
Problemi ed Algoritmi

Prof.ssa Teresa Roselli
Teresa.roselli@uniba.it

PROGRAMMARE UN ELABORATORE



Realizzare un programma per risolvere mediante l'elaboratore un determinato problema

Problema

Nasce dalla necessità di ottenere qualcosa non immediatamente raggiungibile o direttamente ottenibile

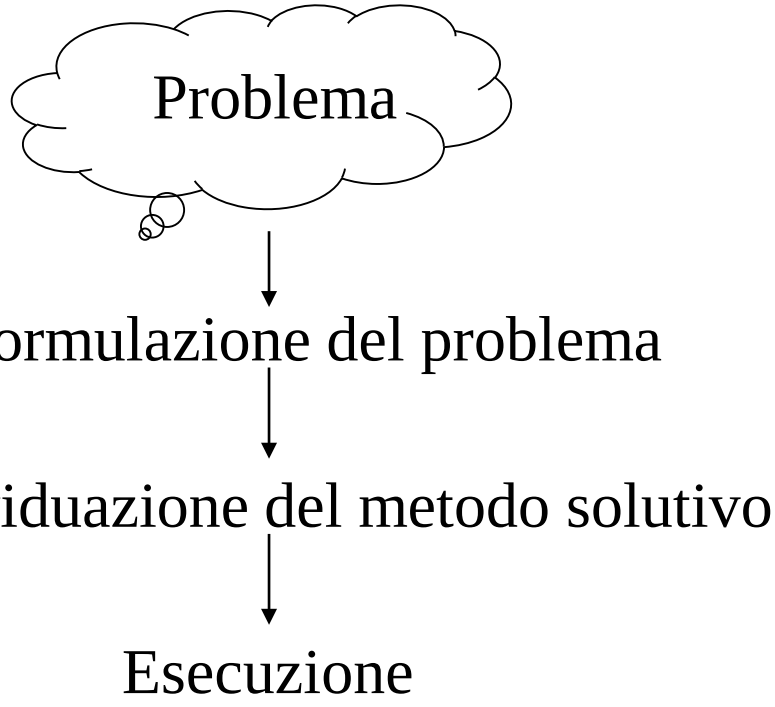
siamo di fronte ad un problema

Spetta al risolutore del problema individuare il metodo di soluzione che consenta di giungere all'obiettivo, ottenere la soluzione desiderata ovvero il risultato

Problema

- **Espressione** da valutare
  per ottenere un *risultato* finale
- **Insieme** finito **di attività** da compiere
  per ottenere un *effetto* desiderato

Problem Solving



-Risolvere un problema significa passare dalla formulazione alla individuazione del metodo di soluzione
-L’applicazione del metodo solutivo ai dati del problema per ottenere il risultato significa attuare il PROCESSO di SOLUZIONE

Problem Solving

Fasi per arrivare alla soluzione di un problema

- Formulazione del problema
  - Si analizza il problema e si individuano gli oggetti su cui operare e i risultati da ottenere: in questa fase si individuano i dati di ingresso e quelli di uscita o risultati
- Costruzione del metodo di soluzione
- Esecuzione

Problem Solving

Come passare dalla formulazione del problema all'individuazione del metodo di soluzione

Disponiamo di

- una descrizione (anche parziale) di una situazione iniziale e di una situazione finale desiderata (obiettivo)
  - Espresse in un linguaggio che permetta di descrivere situazioni in termini di oggetti e relazioni tra essi
- un insieme di operatori applicabili a situazioni per trasformarle in nuove situazioni
  - Espressi in un linguaggio che fa riferimento al processo
  - Ogni sequenza di operatori è un operatore (composto)

Problem Solving

La soluzione è

un operatore (composto) nel linguaggio di processo che trasforma l'oggetto che descrive la situazione iniziale in quello che descrive la situazione desiderata

Problem Solving

- Costruzione del metodo solutivo legata a
  - operazioni semplici disponibili
  - modalità secondo cui possono essere connesse e composte per realizzare operazioni più complesse
    - Es.: Sequenzializzazione ovvero eseguire una dopo l'altra operazioni semplici alla scopo di realizzare operazioni complesse

Problem Solving

Costruzione del metodo di soluzione

- Creazione di un modello che riporti:
  - Relazioni tra dati di ingresso e dati di uscita
  - Informazioni e operazioni necessarie per ottenere il risultato
- Descrizione della soluzione in termini di operazioni eseguibili

  La descrizione di *come* un compito deve essere eseguito è una sequenza ben definita di passi elementari detta

  ALGORITMO

Problem Solving

Esecuzione

L’esecuzione delle istruzioni (passi) dell’algoritmo è un *processo* che opera sui dati di ingresso per produrre dei risultati.

Dato un *soddisfacente* metodo di soluzione è possibile *delegare* l’esecuzione ad un altro soggetto ovvero l’esecuzione è

- delegabile ad un esecutore (artificiale) purchè
  - Capisca la descrizione della soluzione
  - è in grado di eseguire le operazioni richieste

Algoritmo

- Serie di prescrizioni o istruzioni che specifica l'insieme delle azioni da compiere per poter risolvere un problema
  - [Al-Khowarizmi, IX sec., matematico]
    - Regole per eseguire le 4 operazioni aritmetiche sui numeri scritti in notazione decimale
  - Esempi:
    Processo: fare una torta, calcolare l'area di un triangolo,..

    Algoritmo: ricetta da cucina, formula (S=(b*h)/2)

    Passo elementare: prendere 3 uova, moltiplica b per h

Algoritmi

Costruire un algoritmo equivale a:

- Esaminare una specifica realtà o problema
- Costruirne un'**astrazione**
- Rappresentarla (più o meno) formalmente
- Individuare una *sequenza* di azioni che, eseguite, risolvano il problema nel mondo dell'astrazione

Il processo di analisi e astrazione è difficile da dominare per problemi complessi

Algoritmi
Proprietà

- Finitezza
  - Spaziale
  - Temporale

  (ovvero: le istruzioni di un algoritmo devono essere un insieme finito e una qualunque esecuzione dell'algoritmo deve terminare in un tempo finito)
- Generalità
  - Classe di problemi
  - Dominio di definizione

  (l'algoritmo deve risolvere una classe di problemi, ovvero deve essere applicabile a qualsiasi insieme di dati appartenenti al dominio di definizione dell'algoritmo)
- Non ambiguità

  (ovvero l'algoritmo non deve essere costituito da istruzioni che si contraddicono

Algoritmi

La descrizione dell'algoritmo deve essere

- Univoca: non deve dare adito ad interpretazioni errate
- Completa: devono essere previste esattamente tutte le azioni necessarie
- Ripetibile: deve poter essere eseguito da più esecutori, con medesime caratteristiche, con garanzia di successo

Processo di esecuzione

Può essere delegato ad un processore diverso dall'estensore del metodo di soluzione

- Essere umano
- Sistema meccanico

Requisiti per la delega ad un esecutore meccanico:

non può prescindere dalle operazioni eseguibili dette *operazioni basiche* o *azioni primitive*

PROGRAMMAZIONE

Termine usato per indicare le attività che trasformano l'esigenza di risolvere un problema in un programma

- utilizzo del computer come esecutore

La programmazione consiste nel

- Ricondurre il problema a problemi primitivi (ovvero eseguibili come insieme di azioni primitive)
- Organizzare e utilizzare le "risorse" del computer

Il programma è il prodotto finale di un lavoro che comincia dalla formulazione del problema e termina con una procedura eseguibile su un computer garantendo risultati in "tempi accettabili"

PROGRAMMA

Il programma è la *traduzione* in un linguaggio comprensibile all'elaboratore dell'algoritmo ovvero della descrizione del procedimento di soluzione, con indicazioni sui dati di ingresso e di uscita.

Tramite il programma si comunica all'elaboratore:

- Quali dati di ingresso deve trattare
- Come deve operare su questi dati
- Quali dati deve dare come risultati

$$P \longleftrightarrow \{ D,A,R \}$$

In conclusione

- Dati: Astrazioni con cui rappresentiamo proprietà o oggetti della realtà
- Algoritmo: descrizione del procedimento di soluzione di un problema, opera su oggetti che sono rappresentazioni simboliche dei dati
- Programma: traduzione in un opportuno linguaggio di programmazione dell'algoritmo e delle fasi di ricevimento e trasmissione dei dati

Corso di Programmazione

Problemi ed Algoritmi

II parte

Prof.ssa Teresa Roselli

teresa.roselli@uniba.it

Processo (d’esecuzione)

- Applicazione di un metodo solutivo ad una situazione problematica
  - Esecuzione delle operazioni da esso previste
- Può essere delegato ad un processore diverso dall’estensore del metodo solutivo
  - Essere umano
  - Sistema meccanico

Processo d'esecuzione
Requisiti per la Delega

- Algoritmo descritto perfettamente all'esecutore in termini di operazioni effettivamente eseguibili
  - Interpretazione non ambigua
  - Comportamento uniforme
- Esecutore meccanico (macchina)
  - Imprescindibilità dalle operazioni eseguibili
    - Operazioni basiche o Azioni primitive

Processi Sequenziali

- L’esecuzione di un’azione non può sovrapporsi all’esecuzione di un’altra
  - Possibilità di prevedere strade alternative da seguire al presentarsi di una certa condizione

Processi Sequenziali

- Per evitare incomprensioni, la descrizione del processo di esecuzione deve definire esattamente
  - Gli oggetti su cui operare
  - La sequenza esatta delle azioni da compiere
    - Prima operazione
    - Ultima operazione
  - La specifica dei controlli che determinano l'ordine di esecuzione delle azioni

indipendentemente dalla natura dell'esecutore

Rappresentazione di Algoritmi
Notazioni

	Grafico	**Strutturato**
Diagrammi di flusso	X	
Linguaggio Lineare		X
Alberi di Decomposizione	X	X
Grafi di Nassi - Schneidermann	X	X

N.B.: NON sono linguaggi di programmazione

Diagrammi di Flusso

- Il linguaggio dei diagrammi di flusso è un linguaggio grafico tipicamente utilizzato per trasmettere ad un esecutore umano la descrizione di un algoritmo o processo
  - Si parte dal punto iniziale
    - Si seguono i percorsi indicati, intraprendendo le azioni che via via si incontrano
    - In caso di percorsi alternativi, se ne sceglie uno a seconda della condizione specificata

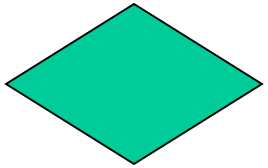
  fino al raggiungimento del punto finale

Diagrammi di Flusso
Elementi Costitutivi

- Operazioni
  - Calcolo (blocco azione)

  - Ingresso/Uscita

  - Decisione



- Controllo
  - Inizio/Fine (limiti del processo)

  - Flusso

  - Connessione

Diagrammi di Flusso
Definizione

E’ un grafo contenente:

- un blocco iniziale
- un blocco finale
- un numero finito di blocchi di azione
- un numero finito di blocchi di controllo

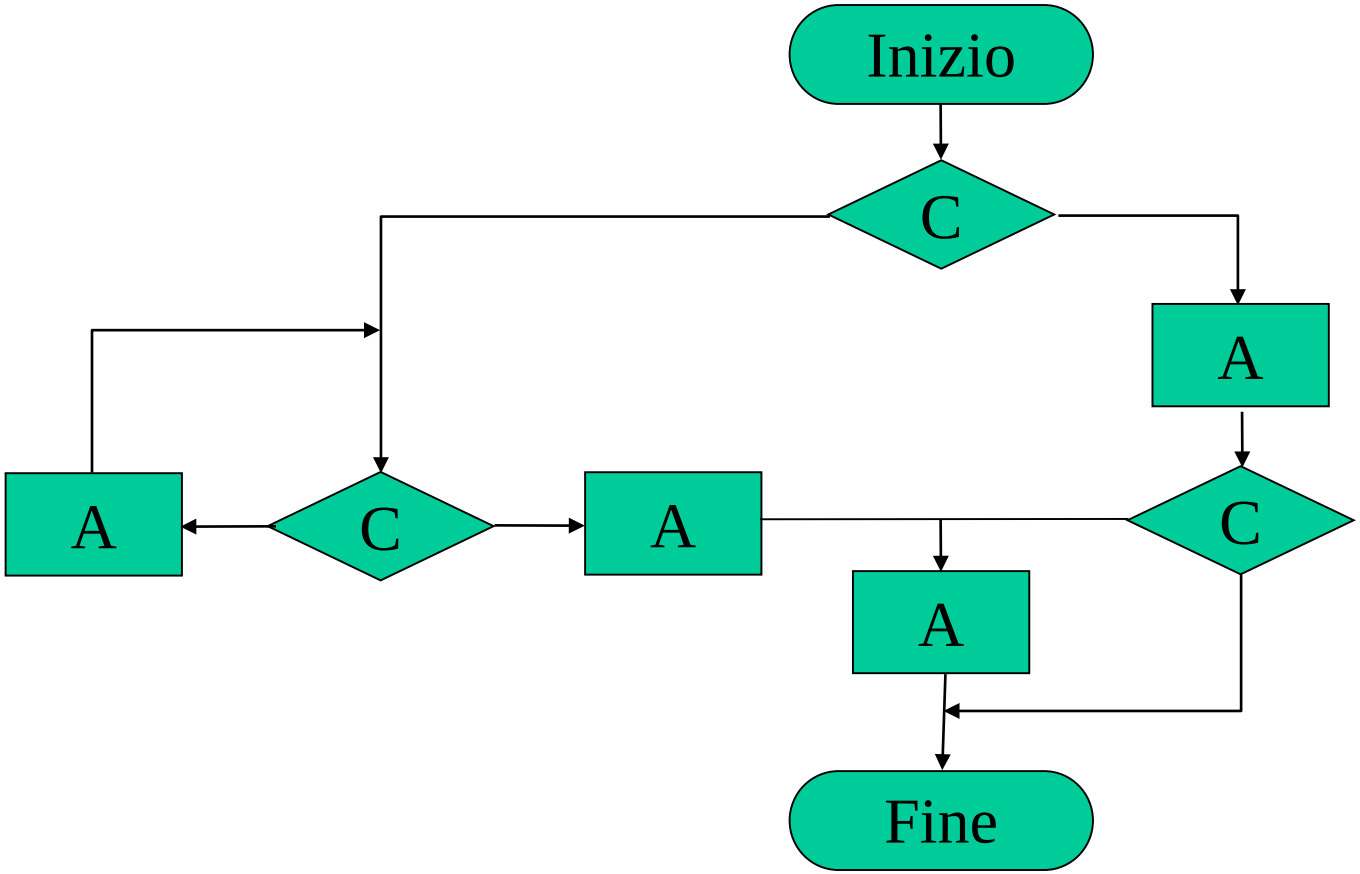
N.B. valgono le regole di costruzione seguenti

Diagrammi di Flusso
Regole di Costruzione

- *Un solo* blocco iniziale e *un solo* blocco finale
  - Ogni blocco è raggiungibile dal blocco iniziale
  - Il blocco finale è raggiungibile da ogni blocco
- I blocchi sono in numero finito
  - Ogni blocco di azione (calcolo o ingresso/uscita) ha *una* freccia entrante ed *una* uscente
  - Ogni blocco di decisione ha *una* freccia entrante e *due* uscenti
- Ogni freccia parte da un blocco e termina in un blocco o su un'altra freccia

Diagrammi di Flusso

Esempio



Diagrammi di Flusso

- La definizione data è costruttiva poiché è basata su regole che consentono di produrre diagrammi di flusso ma anche di riconoscerli

Diagrammi di Flusso
Punti di forza

- Grafici
  - Adatti agli esseri umani
  - Adatti a rappresentare processi sequenziali
  - Immediatamente visualizzabili
- Rispondono all’esigenza di divisione del lavoro
- Documento base per l’analisi organica
- Non ambigui
- Traducibili in vari linguaggi di programmazione

Diagrammi di Flusso
Punti di debolezza

- Spesso non entrano in una pagina
  - Difficili da seguire e modificare
- Non naturalmente strutturati
  - Spesso le modifiche portano a de-strutturazione
- Lontani dai linguaggi dei calcolatori
  - Possono rivelarsi errati in fase di programmazione

Diagrammi di Flusso Strutturati

- Esistono vari modi di connettere blocchi e frecce che rispettino la definizione di diagramma di flusso
- Gli schemi fondamentali o modelli di composizione fondamentali consentono di realizzare ***diagrammi di flusso strutturati***
  - Uso di soli diagrammi strutturati che corrispondono a configurazioni standard di blocchi elementari, comuni a molti processi della vita quotidiana
- Sviluppo per raffinamenti successivi
  - Ogni schema fondamentale ha un solo punto di ingresso e un solo punto di uscita
    - Sostituibile ad un blocco di azione
    - Nella sostituzione, si possono omettere i blocchi di inizio e fine dello schema che si sta inserendo

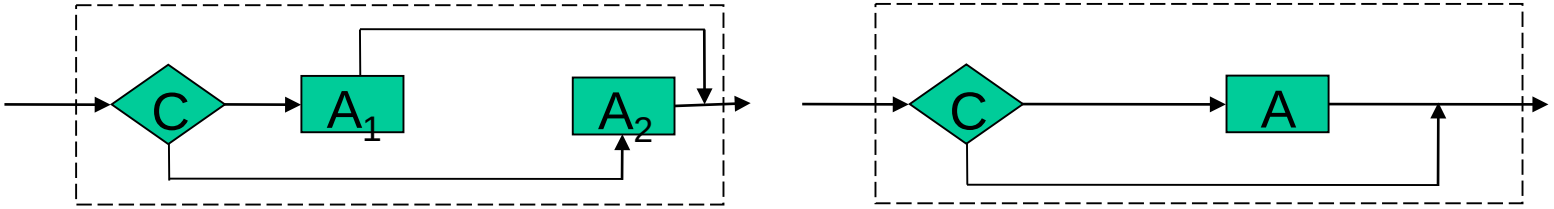
Diagrammi di Flusso Strutturati
Schemi fondamentali

- Sequenza
  - Concatenazione di azioni
- Selezione
  - Scelta di azioni alternative
    - Dipendenza da una condizione
- Iterazione
  - Ripetizione di una certa azione
    - Dati potenzialmente diversi
    - Dipendenza da una condizione

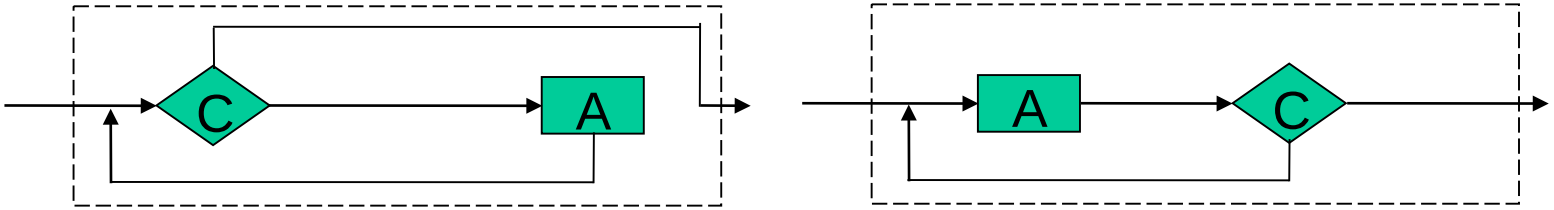
Diagrammi di Flusso Strutturati
Schemi fondamentali

- Sequenza

- Selezione



- Iterazione

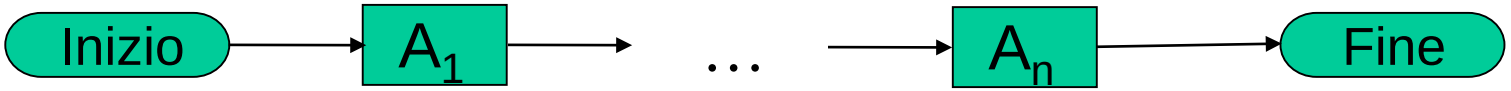


Diagrammi di Flusso Strutturati
Definizione

- **Base**: Dato un blocco di azione A,

è strutturato.

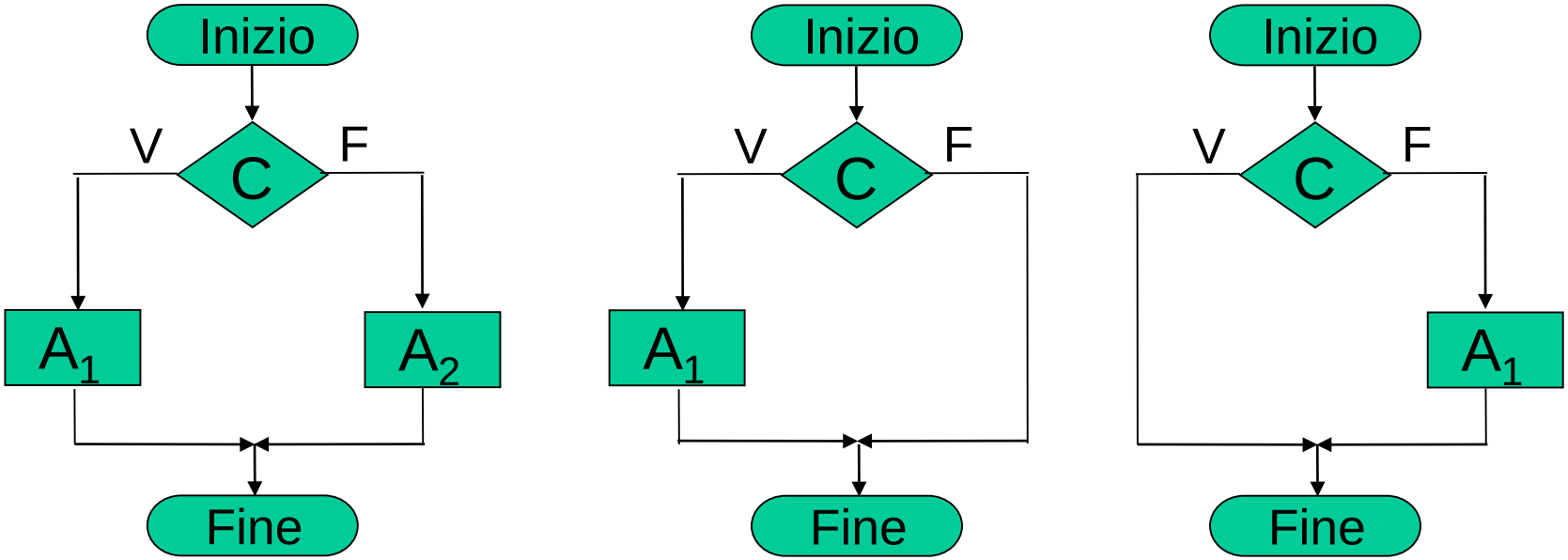
- **Sequenza**: Se A_1, …, A_n sono strutturati,



è strutturato

Diagrammi Strutturati

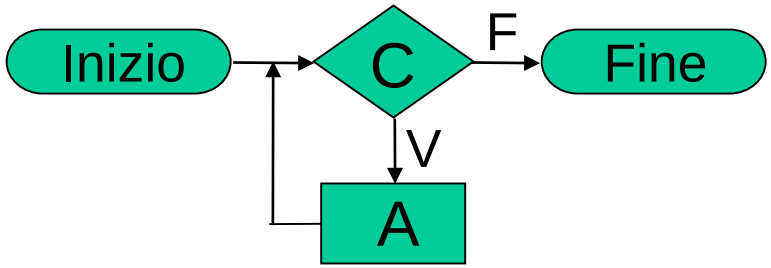
- **Selezione**: Se A_1 e A_2 sono strutturati,

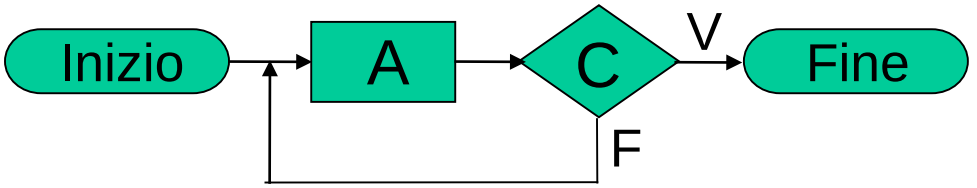


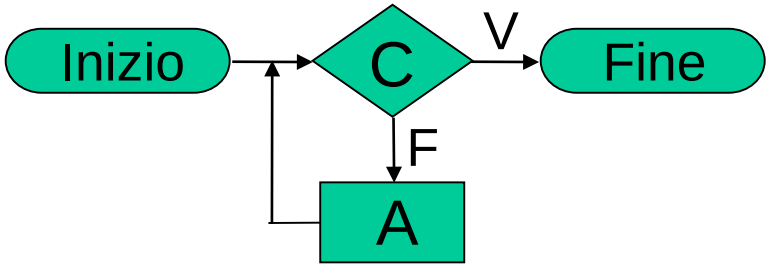
sono strutturati

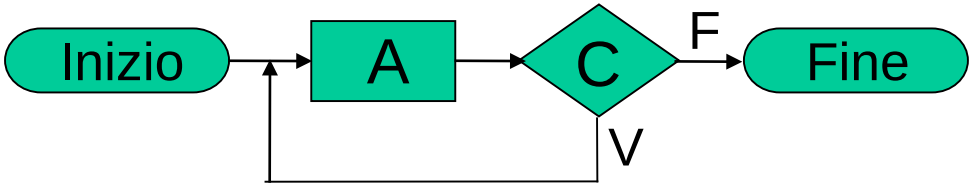
Diagrammi Strutturati

- **Iterazione**: Se A è strutturato,









sono strutturati

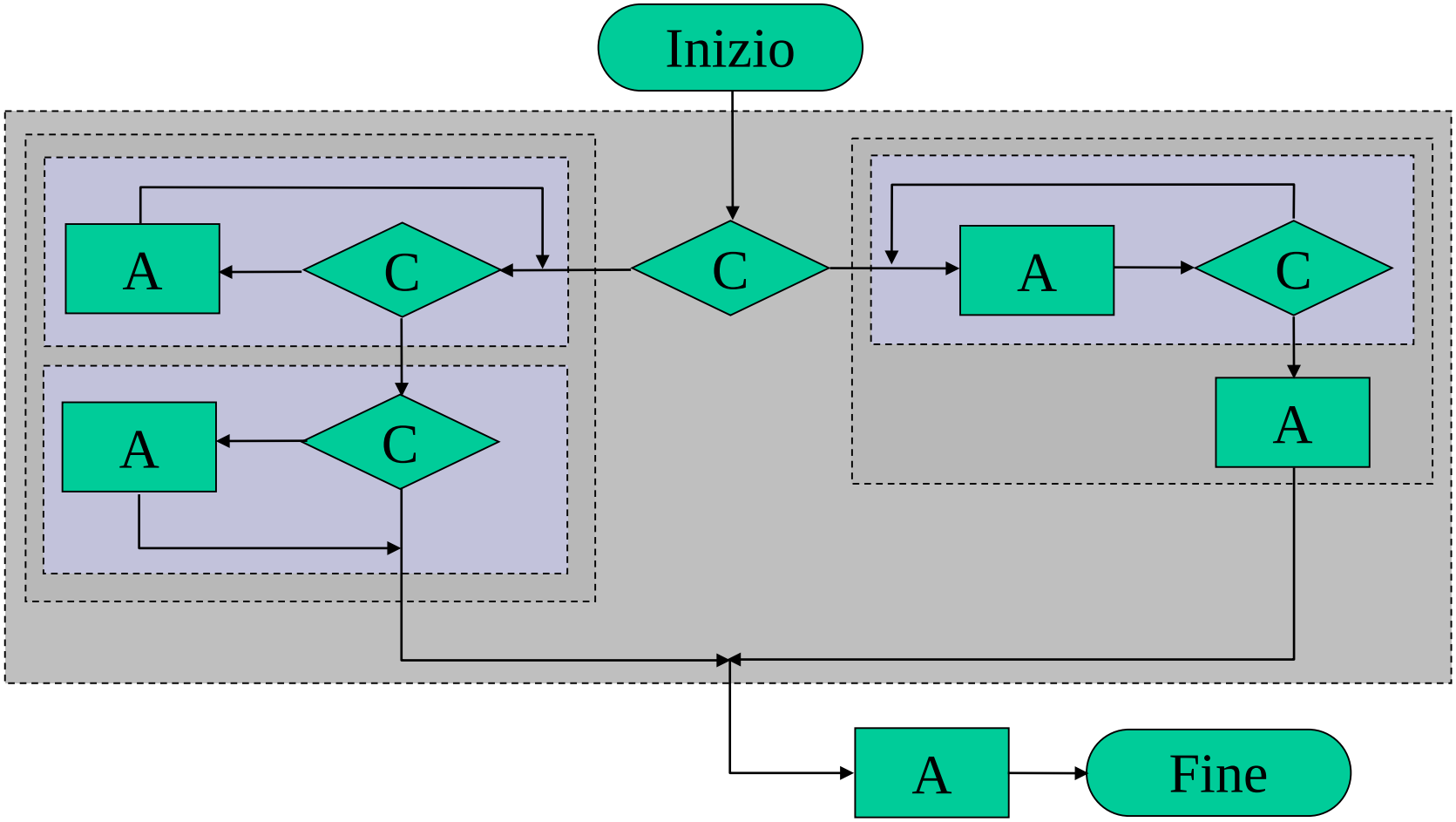
Diagrammi Strutturati

- Nessun altro diagramma è strutturato

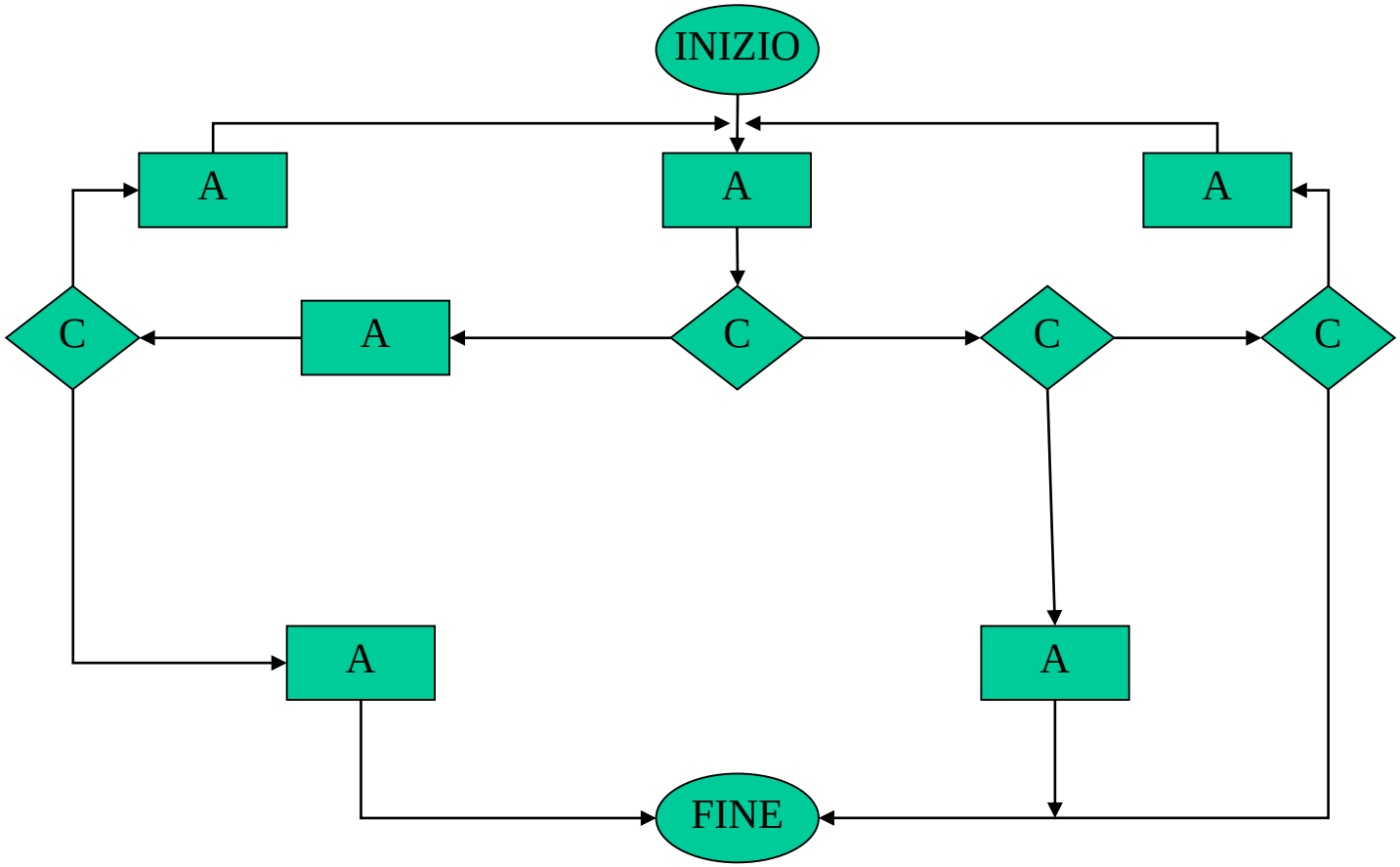
- Note:
  - Definizione ricorsiva

Diagrammi Strutturati

Esempio



Diagrammi non Strutturati
Esempio



Teorema di Böhm-Jacopini

- Dato un processo *P* e un diagramma che lo descrive, è sempre possibile determinare un processo *Q*, equivalente a *P*, che sia descrivibile tramite un **diagramma di flusso strutturato**
- Due processi applicati allo stesso insieme di dati si dicono **equivalenti** se producono lo stesso effetto
- Due processi equivalenti applicati agli stessi dati di ingresso o non terminano o terminano entrambi producendo gli stessi dati di uscita
- Un processo o metodo solutivo può essere sempre descritto tramite diagrammi strutturati

VIETATO
l’uso di istruzioni di salto

- Non necessarie
  - Teorema di Böhm-Jacopini
- Potenzialmente dannose
  - Difficoltà a seguire il flusso del controllo
    - Scarsa modificabilità
    - Interazioni impreviste
  - Goto statement considered harmful [Dijkstra, 68]

Linguaggio Lineare

- Atto alla descrizione di algoritmi
  - Costrutti linguistici non ambigui
- Usa esclusivamente schemi strutturati
- Simile ad un linguaggio di programmazione
  - Sparks [Horowitz, 1978]
    - Corrispondente italiano

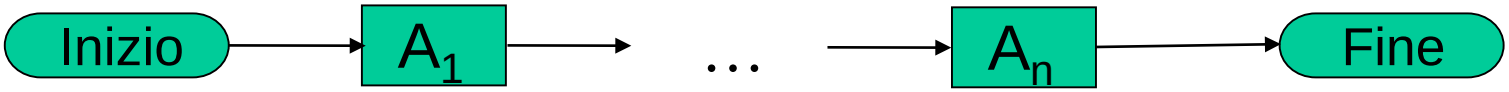
Linguaggio Lineare
Sequenza

- Costrutto base:

  begin A **end**

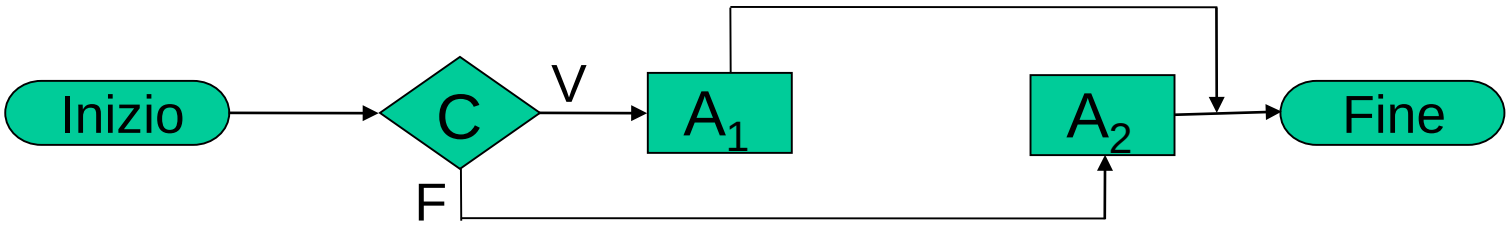
- Inoltre, ogni blocco di azione si può sostituire con uno dei seguenti costrutti

begin A_1 **;** … **;** A_n **end**

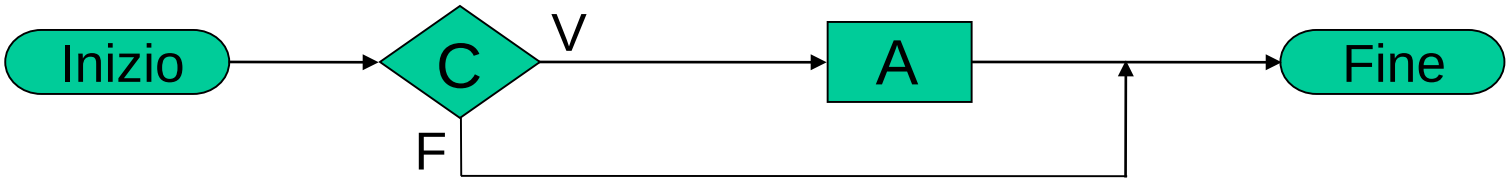


Linguaggio Lineare
Selezione

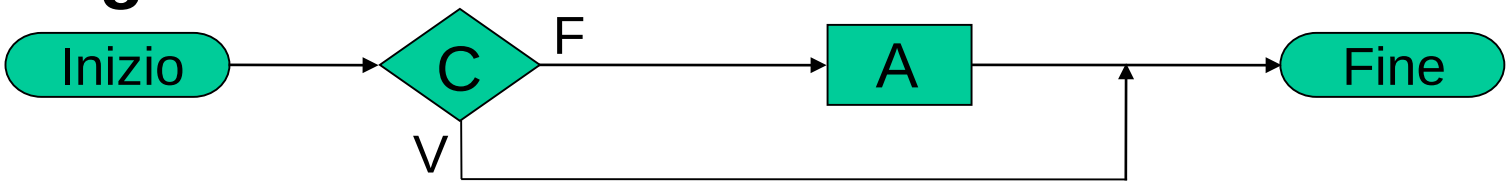
begin **if** C **then** A_1 **else** A_2 **end**



begin **if** C **then** A **end**

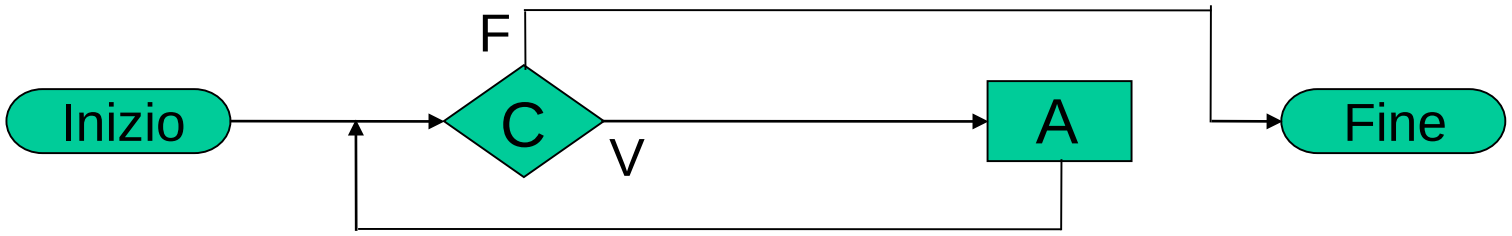


begin **if not** C **then** A **end**

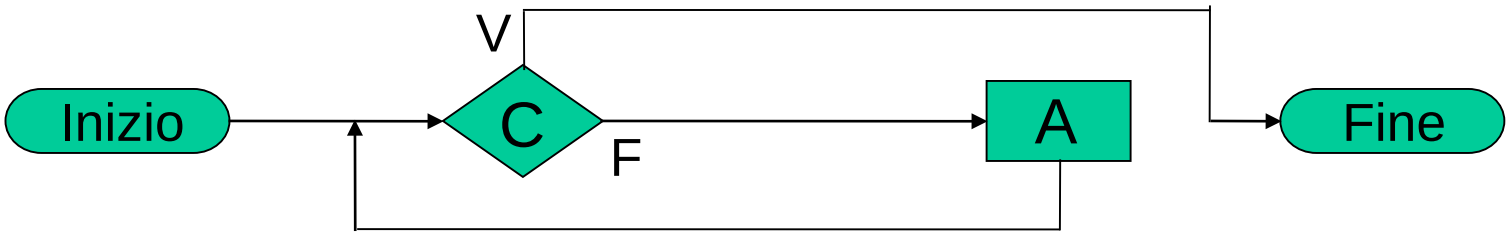


Linguaggio Lineare
Iterazione (while…do)

begin **while** C **do** A **end**

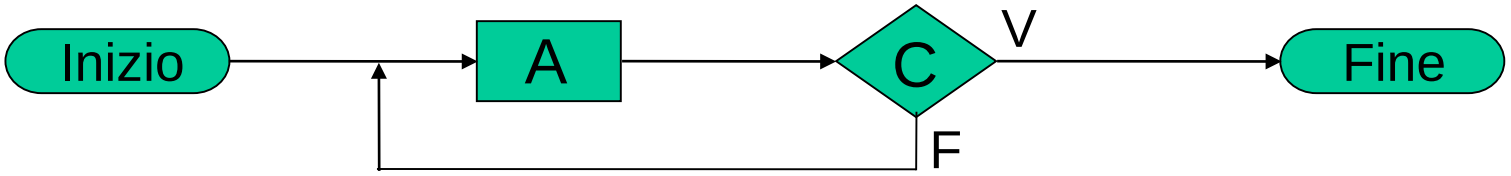


begin **while not** C **do** A **end**

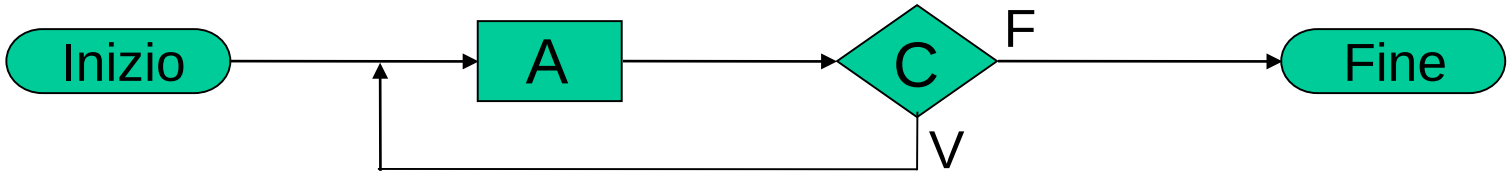


Linguaggio Lineare
Iterazione (repeat…until)

begin **repeat** A **until** C **end**

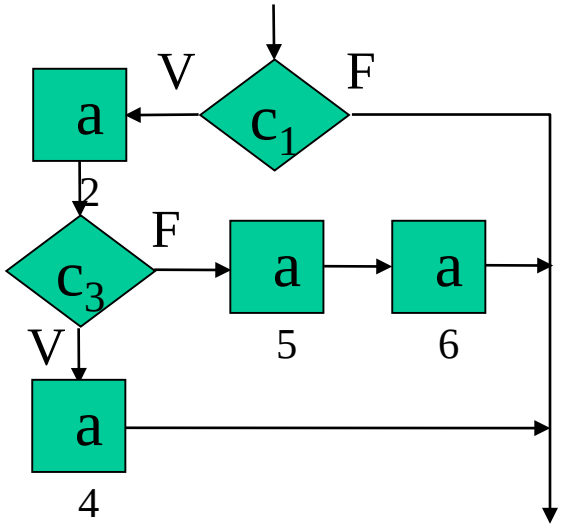


begin **repeat** A **until not** C **end**

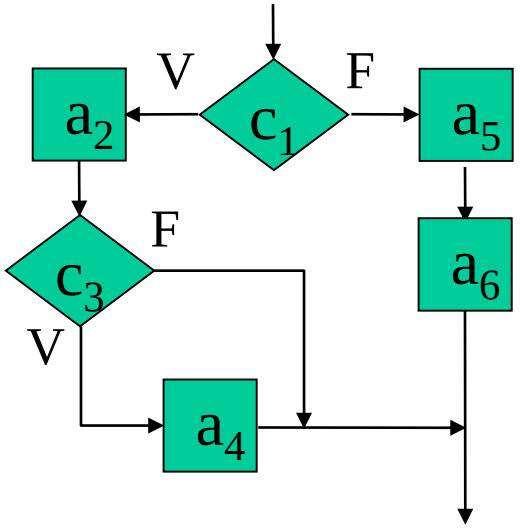


Linguaggio Lineare
Ambiguità

- **if** c_1 **then** a_2; **if** c_3 **then** a_4 **else** a_5; a_6



oppure ?



- Uso dell'indentazione
  - Aiuta ma non risolve

Risoluzione delle Ambiguità
Convenzioni aggiuntive

- Ogni descrizione di un sottoprocesso che sia composizione in sequenza di descrizioni di azioni elementari o sottoprocessi deve essere racchiuso tra le parole **begin – end**
  - Vale, in particolare, per la descrizione di un sottoprocesso (blocco di azioni) che segue le parole chiave
    - **then**
    - **else**
    - **while**

    quando non è un'azione basica

Risoluzione delle Ambiguità
Convenzioni aggiuntive

- Aggiungere i seguenti delimitatori di istruzione
  - Selezione: **endif**
  - Iterazione di tipo *while*: **endwhile**
    - Non necessario per l'iterazione di tipo *repeat*
      - È già presente la clausola *until* come delimitatore

Schemi ridondanti

- Doppio costrutto iterativo
  - Ciascuno dei costrutti **while** e **repeat** è ridondante una volta che sia disponibile l'altro
- Iterazione limitata
  - Basata sulla variazione di un indice di cui sono noti il valore iniziale, il valore finale e l'incremento o passo
- Selezione multipla
  - Basata sul partizionamento dei valori risultanti da una espressione in diverse classi di equivalenza rispetto all'azione da intraprendere

Schemi Ridondanti

```
while C do
    A
endwhile
```

è equivalente a

```
if C then
    repeat
     A
    until not C
endif
```

```
repeat
    A
until C
```

è equivalente a

```
A
while not C do
    A
endwhile
```

Schemi Ridondanti

do varying i
  from expr1 **to** expr2
   A
repeat

i ← expr1
while i ≤ expr2 **do**
  A**;**
  i ← i + 1
endwhile

Se l'incremento o passo non è specificato allora vale 1

Schemi Ridondanti

case espr **of**
    $lista_1$ **:** A_1**;**
    $lista_2$ **:** A_2**;**
    …
    $lista_n$ **:** A_n**;**
else
    A_0
endcase

if espr in $lista_1$
    then A_1
else if espr in $lista_2$
    then A_2
…
else if espr in $lista_n$
    then A_n
else A_0
endif

Schemi ridondanti
Esempio

- Iterazione limitata
  - Quadrato dei primi *n* numeri interi

per i **che va**
  da 0 **a** n
  stampa i * i
ripeti

- Selezione multipla
  - Numero di giorni in un mese

nel caso che mese **sia**
  11,4,6,9 **:** giorni ← 30
  2 **:** giorni ← 28
altrimenti
  giorni ← 31
finecasi

Massimo Comune Divisore

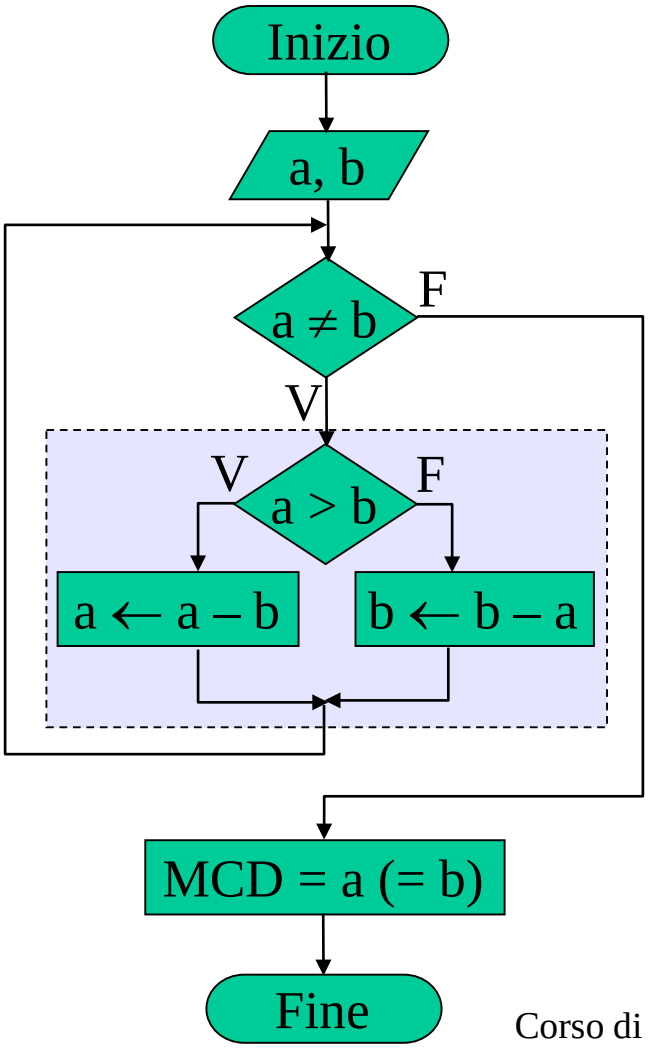
- Il massimo comune divisore può essere calcolato, in linea di principio, determinando la scomposizione in fattori primi dei due numeri dati e moltiplicando i fattori comuni, considerati una sola volta con il loro minimo esponente. Per esempio, per calcolare il MCD(18,84) si scompongono dapprima i due numeri in fattori primi, ottenendo 18 = 2·32 e 84 = 22·3·7, e poi si considerano i fattori comuni ai due numeri, 2 e 3: entrambi compaiono con esponente minimo uguale a 1, e quindi si ottiene che MCD(18,84)=6. Non trovando fattori primi comuni, il MCD è 1, così ad esempio MCD(242,375)=1.

- Un metodo molto più efficiente è fornito dall'algoritmo di Euclide: si divide 84 per 18 ottenendo un quoziente di 4 e un resto di 12. Poi si divide 18 per 12 ottenendo un quoziente di 1 e un resto di 6. Infine si divide 12 per 6 ottenendo un resto di 0, il che significa che 6 è il massimo comun divisore.

Esempio
Algoritmo Euclideo per il MCD

- ***Considera*** la coppia di numeri dati
- **Fintantoché(mentre)** i numeri sono diversi **ripeti**
  - **Se** il primo numero è minore del secondo **allora**
    - Scambiali
  - **Sottrai** il secondo dal primo
  - **Rimpiazza** i due numeri col sottraendo e con la differenza, rispettivamente
- Il ***risultato*** è il valore ottenuto

Esempio
Algoritmo Euclideo per il MCD




```
begin
    leggi a, b
    while (a ≠ b) do
     if (a > b) then
          a ← a – b
     else
          b ← b – a
    endif
    MCD ← a
end
```

a	b
84	18
66	18
48	18
30	18
12	18
12	6
6	6

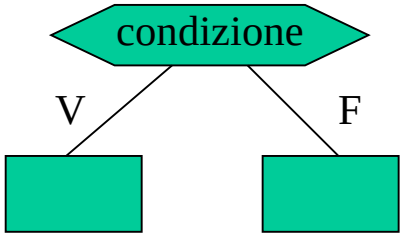
MCD = 6

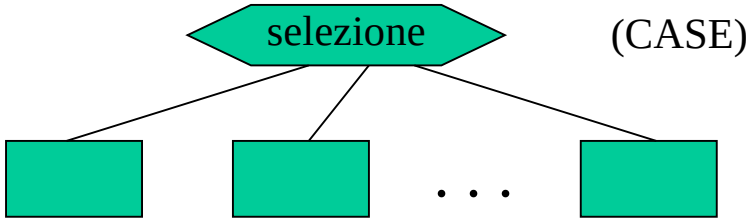
Alberi di Decomposizione

- Rappresenta tramite la relazione padre-figlio la scomposizione di una operazione in operazioni più semplici
  - Più strutturata
  - Consente un'analisi dall'alto verso il basso
    - Adatta alla scomposizione per raffinamenti successivi

Alberi di Decomposizione

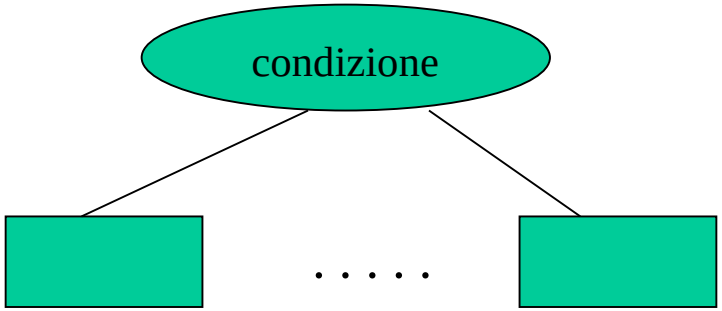
- Sequenza
  - operazioni su uno stesso livello da sinistra verso destra
- Selezione
  - blocco di condizione che si diparte in più strade





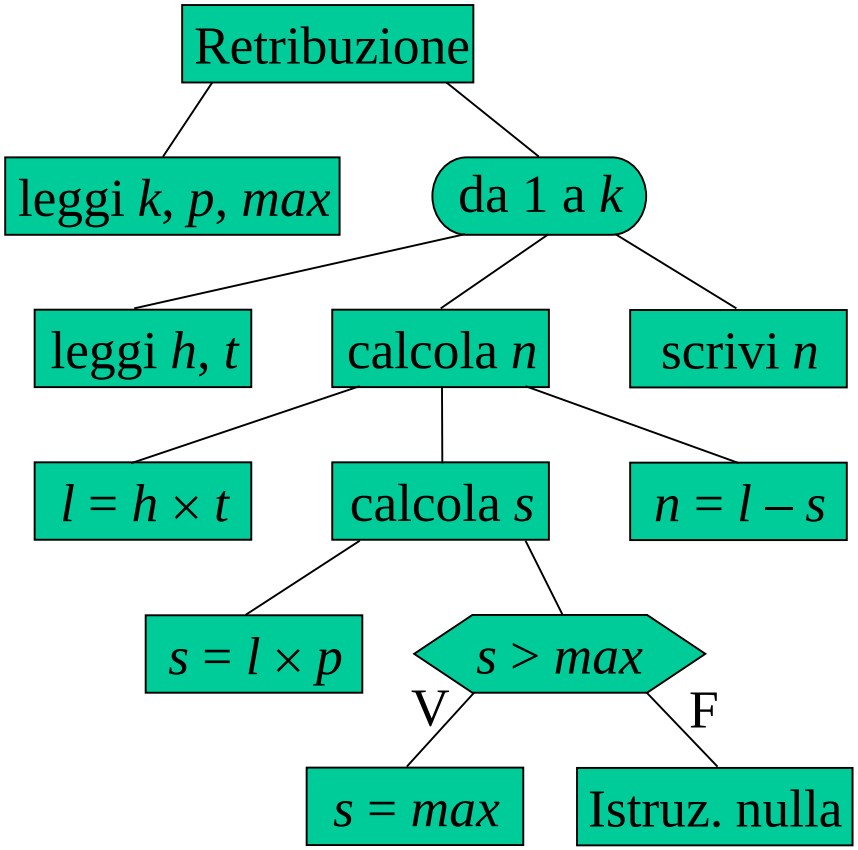
Alberi di Decomposizione

- Iterazione
  - blocco di condizione che controlla la terminazione dei nodi figli



Alberi di Decomposizione
Esempio

- Calcolo retribuzione al netto delle trattenute per *k* individui
  - *t* ore lavorate
  - *h* retribuzione oraria
  - *p* percentuale trattenute su retribuzione base
  - *max* tetto trattenute
  - *n* retribuzione netta
  - *l* retribuzione lorda
  - *s* trattenute calcolate

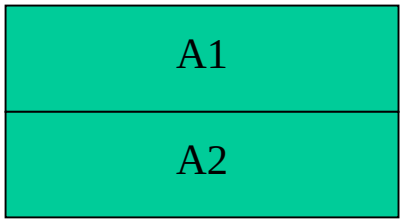


Grafi di Nassi-Schneidermann

- Uniscono il vantaggio di una rappresentazione grafica con quello di poter rappresentare schematicamente metodi strutturati

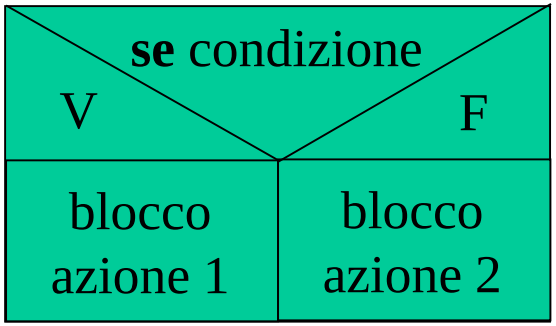
SCHEMI

- Sequenza

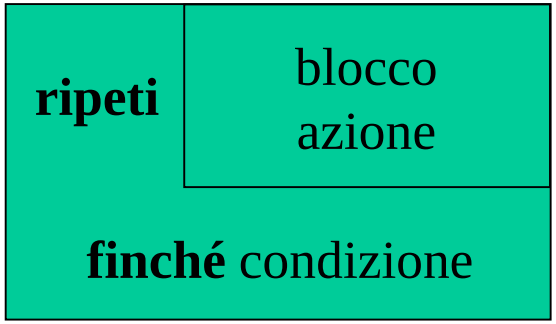


Grafi di Nassi-Schneidermann
Schemi

- Selezione

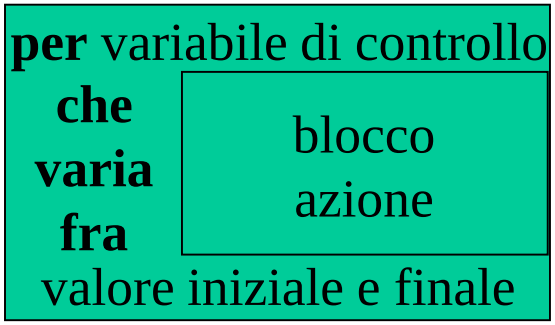


- Iterazione repeat



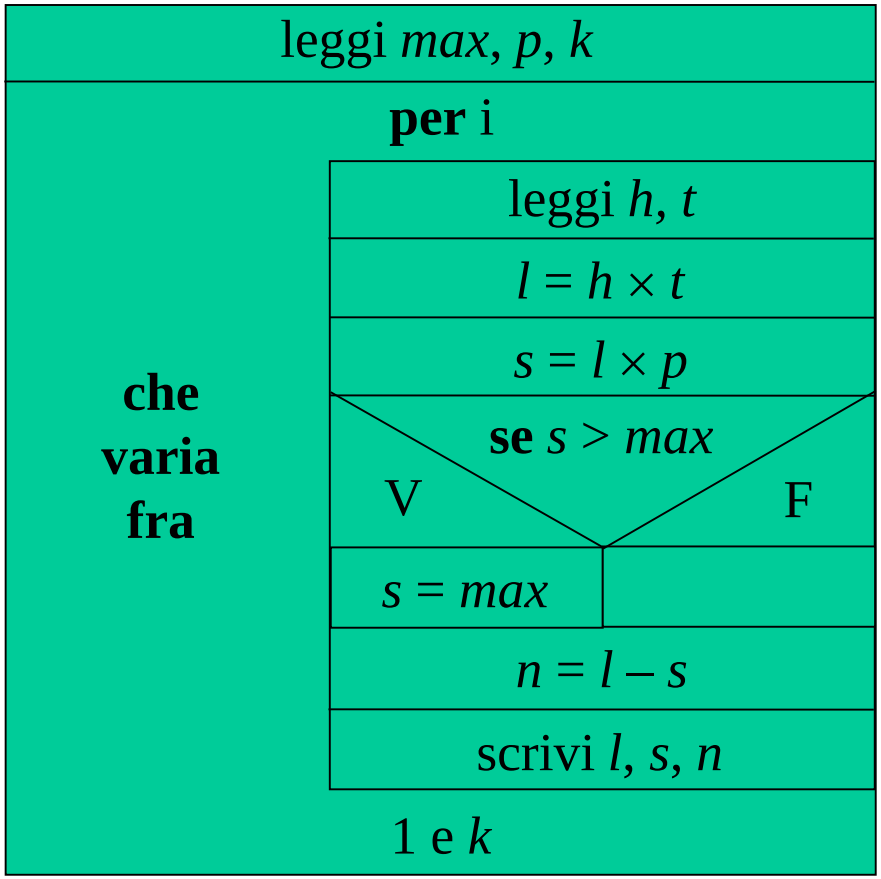
- Iterazione while

- Iterazione limitata



Grafi di Nassi-Schneidermann
Esempio

- Calcolo retribuzione al netto delle trattenute per *k* individui
  - *t* ore lavorate
  - *h* retribuzione oraria
  - *p* percentuale trattenute su retribuzione base
  - *max* tetto trattenute
  - *n* retribuzione netta
  - *l* retribuzione lorda
  - *s* trattenute calcolate



Corso di Programmazione
Problem solving: Fasi di Sviluppo di un programma

Prof. ssa Teresa Roselli
teresa.roselli@uniba.it

Sviluppo di un Programma

- Risoluzione di un problema mediante un calcolatore
  - Partire dalla descrizione del problema
    - In linguaggio naturale

    per giungere alla stesura di un programma
    - In un certo linguaggio di programmazione

Sviluppo di un Programma
Fasi

- (Studio di fattibilità)
- Analisi
  - Chiarifica del problema
    - Cosa
- Progettazione
  - Individuazione di una strategia di soluzione
    - Come
  - Scelta delle strutture di dati

Sviluppo di un Programma
Fasi

- Codifica
  - Scrittura del programma
- Verifica (e correzione)
  - Test del programma
    - Rimanda ad una delle fasi precedenti
- Manutenzione
  - Correttiva
  - Adattativa
  - Migliorativa

Sviluppo di un Programma

- Accumulo degli errori
  - Gli errori in ciascuna fase si ripercuotono su tutte le fasi successive
    - Più costosi
  - Ciascuna fase può aggiungere errori a quelli delle fasi precedenti
- Vari approcci
  - Differente sequenza di esecuzione delle fasi di sviluppo

Sviluppo di un Programma

- Documentazione di ogni fase
  - Necessaria per chi riprenderà in seguito il programma per modificarlo
    - L'autore stesso
    - Altre persone
- La documentazione in output da ciascuna fase è input per la fase successiva

Problema
Parti principali

- *Incognita*/e o *Obiettivo*/i
  - Ciò che si vuole trovare
- *Dati*
  - Ciò che è dato o conosciuto
- *Condizione*
  - Specifica come l'incognita è connessa ai dati
    - Per mezzo di quali relazioni

Problema

- Comprendere il problema
  - Il suo significato
  - Il suo scopo
- Vederne molto chiaramente le parti principali
  - Incognita
  - Condizioni
  - Dati

Problema

- Scopo di un problema
  - Trovare
    - Trovare, produrre, costruire, identificare, elencare, caratterizzare, …

    un certo oggetto
    - L'*incognita* del problema

    che soddisfa la *condizione* del problema
    - Collega l'incognita ai *dati* del problema

Formulazione del Problema

- I problemi reali non sempre sono ben formulati
  - I dati
    - Sono sufficienti?
    - Sono sovrabbondanti?
  - La condizione
    - È sufficiente, è sovrabbondante?
    - È contraddittoria?

Formulazione del Problema

- Caso di problemi perfettamente determinati (ad es. un teorema)
  - Tutte le affermazioni contenute nell'ipotesi del teorema sono essenziali per dimostrare la tesi
    - Se la dimostrazione non tiene conto di una qualsiasi parte dell'ipotesi è senz'altro errata
    - Analogamente: se una parte della condizione di un problema che "chiede di trovare" non viene presa in considerazione, si finisce col risolvere un problema diverso da quello originale

Formulazione del Problema

- Un problema *chiaramente enunciato* deve specificare
  - La *categoria* a cui appartiene l'incognita
    - Che genere di cosa bisogna trovare
      - Esempio: un numero, una parola, ecc.
  - La *condizione* che deve soddisfare l'incognita
    - Il sottoinsieme di quegli oggetti che soddisfano la condizione rappresenta le soluzioni
  - I *dati*

Formulazione del Problema

- Quasi tutti i problemi sono mal formulati
  - Non chiaramente enunciati
    - Omissioni
    - Ambiguità
    - Imprecisioni
- Possibilità di non capire bene le richieste del problema
  - Necessità di esplicitare le ipotesi

Formulazione del Problema
Esempio

- Problema
  - Calcolare l'*altezza* di uno stabile avente *n* piani di *h* metri
- Obiettivo
  - Trovare il valore dell'*altezza complessiva* dello stabile
- Dati
  - *Numero di piani* dello stabile (*n*)
  - *Altezza dei piani* (*h*)

Formulazione del Problema
Esempio

- *Imprecisione* ed *ambiguità*
  - Nell'obiettivo (che rappresenta l'incognita)
  - Nei dati
- L'*incognita* è un valore numerico (espressa nella stessa unità di misura dei valori in ingresso) che rappresenta l'altezza complessiva di uno stabile

Formulazione del Problema
Esempio

- Cosa si intende per *altezza complessiva*?
  - la distanza fra suolo e parapetto del terrazzo?
  - o questo è escluso?
- Cosa si intende per numero di piani?
  - il piano terra costituisce un piano?
  - ed è compreso nel dato *n* fornito?
- Cosa si intende per *altezza dei piani*?
  - la distanza che intercorre fra pavimento e soffitto?
  - o fra due solai successivi?

Formulazione del Problema

- Tutti i problemi enunciati a parole contengono delle ipotesi semplificatrici sottintese
  - Necessità di un certo lavoro preliminare di interpretazione e di astrazione da parte di colui che risolve il problema
    - In particolare questo avviene quando si parte da un problema relativo ad oggetti reali per ricondurlo ad un problema matematico

Formulazione del Problema
Esempio

- Un aeroplano di pattuglia percorre 220 miglia all'ora in atmosfera tranquilla. Esso trasporta benzina per 4 ore di volo sicuro. Se decolla in pattuglia contro un vento di 20 miglia orarie, di quanto può allontanarsi per ritornare sano e salvo?
  - È sottinteso:
    - Che si suppone che il vento continui a soffiare con la stessa intensità durante tutto il volo
    - Che l'aeroplano voli in linea retta
    - Che il tempo necessario per cambiare direzione nel punto estremo sia trascurabile
    - …e così via

Formulazione di un Problema

- NON iniziare a risolvere un problema prima di averlo *capito*
  - Lo si è capito quando si è in grado di riformulare il problema indicando chiaramente
    - Le *incognite*
    - I *dati*

    e spiegando
    - La *condizione* (ovvero l'insieme delle relazioni che collegano i dati con le incognite)

Chiarifica

- Riformulazione del problema – considerandolo risolto – cercando di vedere con chiarezza, in ordine conveniente, tutte le relazioni che devono intercorrere fra le incognite ed i dati, al fine di soddisfare l'obiettivo del problema.
- Tutte queste relazioni rappresentano la condizione del problema.
  - Se *n* è il numero delle incognite, ottenere *n* equazioni

Chiarifica
Esempio

- Un fattore ha polli e conigli. Questi animali hanno 50 teste e 140 zampe. Quanti polli e quanti conigli ha il fattore?
  - Bisogna trovare 2 numeri, *p* e *c* *(incognite)*, che rappresentano il numero dei polli e il numero dei conigli rispettivamente
  - Sono forniti il numero delle teste (50) ed il numero delle zampe (140) *(dati)*

$$\left.\begin{aligned} p + c &= 50 \\ 2p + 4c &= 140 \end{aligned}\right\} \quad \textit{condizione}$$

- 2 = nro zampe per ciascun pollo
- 4 = nro zampe per ciascun coniglio

} Ulteriori dati emersi dall'analisi del problema

Chiarifica

- Il chiarimento del problema avviene
  - Se chi propone il problema è disponibile
    - Si pongono domande puntuali riguardo lo scopo del problema, i dati e le relazioni intercorrenti tra incognita e dati
  - Altrimenti facendo opportune ipotesi
    - definendo dei campioni

Chiarifica

- La fase di chiarifica
  - PUÒ raffinare e definire meglio, eventualmente ricorrendo a delle ipotesi semplificative, quanto non esplicitamente presente nella traccia del problema come fornita dal committente
  - NON DEVE disattendere o contravvenire a quanto esplicitamente riportato nella traccia del problema come fornita dal committente

Chiarifica

- Generalmente per essere in grado di comprendere il problema bisogna avere delle conoscenze pertinenti
  - Dominio del problema
  - Esempio: problemi geometrici
    - Il teorema di Pitagora, la proporzionalità dei lati nei triangoli simili, formule per aree e volumi, ecc.
    - In generale si fa ricorso alle definizioni

Analisi

- Input: Traccia del problema
- Processo: Chiarire con precisione *cosa* vuole il problema (non *come* va fatto)
  - Obiettivo del problema
    - Dati a disposizione
    - Risultati desiderati
  - Ambiguità e imprecisioni nella definizione del problema
    - Obiettivo, dati
    - Possibilità di capire male le richieste
- Output: Descrizione dettagliata discorsiva del problema

Analisi
Punti chiave

- Dati (input)
  - Formato o tipo, ordine, limiti sui valori, limiti sul volume, fine dei dati, ipotesi di ordinamento
- Risultati (output)
  - Contenuto, formato, ordine, limite sul volume
- Errori e Casi limite
  - Tipi di errori (di input e/o di elaborazione) ed azioni da intraprendere
  - Campioni di input e di output corrispondenti

Analisi
Ambiguità

- Alcune informazioni su cui è necessario fare delle assunzioni
  - Ordine dell'input
    - 8/7 = 8 luglio o 7 agosto?
  - Limiti sui valori
    - 0 < età < 125
  - Errore di elaborazione

    $\forall \Delta < 0$ nelle equazioni di II grado

Analisi
Esempio

- Data una lista di numeri, stampare il 1° numero della lista, il 2° e così via fino al più grande dei valori presenti nella lista

  3 8 2 *25* 13 19

- Necessità di una fase di chiarifica del problema

Analisi
Esempio

- Input: lista di numeri
  - Insieme composto da non più di 100 valori interi positivi (>0).
    - Un valore fittizio nullo, aggiunto in coda all'insieme, definisce la fine dei dati
      - 0 non fa parte dei valori
  - I dati di ingresso non sono ordinati rispetto al valore
- Campione di Ingresso: 1 7 3 9 5 8

Analisi
Esempio

- Output: elenco di valori
  - Si vuole la stampa di una *colonna* di numeri che corrispondono ai numeri di ingresso nello *stesso ordine*
  - La colonna inizia con il primo numero e termina quando l'ultimo numero stampato è il più grande dell'intero insieme di ingresso
  - Verranno stampati *al massimo* un numero di valori *pari alla numerosità* dell'insieme di input se il più grande è l'ultimo
- Campione di Uscita: 1 7 3 9 (incolonnati)

Analisi
Esempio

- Input
  - Quali numeri?
    - Reali, interi
  - Limiti sui valori?
    - Qual è il minimo ammissibile? Qual è il massimo?
      - Ad es., se rappresentassimo età non avrebbero senso dei valori negativi
      - Se fossero anni di nascita sarebbero fuori range tutti i valori > 2002
  - Limiti sul volume?
    - Quanti al massimo potranno essere i dati?

Analisi
Esempio

- Come si riconosce la fine dei dati?
- Sono ordinati?
  - Qual è il criterio di ordinamento?
    - Crescente, decrescente, non crescente, non decrescente
    - Lessicografico, …
  - Rispetto a cosa (per dati non atomici)?
    - Cognome, anno di nascita, …

- Controlli sui dati in ingresso
  - Da realizzarsi nel programma

Analisi
Esempio

- Errori e casi limite
  - Valori negativi
    - Scartare il dato, oppure
    - Prendere il valore assoluto
  - Troppi dati (più di 100)
    - Prendere in considerazione solo i primi 100
  - Presenza di più massimi
    - Fermarsi in output alla prima occorrenza del massimo

Analisi
Esempio

- Mancanza di dati (lista vuota)
  - Prevedere il controllo adeguato da programma
- Mancanza del flag di fine dati
  - Non rilevabile da programma

- N.B.:
  - Per ogni situazione di errore individuata va almeno inviato un messaggio di notifica all'utente

Progettazione

- Si occupa di definire una strategia di soluzione che porti ad ottenere il risultato desiderato
  - Necessita di una definizione chiara del *cosa* si vuole ottenere
- Organizza la soluzione in un insieme di moduli cooperanti
  - Tecniche basate sul metodo di soluzione di problemi consistente nello scomporre un problema in sottoproblemi più semplici

Progettazione
Tecniche di Sviluppo

- Top-Down
  - Raffinamento successivo della procedura di soluzione
  - Raffinamento della descrizione dei dati
- Bottom-Up
  - Sfrutta algoritmi codificati già esistenti raggruppandoli per adattarli a nuove situazioni
- Ibrida
  - Basata su una cooperazione fra le tecniche precedenti

Progettazione

- Input: Analisi del problema
- Processo: Chiarire con precisione *come* va ottenuto lo scopo voluto dal problema
  - Strategia di soluzione
  - Strutture di dati
- Output: Descrizione dettagliata della struttura della soluzione

Progettazione

- Individuare una possibile scomposizione in sottoproblemi più semplici
- Descrivere un procedimento risolutivo per ciascun sottoproblema
- Riportare l'algoritmo per il problema originario
  - Attenzione: l'algoritmo dipende sempre dal modo in cui sono organizzati i dati

Progettazione
Esempio

- Problema
  - Trovare in una agendina il numero telefonico di una persona di cui sia noto il cognome
  - *Dati di ingresso*
    - Agendina e cognome
  - *Dati di uscita*
    - Numero telefonico
      - Caso particolare: se nell'agenda non esiste quel cognome? Risposta "non trovato"

Progettazione
Esempio

- Organizzazione di un’agendina
  - Pagine consecutive
  - Ordinate alfabeticamente in base all’intestazione
    - Una singola lettera alfabetica
    - Un gruppo di lettere consecutive
  - Sono presenti tutte le 26 lettere!

Progettazione
Esempio

- Ogni pagina contiene cognomi ed i corrispondenti numeri telefonici relativi alla propria intestazione
  - All'interno di ciascuna pagina, i cognomi non sono generalmente ordinati ma raggruppati
  - Esempio: **GH**
    - Gruppo dei cognomi che iniziano con G o con H
    - Non ordinati

Progettazione
Esempio

- Descrivere il metodo da usare per risolvere questo problema
  - Algoritmo di ricerca in una agendina
    - Occorre:
      - Pensare "*come*" si effettua la ricerca in un'agendina telefonica
      - Dare una descrizione *sistematica*

Corso di Programmazione

Progettazione di programmi

Prof.ssa Teresa Roselli
teresa.roselli@uniba.it

Tipi di Problemi

- Semplici: facile individuazione di algoritmi
  - Risolubili tramite
    - Un’azione primitiva o una sequenza di azioni primitive

- Complessi: difficilmente la soluzione è data pensando al problema nella sua interezza
  - Non risolubili tramite un’azione nota
    - Al solutore
    - All’esecutore

Tipi di Problemi

- La progettazione di un algoritmo non è immediata, avviene alternando fasi di analisi e scelte realizzative.

- Da un'analisi generale del problema si passa ad una soluzione a grandi linee e, per gradi, si procede alla costruzione della soluzione finale specificando un numero sempre maggiore di dettagli.

Problemi Complessi
Soluzione

- Scomposizione in sottoproblemi
  - Fino a trovare un insieme di sottoproblemi primitivi che risulti equivalente al problema di partenza
- Individuazione del procedimento che porta alla soluzione
  - Processo di cooperazione tra sottoproblemi dei quali si è in grado di fornire facilmente una soluzione

Problemi Complessi
Approccio

- Principio del *Divide et Impera*
  - Ridurre la complessità del problema
    - Scomposizione in sottoproblemi più semplici
      - Individuare Struttura e Relazioni fra di essi
    - Progettazione di un algoritmo per ciascun sottoproblema
      - Se è complesso, riapplicare la scomposizione

    fino ad arrivare a problemi primitivi
    - Risolubili tramite azioni note

Scomposizione di Problemi

- Tecnica per raffinamenti successivi

  (STEP-WISE REFINEMENT o TOP DOWN DESIGN)

  - Uno dei fondamenti della programmazione strutturata
  - Trasformazione di un problema in una *gerarchia di problemi* di difficoltà decrescente
    - Di un problema si affronta, prima, l'aspetto più generale e si passa, poi, a livelli sempre più dettagliati di descrizione sino ad arrivare agli elementi fondamentali

Scomposizione di Problemi

- Scomporre il problema originario in sottoproblemi più semplici
- Affrontare un sottoproblema per volta

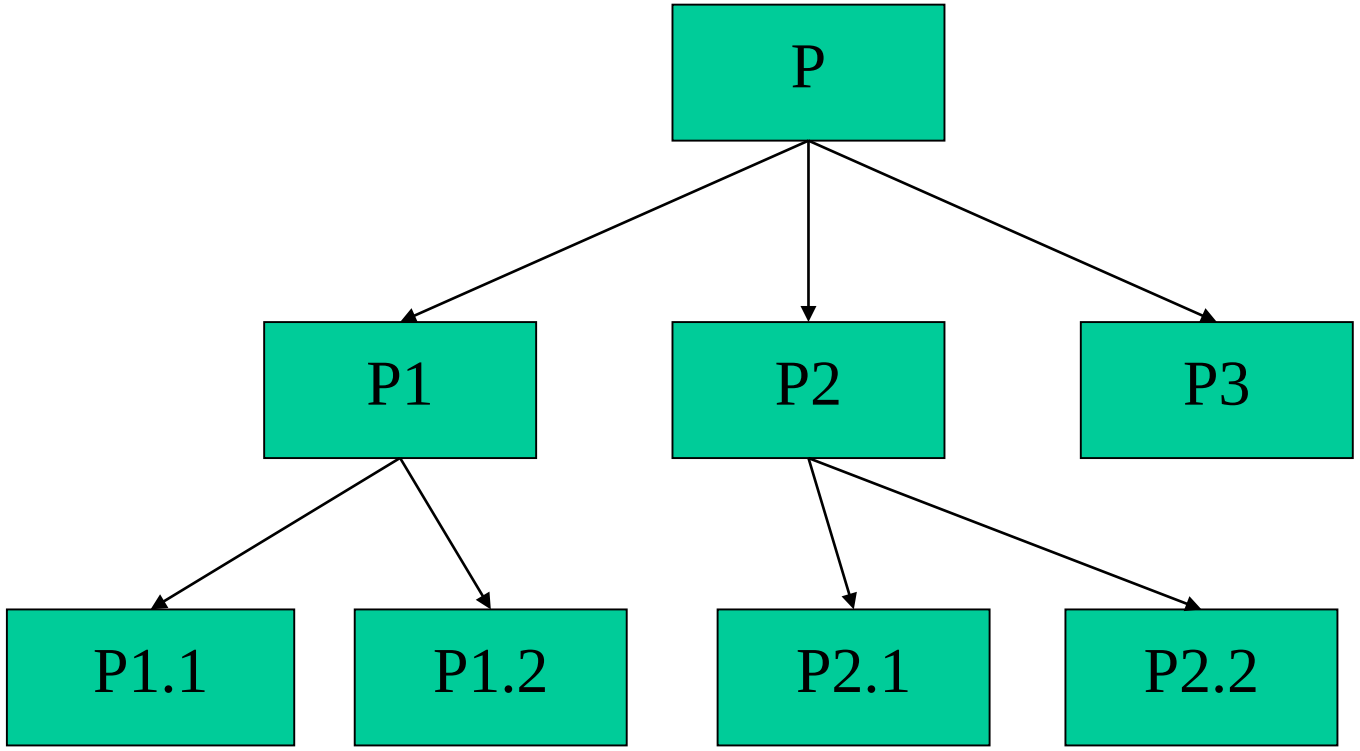
TOP — alto livello di descrizione

DOWN — basso livello di descrizione

Scomposizione di Problemi

Nel processo di raffinamento inizialmente l'attenzione è rivolta a *cosa* poi, man mano, raffinando si passa a *come* ovvero ad un algoritmo che indichi *come fare per ottenere cosa*.

Scomposizione di Problemi
Albero dello sviluppo TOP-DOWN



I scomposizione

II scomposizione

Ad ogni passo di scomposizione ci si allontana dal linguaggio naturale (ad alto livello) e ci si avvicina alla descrizione nel linguaggio di programmazione: uso del linguaggio lineare

Scomposizione di Problemi

- Conseguenze
  - Aumento del numero problemi
  - Diminuzione della complessità di ciascuno
- Può capitare che il medesimo sottoproblema debba venire risolto più volte, applicandolo a dati diversi, per risolvere il problema complessivo

Scomposizione di Problemi
Requisiti

- Ogni passo della suddivisione o specificazione deve garantire che
  - La soluzione dei sottoproblemi conduca alla soluzione generale
  - La successione di passi da eseguire abbia senso e sia possibile
  - La suddivisione dia luogo a sottoproblemi “più vicini” agli strumenti disponibili
    - Risolubili direttamente con gli operatori a disposizione

Scomposizione di Problemi
Quando fermarsi?

- Bisogna arrivare al punto in cui
  - Tutti i problemi sono primitivi
  - È fissato l'ordine di soluzione dei sottoproblemi
  - È previsto il modo in cui un sottoproblema utilizza i risultati prodotti dalla soluzione dei sottoproblemi che lo precedono
    - Livello di cooperazione

Scomposizione di Problemi

- I problemi ottenuti dalla scomposizione sono
  - Indipendenti
    - Si può progettare un algoritmo per ciascuno
    - Soluzione delegabile a solutori diversi (in parallelo)
  - Cooperanti
    - Ciascuno può usare il risultato della soluzione di altri
- Quanto più complesso è il problema, tanti più livelli saranno necessari in profondità
  - Di solito 2 o 3
- Uno stesso problema può essere scomposto in modi diversi

Scomposizione di Problemi

- I problemi ottenuti dalla scomposizione sono
  - Indipendenti
    - Si può progettare un algoritmo per ciascuno
    - Soluzione delegabile a solutori diversi (in parallelo)
  - Cooperanti
    - Ciascuno può usare il risultato della soluzione di altri
- Quanto più complesso è il problema, tanti più livelli saranno necessari in profondità
  - Di solito 2 o 3
- Uno stesso problema può essere scomposto in modi diversi

Progettazione

Esempio della ricerca in una agendina telefonica

- Scomposizione del problema in sottoproblemi più semplici tali che l'insieme dei sottoproblemi sia equivalente al problema di partenza

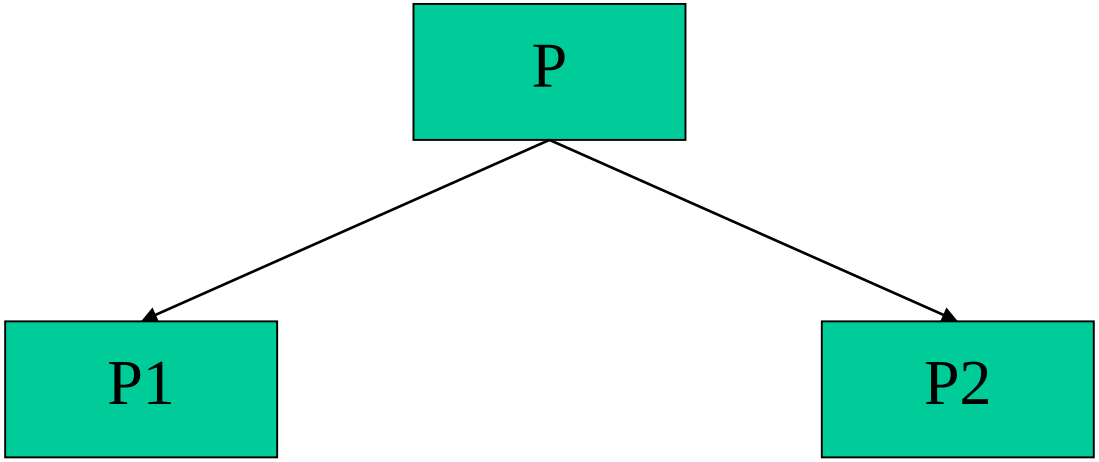
  P1. Trovare la pagina dell'agendina in cui si dovrebbe trovare (se c'è) il cognome dato

  P2. Ricercare il cognome dato nella pagina trovata

- Risoluzione dei sottoproblemi individuati
  - Per quanto possa essere chiara la *formulazione* di un problema, essa non fornisce, in genere, un *metodo* che consenta di ottenere il risultato partendo dai *dati iniziali*!

Progettazione
Esempio

Albero dello sviluppo TOP-DOWN



Progettazione
Esempio

- Sottoproblema 1
  - Si tratta di un problema di ricerca in un insieme di pagine ordinato alfabeticamente
  - L'insieme delle pagine che costituiscono l'agendina è piccolo
  - La scansione è sequenziale
    - Si sfogliano tutte le pagine, una dopo l'altra, a partire dalla prima
  - È una ricerca di sicuro successo
    - La pagina cercata esiste sicuramente nell'agendina

Progettazione
Esempio

- *Come* controlliamo se la pagina attuale è proprio la pagina cercata?
    - In tal caso avremmo raggiunto il nostro obiettivo (termine della ricerca)
- *Sfogliare una pagina alla volta, iniziando dalla prima pagina, fino a che la pagina attuale contiene nell'intestazione la lettera iniziale del cognome cercato*
  - Confrontare la lettera iniziale del cognome con la lettera o il gruppo di lettere contenute nell'intestazione della pagina attuale

Progettazione
Esempio

- Individuazione della pagina in cui cercare il cognome cercato
  - A B C
  - D E F ← se il cognome è DANGELO,
  - G H I J termina la scansione
  - K L M dell'agendina alla ricerca della
  - N O P pagina
  - Q R S
  - T U V
  - W X Y Z

Progettazione
Esempio

- Entità da rappresentare
  - Sequenza di pagine
    - 2 componenti
      - Intestazione
      - Insieme di righi
  - Ogni pagina ha un insieme di righi
    - Tutti con la stessa struttura
  - Ogni rigo ha le stesse suddivisioni
    - Ciascuna destinata ad un'informazione diversa
      - Cognome
      - Numero telefonico

Progettazione
Esempio

- Sottoproblema 2
  - Si tratta di un problema di ricerca in un insieme di cognomi non ordinati
  - La ricerca può non avere successo
  - La ricerca è di tipo sequenziale (come per il sottoproblema 1)
    - Si parte dal primo cognome presente sulla pagina e si passa al cognome immediatamente successivo

Progettazione
Esempio

- La scansione dei cognomi nella pagina ha termine quando:
    - si è trovato un cognome uguale a quello cercato
    - il numero riportato accanto è il risultato

  oppure

  quando:
    - si sono scanditi tutti i cognomi senza trovare quello cercato
    - il risultato è “non presente”

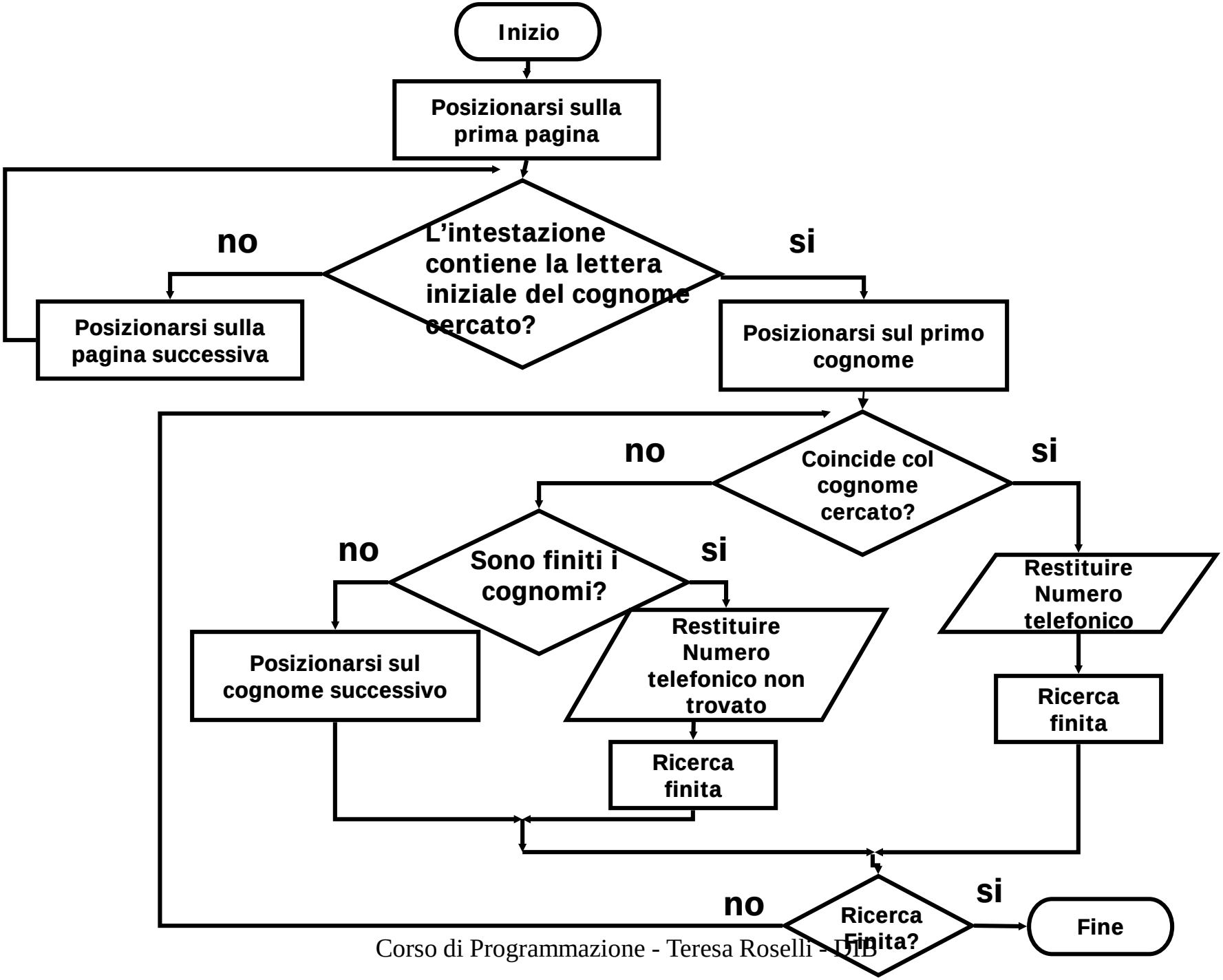
Progettazione
Esempio

- Sfogliare una pagina alla volta, iniziando dalla prima pagina, fino a che la pagina attuale contiene nell'intestazione la lettera iniziale del cognome cercato
- Scandire sequenzialmente un cognome alla volta, iniziando dal primo, fino a che il cognome attuale coincide con il cognome cercato
  - in tal caso comunicare il numero riportato accanto

  oppure non ci sono più cognomi
  - in tal caso comunicare "non presente"

Progettazione

- Elementi a disposizione
  - Dati in ingresso
  - Dati in uscita
  - Relazioni esistenti fra essi
- Prodotto
  - Rappresentazione delle entità del problema
  - Suddivisione in sottoproblemi
  - Strategia per passare dalle entità in ingresso a quelle in uscita


Inizio
Posizionarsi sulla prima pagina
L'intestazione contiene la lettera iniziale del cognome cercato?
no
si
Posizionarsi sulla pagina successiva
Posizionarsi sul primo cognome
Coincide col cognome cercato?
no
si
Sono finiti i cognomi?
no
si
Posizionarsi sul cognome successivo
Restituire Numero telefonico non trovato
Ricerca finita
Restituire Numero telefonico
Ricerca finita
Ricerca Finita?
no
si
Fine

S-composizione di Problemi
sviluppo BOTTOM-UP

Metodo opposto al Top-Down (dal basso verso l'alto)

Si parte dalle azioni primitive e costruendo da queste algoritmi semplici si collegano tra loro per ottenere algoritmi sempre più complessi sino ad arrivare all'algoritmo finale che costituisce la soluzione completa del problema.

Nella costruzione di un nuovo algoritmo ⟶ Top-Down

Nell'adattamento per scopi diversi di programmi già scritti ⟶ Bottom-Up

Sia la scomposizione che la composizione dà luogo a costruzioni

ben strutturate ⟶ necessità di linguaggi che consentano l'articolazione in sottoproblemi

Scomposizione di Problemi
Strumenti

- I costrutti offerti dai linguaggi di programmazione strutturata supportano le scomposizioni
  - Sequenziale
    - Suddividere un problema in parti disgiunte
  - Selettiva
    - Trattamento differenziato in casi differenti
  - Iterativa
  - Ricorsiva

Scomposizione Sequenziale

- La soluzione si ottiene tramite una sequenza di passi
  - I passi sono eseguiti uno alla volta
  - Ogni passo è eseguito una sola volta
    - Nessuno è ripetuto o omesso
  - L'ordine in cui i passi vanno eseguiti è lo stesso in cui sono scritti
  - L'ultimo passo equivale alla terminazione del procedimento

Scomposizione Sequenziale
Esempio

- Problema: Preparare una tazza di tè
- Soluzione: l'algoritmo per la tazza di tè
  1. Bollire l'acqua
  2. Mettere il tè nella tazza
  3. Versare l'acqua nella tazza
  4. Lasciare in infusione per 3 minuti
- Non sono problemi primitivi
  - Ciascuno va ulteriormente scomposto

Scomposizione Sequenziale
Esempio

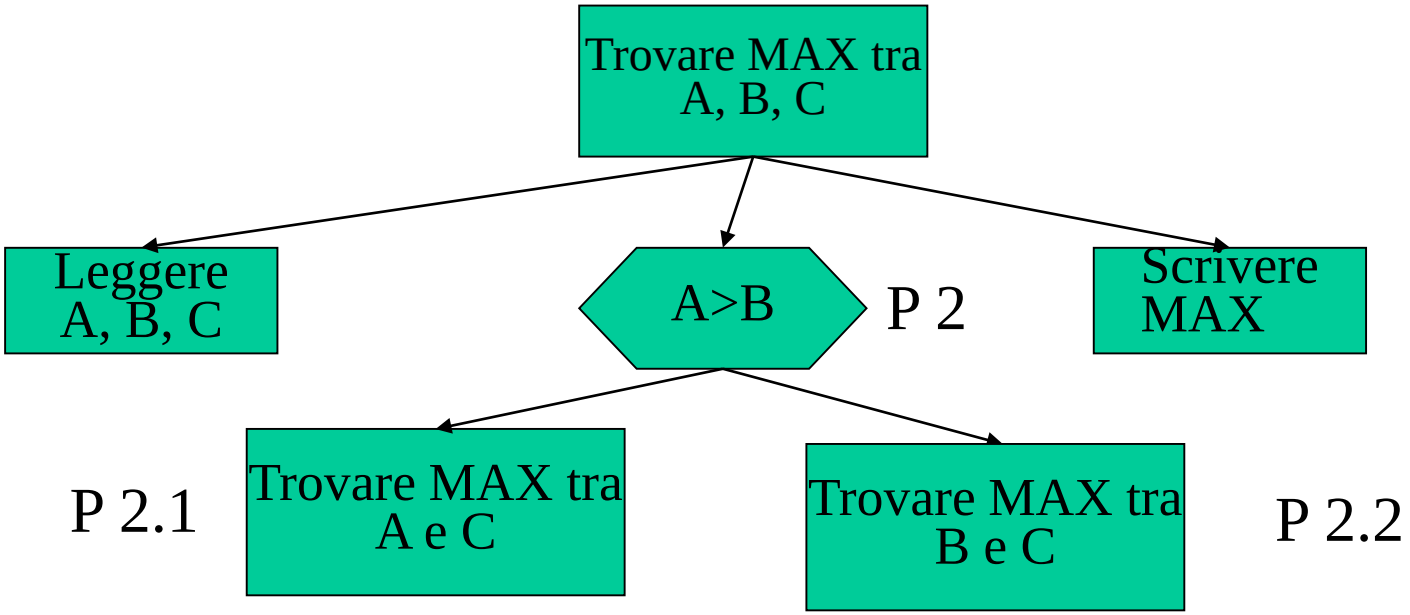
1.
   - 1.1 Riempire d’acqua il bollitore
   - 1.2 Accendere il gas
   - 1.3 Aspettare che l’acqua bolla
   - 1.4 Spegnere il gas
2.
   - 2.1 Aprire la scatola del tè
   - 2.2 Prendere un sacchetto-filtro
   - 2.3 Chiudere la scatola del tè
   - 2.4 Mettere il sacchetto nella tazza
3.
   - 3.1 Versare l’acqua bollente nella tazza finché non è piena
4.
   - 4.1 Aspettare 3 minuti
   - 4.2 Estrarre il sacchetto-filtro

Scomposizione Selettiva

- Soluzione ottenuta tramite scelte multiple
  - Strutture di selezione all'interno di altre strutture di selezione
- La scomposizione fa ricorso a strutture multiple
  - Nidificazione (o *annidamento*) di strutture

Scomposizione Selettiva
Esempio

- Problema: Trovare il più grande dei numeri *a*, *b*, *c* diversi tra loro
  - Azione primitiva (nota):
    - Verificare se un numero è maggiore di un altro



Scomposizione Selettiva
Esempio

- **se** $a > b$ *P2*
  - **allora**: verifica **se** $a > c$ *P2.1*
    - **allora**: a è la soluzione P2.1.1
    - **altrimenti**: c è la soluzione P2.1.2
  - **altrimenti**: verifica **se** $b > c$ *P2.2*
    - **allora**: b è la soluzione P2.2.1
    - **altrimenti**: c è la soluzione P2.2.2

Scomposizione Selettiva
Esempio

Struttura CASE

Esempio di applicazione

In un ospedale lo schedario di cartelle cliniche contiene per ogni cartella

- codice numerico del paziente a cui si riferisce;
- dati anagrafici;
- Informazioni mediche

Si suppone che lo schedario sia ordinato secondo il codice numerico crescente.

Scomposizione Selettiva
Esempio

Le operazioni possibili sullo schedario sono:

- eliminazione della cartella di un paziente dimesso
- aggiornamento di una cartella con nuove informazioni
- inserimento di nuove cartelle per nuovi pazienti

Gli aggiornamenti vengono effettuati da un addetto che riceve un insieme di informazioni così organizzate:

- Si tratta di una successione di comandi
- Il comando può essere
  - Inserimento I sulla scheda seguito dal codice dopo il quale inserire
  - Eliminazione E seguito dal codice della scheda da eliminare
  - Aggiornamento A seguito dal codice della scheda da aggiornare

Si usa l'asterisco * per distinguere un comando dall'altro

Scomposizione Selettiva
Esempio

Esempio:

```
* I  117
       informazioni su paziente 118
* I  118
       informazioni su paziente 119
* E  120
* I   122
       informazioni su paziente 123
* A  125
       informazioni aggiuntive per
       paziente 125
   ..............
```

Scomposizione Selettiva
Esempio

Uso della struttura CASE per l'esecuzione di un singolo comando (scelta multipla)

```
If  c'è "*"  then
            case    lettera
                    I: effettua inserimento
                    E: effettua eliminazione
                    A: effettua aggiornamento
                    else
                    segnala comando errato
            endcase
        else
            segnala assenza scheda comando
endif
```

Scomposizione Selettiva
Verso l'Iterazione

- Altro modo per scomporre il problema:

  Trovare il più grande dei numeri *a*, *b*, *c* diversi tra loro
  - Trova il più grande fra *a* e *b*
  - Trova il più grande tra la soluzione del passo precedente e *c*

- Generalizzazione del problema precedente
  - Necessità di definire una *struttura di dati*

Scomposizione Iterativa

- Successione di problemi tutti dello stesso tipo
  - Ripetere un numero finito di volte un passo di soluzione in modo da avere, alla fine, la soluzione completa

Scomposizione Iterativa

- Quando durante la scomposizione di un problema la soluzione porta all'individuazione di una catena di sottoproblemi che soddisfano le seguenti condizioni:
  - I sottoproblemi sono uguali, oppure differiscono solo per i dati su cui agiscono
  - I dati su cui agiscono i sottoproblemi sono in relazione d'ordine e un sottoproblema differisce dal precedente solo perchè utilizza il dato successivo.

allora si dice che il problema è risolto ITERATIVAMENTE

Scomposizione Iterativa
Esempio

- Trovare il più grande fra $n > 3$ numeri tutti diversi fra loro
  - Supponiamo che i dati siano in relazione d'ordine
    - Si può parlare di 1° dato, 2° dato, … n° dato

Scomposizione Iterativa
Esempio

- Trova il più grande tra i primi 2 numeri
- Trova il più grande fra il risultato precedente e il 3° numero
- …
- Trova il più grande fra il risultato precedente e l'ultimo numero (*n*° dato)

- Più concisamente:
  - Trova il più grande fra i primi 2 numeri
  - **Mentre** ci sono numeri da esaminare **esegui**
    - Esamina il primo numero non ancora considerato
    - Trova il più grande tra questo e il più grande precedentemente trovato

Esempio

Si dispone del livello delle precipitazioni relativo a tutti i giorni di un periodo di 4 settimane . Si vuole determinare il livello totale di precipitazioni per settimana e per tutto il periodo e il giorno di massima precipitazione per settimana.

Esempio

Dati di Input:

- Numero delle settimane
- Per ogni settimana:
  - Data
  - Precipitazione espressa in mm

Output:

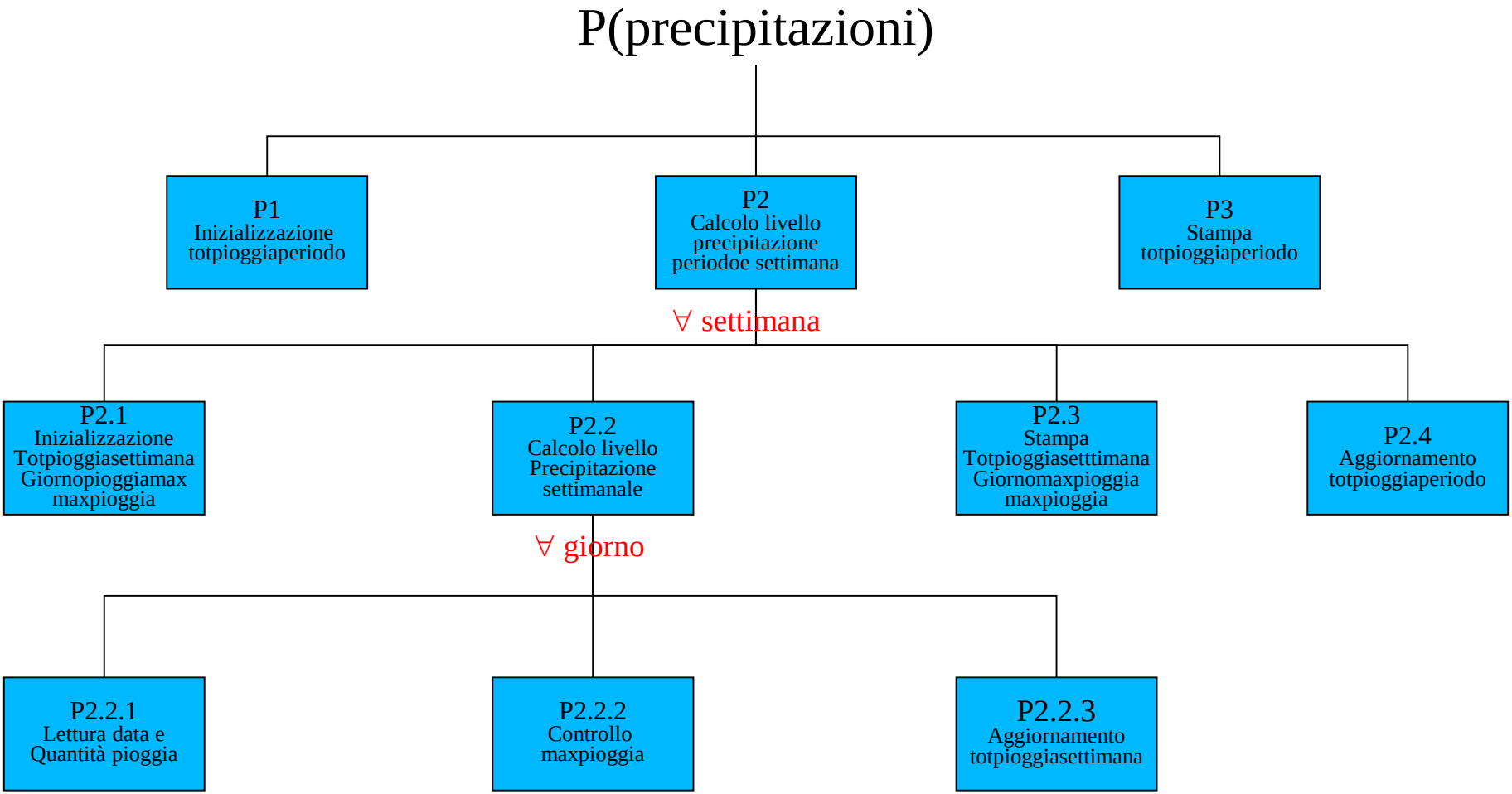
- Livello totale di precipitazione
- Per ogni settimana:
  - Livello totale di precipitazione
  - Data di massima precipitazione

Esempio

Metodo di soluzione

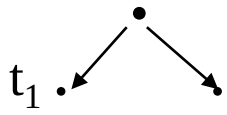
- Per calcolare il livello totale di precipitazione di n settimane occorre sommare i livelli di precipitazione di ogni settimana.
  - Per calcolare il livello di precipitazione di ogni settimana occorre sommare i livelli di precipitazione di ogni giorno della settimana.
    - Per determinare il giorno di massima precipitazione di ogni settimana occorre confrontare i livelli di precipitazione di tutti i giorni della settimana

Esempio


P(precipitazioni)
P1
Inizializzazione
totpioggiaperiodo
P2
Calcolo livello
precipitazione
periodoe settimana
P3
Stampa
totpioggiaperiodo
∀ settimana
P2.1
Inizializzazione
Totpioggiasettimana
Giornopioggiamax
maxpioggia
P2.2
Calcolo livello
Precipitazione
settimanale
P2.3
Stampa
Totpioggiasetttimana
Giornomaxpioggia
maxpioggia
P2.4
Aggiornamento
totpioggiaperiodo
∀ giorno
P2.2.1
Lettura data e
Quantità pioggia
P2.2.2
Controllo
maxpioggia
P2.2.3
Aggiornamento
totpioggiasettimana

Oggetto Ricorsivo

- Definito in termini di se stesso
  - Esempi
    - Numeri naturali:
      - 1 è un numero naturale
      - Il successore di un numero naturale è un numero naturale
    - Strutture ad albero:
      - • è un albero (*albero vuoto*)
      - Se t_1 e t_2 sono alberi, allora anche la struttura



è un albero

Definizione Ricorsiva

- Ottenuta in termini di versioni più semplici della stessa cosa che definisce
- 2 livelli:
  - Passo
    - Operazione che riconduce la definizione ad un dato livello a quella di livello inferiore
  - Base (o *livello assiomatico*)
    - Definizione di partenza
      - Necessario per ricostruire i livelli successivi
      - Consente di definire problemi di ordine superiore

Scomposizione Ricorsiva

- Un problema di ordine *n* è definito in termini del medesimo problema di ordine inferiore
  - Entità definita in termini più semplici della stessa entità
  - Nella definizione deve apparire il livello assiomatico (o *di ricostruzione*)
    - Problema risolubile tramite un'azione primitiva

Scomposizione Ricorsiva
Requisiti

- Almeno uno dei sottoproblemi è formalmente uguale al problema di partenza, ma di ordine inferiore
- Al momento della scomposizione è noto l'ordine del problema
  - Scomposizione attuabile mediante *regola di copiatura*
    - Sostituire ad ogni occorrenza del problema la scomposizione ricorsiva

    fino ad avere problemi primitivi (raggiungere il livello assiomatico)

Scomposizione Ricorsiva
Esempio

- Trovare un metodo per *invertire* una sequenza di lettere
  - **se** la sequenza contiene una sola lettera **allora**
    - Scrivila (il problema è risolto)
  - **altrimenti**:
    - Rimuovi la prima lettera dalla sequenza
    - *Inverti* la sequenza rimanente
    - Appendi in coda la lettera rimossa

Scomposizione Ricorsiva
Esempio

- Invertire “roma”
  - “roma” (4 lettere): rimuovi “r”, inverti “oma”
    - “oma” (3 lettere): rimuovi “o”, inverti “ma”
      - “ma” (2 lettere): rimuovi “m”, inverti “a”
        - » “a” (1 lettera): è già invertita
      - Appendi “m”: “am”
    - Appendi “o”: “amo”
  - Appendi “r”: “amor”

Scomposizione Ricorsiva
Caratteristiche

- Un problema espresso ricorsivamente termina sempre
- Un problema esprimibile ricorsivamente si può risolvere iterativamente
- Una funzione esprimibile ricorsivamente è computabile
  - Esiste un algoritmo che la calcola e termina sempre
- Ogni funzione computabile per mezzo di un programma è ricorsiva

Iterazione e Ricorsione
Esempio

- Calcolo del prodotto di due interi *n* e *m*
  - Definizione iterativa:
    - $n \times m = m + m + \ldots + m$ (*n* volte)
  - Definizione ricorsiva:
    - $$n \times m = \begin{cases} 0 & \text{se } n = 0 \\ (n - 1) \times m + m & \text{se } n > 0 \end{cases}$$

Iterazione e Ricorsione
Esempio

- Soluzione iterativa

inizialmente sia il risultato uguale a 0

ripeti per ogni intero da 1 ad *n*

somma *m* al risultato per ottenere un nuovo risultato

il prodotto è il risultato finale

- Soluzione ricorsiva

se *n* è uguale a 0

allora il prodotto è 0

altrimenti

calcola il prodotto di $n - 1$ per m

somma *m* al prodotto

Scomposizione di Problemi
Conclusioni

- Può esserci più di una scomposizione di un problema in sottoproblemi
- Cause di difficoltà nella scomposizione
  - Comprensione intuitiva della complessità di un problema
  - Scelta tra le possibili scomposizioni
  - Necessità di una formulazione “adeguata” per ciascun sottoproblema

Scomposizione di Problemi
Livelli di Complessità

- Corrispondenti ai metodi di scomposizione
  - Formula
  - Sequenza di formule
    - Risultati intermedi
  - Algoritmi condizionali
  - Algoritmi iterativi
  - Algoritmi ricorsivi

Corso di Programmazione

Linguaggi di Programmazione

I parte

Prof.ssa Teresa Roselli

teresa.roselli@uniba.it

Linguaggio

- Insieme di sequenze di *simboli* appartenenti ad un definito *lessico*, giustapposti in sequenza secondo una opportuna *grammatica* (o *sintassi*)
- Per descriverlo è necessario un *meta-linguaggio*
  - Linguaggio che parla di un linguaggio

Messaggio

- Sequenza di frasi espresse in un linguaggio
  - Analizzabile dal punto di vista
    - Sintattico
      - Si verifica la forma linguistica in cui è codificato (sintassi)
    - Semantico
      - Si individua il significato associato alla forma linguistica (semantica)

Comunicazione Diretta

- Requisiti per i due interlocutori:

  l'**estensore** del messaggio

  - al momento della sua formulazione

  e il **ricevitore**

  - al momento della ricezione

  devono dare al messaggio **eguale significato**

Comunicazione Indiretta

A volte la comunicazione diretta non è possibile

- Cause:
  - Il ricevitore non conosce il linguaggio usato per la stesura del messaggio
  - Estensore e ricevitore hanno un diverso grado di conoscenza del linguaggio
  - Tra i due mancano adeguate convenzioni per un'interpretazione unica del messaggio

Si deve ricorrere ad una comunicazione intermediata da un **traduttore**

Programma

- Messaggio di comunicazione fra l'uomo e la macchina
  - Insieme di frasi costruite secondo regole molto rigide
    - Eliminazione di ambiguità nell'interpretazione dei comandi da parte della macchina
  - Necessità di linguaggi molto precisi
    - Le istruzioni obbediscono a rigorose regole grammaticali

Sistema di Calcolo

- Gerarchia di componenti software e hardware
  - Software applicativo (es. pacchetti ad usi speciali)
    - Insieme di programmi da utilizzare per applicazioni particolari
    - Generalmente scritti in un linguaggio ad alto livello
    - Forniti direttamente da costruttori di computer o ditte specializzate nella produzione di programmi
  - Software di sistema (sistemi operativi, traduttori, ecc.)
    - Insieme dei programmi che forniscono servizi e svolgono funzioni vitali per il software applicativo
  - Hardware (CPU, memorie, dispositivi di I/O, ecc.)

Comunicazione Uomo-Macchina

- Processore nudo
  - La macchina comprende il linguaggio macchina
    - Costituito da un insieme di istruzioni elementari
    - Ogni istruzione è una stringa di cifre binarie che specifica un'operazione e la cella di memoria implicata nell'operazione
- Processore vestito
  - Macchina in grado di comprendere un linguaggio di livello superiore
    - Usato per facilitare il compito di programmare la soluzione di un problema

Linguaggio Naturale

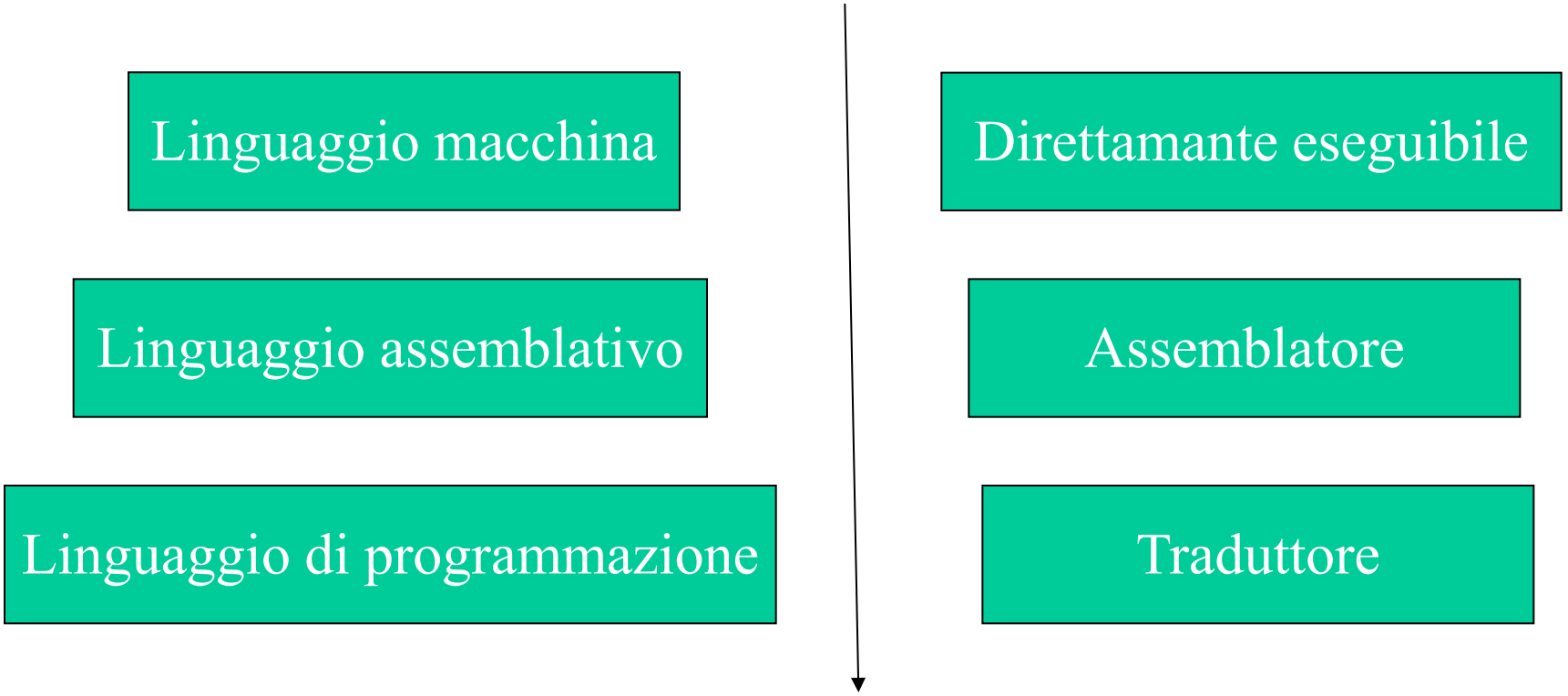
- Usato per la comunicazione verbale fra esseri umani
  - Fonti di ambiguità:
    - Evoluzione
      - Neologismi, Arcaismi
    - Polisemia
      - Parole con significati differenti a seconda del contesto
    - Intrinseca
      - …una vecchia porta la sbarra…
  - Inadatto alla comunicazione con la macchina

Linguaggi di Programmazione

- A basso livello
  - Più vicini alla struttura reale della macchina ed al suo linguaggio
- Ad alto livello
  - Più vicini al linguaggio dei problemi
  - Più facili da comprendere per l'uomo
  - Portabili
    - Utilizzabili, senza modifiche, su diversi tipi di macchine

Linguaggi

Basso livello



Alto livello

LINGUAGGIO NATURALE

Linguaggi di Programmazione
ad alto livello

- Procedurali
  - Descrivono i passi necessari per ottenere i risultati desiderati
    - "come"
  - Basati sui concetti di
    - Variabile
    - Assegnamento
- Non procedurali
  - Esprimono le proprietà dei risultati che si vogliono ottenere
    - "cosa"
  - Esempio
    Radice quadrata di *y*
    - Quel valore *x* tale che $x*x = y$

Linguaggi di Programmazione
Sintassi

- L'insieme delle regole che indicano quali sono le istruzioni formali permesse
  - Poche, semplici, rigide
- Il programma va accuratamente controllato dal punto di vista formale per garantire la correttezza sintattica
  - Codifica ambigua o non interpretabile
    - Controllo delegato al traduttore

Linguaggi di Programmazione
Semantica

- Riguarda il contenuto informativo ed il significato di una frase
  - verifica e controllo del programma dal punto di vista logico per garantire la correttezza a livello semantico
  - Informazione trasmessa non corrispondente allo scopo desiderato
  - Riguarda l'analisi del problema e l'algoritmo

Sintassi e Semantica
Esempio

- “Io ho andato”
  - Errata sintatticamente
- “La penna sta mangiando”
  - Corretta sintatticamente
    - Forma
  - Errata semanticamente
    - Significato

SINTASSI

Le regole sintattiche dei linguaggi di programmazione

- Non sono molto numerose
- Sono semplici
- Devono essere rispettate rigidamente

Il programma traduttore controlla la correttezza sintattica

Ad ogni istruzione deve corrispondere una unica interpretazione

Traduttore

- Programma che traduce in linguaggio macchina programmi in un linguaggio di livello superiore
  - Analizza i messaggi (comandi) e verifica che siano scritti (codificati) in un linguaggio a lui noto
    - Correttezza sintattica
  - Attribuisce alle sequenze di simboli l'opportuno significato in modo da eseguire le giuste azioni
    - Interpretazione unica di ogni istruzione
  - Fa parte del software di sistema
    - Livello intermedio della gerarchia software-hardware

Traduttori

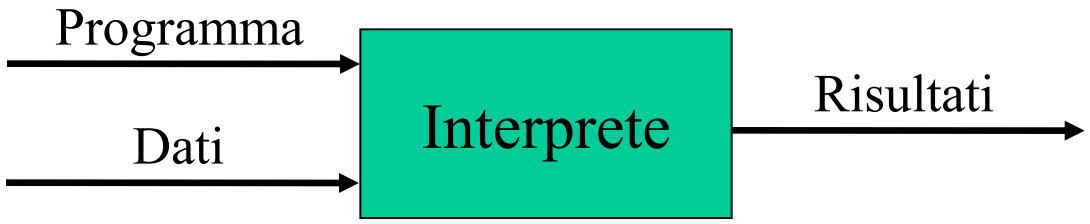
- Nei programmi ad alto livello operano su due tipi di entità:
  - Istruzioni
    - Molto più potenti che nel linguaggio macchina
  - Strutture di dati (liste, sequenze, alberi, ecc.)
    - Non direttamente disponibili al livello di linguaggio macchina
    - Devono essere rappresentate in termini di bit, indirizzi e legami tra locazioni

Traduttori

- Interpreti
- Compilatori
  - Specifici per ogni linguaggio
  - Forniti entrambi dai sistemi di sviluppo del software per i linguaggi supportati

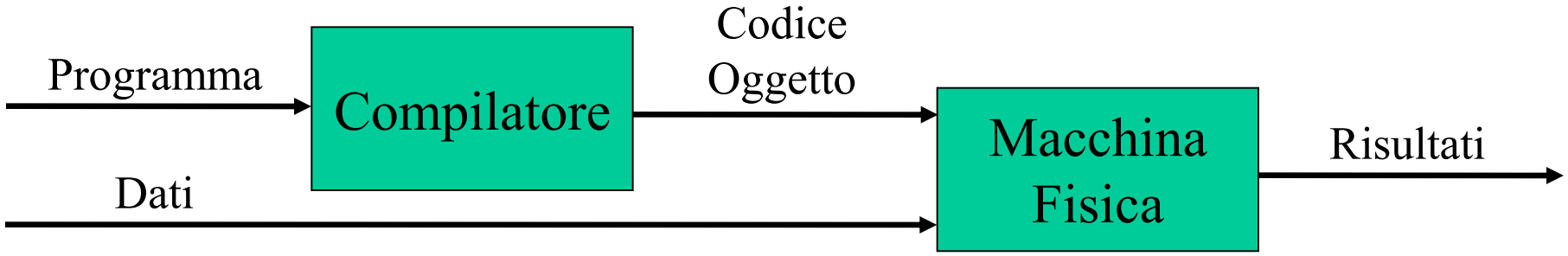
Interpretazione

- Dopo l'analisi sintattica, la traduzione procede passo passo con l'esecuzione
  - Traduzione ed esecuzione istruzione per istruzione
    - Ogni istruzione tradotta tante volte quante viene eseguita



Compilazione

- Il programma originale (*Sorgente*) è analizzato sintatticamente e tradotto in *codice oggetto*, quindi eseguito
  - Traduzione completamente effettuata prima che cominci l'esecuzione
    - Ogni istruzione è tradotta una sola volta



Interpreti vs. Compilatori

Programma sorgente residente in memoria

- \+ Semplici
- – Efficienti
  - Tempo e Spazio
- \+ Interattivi
- \+ Errori comprensibili
  - Riferiti al sorgente

Programma sorgente non residente in memoria

- \+ Ottimizzabili
- \+ Efficienti
  - Tempo e Spazio
- – Interattivi
- \+ Errori scoperti prima
  - Riferiti al codice oggetto

Processo di Compilazione
Fasi

- Analisi Lessicale
  - Divisione della stringa di caratteri del programma in *token*
    - Segni di interpunzione, nomi di dati, operatori, parole riservate
- Analisi Sintattica
  - Definizione della struttura sintattica del programma usando le regole grammaticali del linguaggio
- Generazione del Codice
  - Generazione di appropriate istruzioni in linguaggio macchina per ogni elemento sintattico del programma
    - L'insieme finale di queste istruzioni è il *programma oggetto*

Corso di Programmazione

Programmazione

I parte

Dati – Istruzioni

Prof.ssa Teresa Roselli

teresa.roselli@uniba.it

Comunicazione dell'algoritmo all'elaboratore

- Linguaggi non ambigui e sequenziali
  - Comprensibili alla macchina
- Requisiti per la descrizione
  - Univoca
    - Non dà adito ad interpretazioni errate
  - Completa
    - Prevede tutte le azioni necessarie
  - Ripetibile
    - Garantisce un buon risultato se eseguita da più esecutori con medesime caratteristiche

Programmazione

- Descrizione del procedimento di soluzione di un problema ad un esecutore *meccanico*
- Poiché l'esecutore è meccanico, consiste nel
  - Ricondurre il problema da risolvere a problemi primitivi
    - Eseguibili come insieme di azioni primitive
  - Organizzare ed utilizzare le "risorse" dell'elaboratore

Programmazione

- Trasformazione della descrizione di un algoritmo in un *messaggio*
  - Insieme di istruzioni codificate in un linguaggio interpretabile da un esecutore
- Passa attraverso un'astrazione
  - *Sia* delle **operazioni** che il procedimento prevede
  - *Sia* degli **oggetti** su cui il procedimento deve operare

Programmazione

- Metodologie
  - Astrazione
    - Oggetti
    - Azioni
- Tecniche
- Strumenti
  - Linguaggi
  - Ambienti

Programma

- Traduzione, in un linguaggio comprensibile alla macchina, della procedura di soluzione, con indicazioni sui dati di ingresso e di uscita
- Comunica al calcolatore istruzioni su
  - Quali dati di ingresso deve trattare
  - Come deve operare su questi dati
  - Quali dati deve dare come risultato

Programma

- Procedura eseguibile su calcolatore, che rappresenta una soluzione ad un problema
  - Risultato di un lavoro di analisi e progetto che inizia dalla formulazione del problema
  - Corrisponde alla tripla
    (Dati, Algoritmo, Risultati)
- Algoritmi + Strutture Dati = Programmi
  - [Wirth]

Programma

- Traduzione di un metodo di soluzione eseguibile in un linguaggio comprensibile alla macchina
  - Descrive come vanno elaborati insiemi di valori che rappresentano le entità del problema (2+3)
  - Usa rappresentazioni simboliche per estendere l'applicabilità del metodo di soluzione a valori diversi
    - Uso di ***variabili*** (x+y)

Dati

- Entità su cui lavora il problema
  - Costanti
  - Variabili
- Rappresentati come sequenze di bit
  - Nei linguaggi ad alto livello il programmatore può ignorare i dettagli della rappresentazione
    - Tipo di dato

Tipi di Istruzioni

- Un linguaggio di programmazione deve poter disporre di:
  - Istruzioni di ingresso
    - Permettono all'esecutore di conoscere informazioni fornite dall'esterno
  - Istruzioni di uscita
    - Permettono all'esecutore di notificare all'utente i risultati ottenuti dall'elaborazione
  - Istruzioni operative
    - Permettono di effettuare calcoli o, comunque, operazioni sulle entità astratte rappresentanti gli elementi del problema

Tipi di Istruzioni

- Istruzioni dichiarative
  - Consentono di definire come rappresentare le entità del problema in termini di variabili nel programma
    - Totale caratterizzazione attraverso la definizione di:
      - Un nome
      - Un tipo
- Strutture di controllo

Istruzioni Dichiarative

- Definiscono le aree di memoria in cui sono conservati i dati cui fa riferimento un algoritmo
  - Predispongono le posizioni di memoria da utilizzare
  - Associano un nome a ciascuna di esse
    - Identificatore
  - Determinano il tipo di dati che vi possono essere memorizzati
    - Insieme dei valori permessi
    - Insieme delle operazioni applicabili

Istruzioni di Ingresso/Uscita

- A livello di descrizione dell'algoritmo
  - Soddisfano la necessità di indicare i dati su cui operare
- A livello di programma
  - Soddisfano la necessità di comunicare i dati e i risultati
    - Istruzioni di lettura e scrittura

Istruzioni Dichiarative

- Forniscono una lista contenente i nomi scelti per le variabili e i tipi corrispondenti
  - Convenzione: indicare *tutte* le variabili
- Necessarie nei linguaggi di programmazione

Variabile

- Identificatore di variabile: nome simbolico per denotare un'area di memoria

- Individua un oggetto su cui l'algoritmo opera
- La memorizzazione di un valore nell'indirizzo di memoria associato ad una variabile avviene secondo uno dei seguenti modi
  - Istruzioni di ingresso
  - Istruzioni di assegnamento

Variabile

- Rappresenta una locazione di memoria del computer, contraddistinta da uno specifico *indirizzo*, che contiene il *valore* su cui applicare le istruzioni del programma
- Un identificatore di variabile denota una coppia
  - Posizione di memoria
  - Quantità in essa contenuta
    - Una limitazione nel rappresentare dati di tipo numerico o alfanumerico viene dalle dimensioni limitate della memoria

Variabile

- Caratterizzata da
  - Nome
    - *Identificatore*
  - Indirizzo
  - Valore
  - Tipo
    - Attributo che specifica l'insieme di valori che la variabile può assumere

Istruzioni di Ingresso

- Permettono di acquisire informazioni dall'esterno che vengono inserite nelle locazioni di memoria delle variabili dichiarate nel programma
- Attivano un'operazione di lettura
  - Assegnazione del valore letto da un supporto esterno
    - tastiera, dischi magnetici, dischi ottici …

    ad un'area di memoria individuata dal nome che compare nell'istruzione di lettura
- Esempio: read x

Assegnazione di valori a variabili

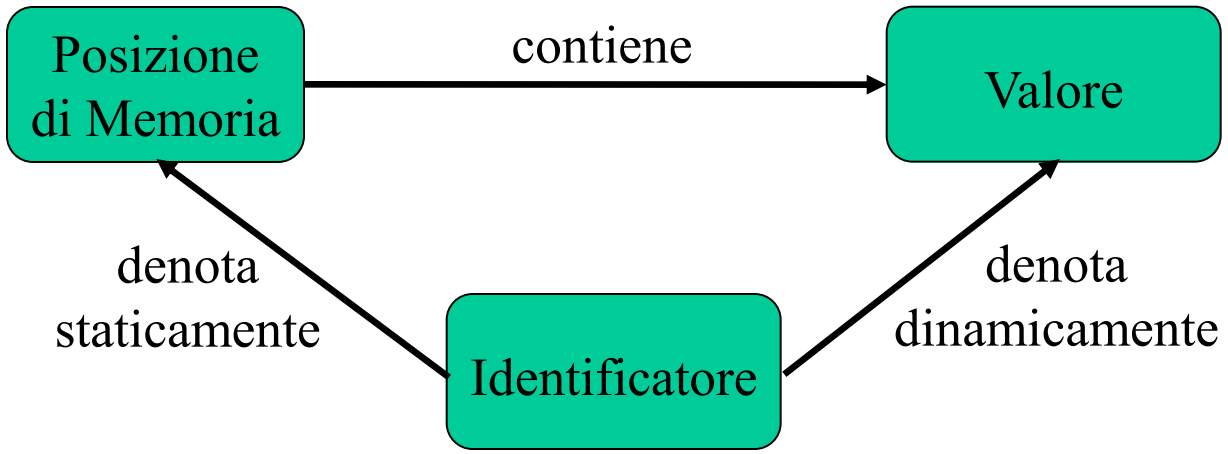
- Avviene in 2 fasi
  - Produzione di un nuovo valore
  - Assegnamento di quel valore alla variabile
- L'operazione di assegnazione viene indicata con ← := = a seconda dei diversi linguaggi di programmazione
  - Confusione fra uguaglianza e assegnazione

Assegnazione

- Per produrre il valore si possono usare espressioni (aritmetiche o logiche) il cui risultato è un singolo valore
- Il valore prodotto dall'espressione a destra viene memorizzato nella locazione di memoria riservata alla variabile a sinistra
  - La memorizzazione del valore ottenuto nella locazione di memoria riservata alla variabile implicata nell'assegnamento cancella qualunque valore contenuto in precedenza

Legami degli Identificatori

- Identificatore – Posizione di memoria
  - Legame statico
- Identificatore – Valore
  - Legame dinamico nel programma

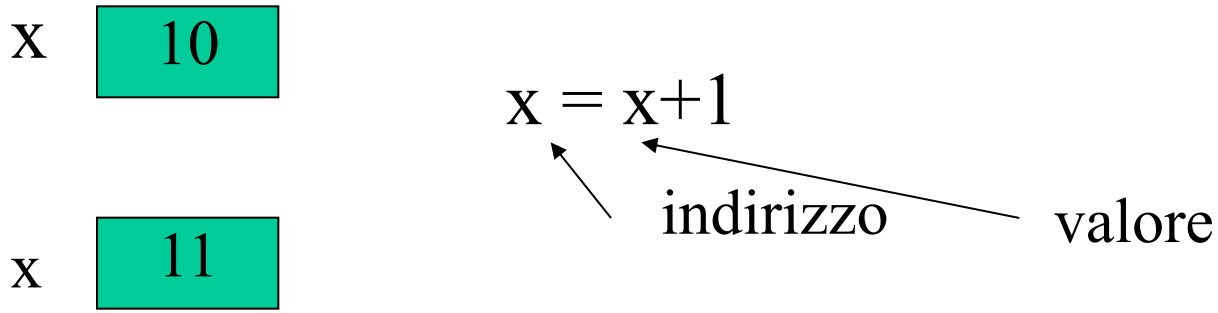


Legami degli Identificatori

- La dinamicità del legame identificatore-valore consente di scrivere senza contraddizioni assegnazioni del tipo

  $x \leftarrow x + 1$ (non è una uguaglianza)

  – Alla posizione di memoria identificata da *x* assegna il valore ottenuto calcolando la somma del valore già memorizzato e 1



Istruzioni di Uscita

- Consentono di notificare all'utente il valore di una variabile del programma
  - Visualizzazione, stampa, …
- Attivano un'operazione di scrittura
  - Copiatura su un supporto esterno
    - carta, display, dischi magnetici …

  del contenuto di un'area di memoria denotata dal nome della variabile che compare nell'istruzione di scrittura
- Esempio: write y

Assegnazione
Computo dei Valori

- Le operazioni di assegnazione possono implicare calcoli complessi
  - Espressioni aritmetiche
  - Espressioni logiche e predicati

Espressioni Aritmetiche

- Formate da associazioni di variabili e costanti secondo regole opportune e attraverso l'applicazione di definiti operatori numerici

Simbolo	Tipo di valore cui dà luogo	Operazione
+	Numerico	Somma
–, ÷	"	Differenza
*, •	"	Prodotto
/, ÷, DIV	"	Divisione
**, ↑, …	"	Elevamento a potenza

Espressioni Logiche o Predicati

- Usate in:
  - Assegnazioni che riguardano una variabile logica
  - Strutture di controllo che comportano la verifica di condizioni

Simbolo	**Tipo di valore cui dà luogo**	**Operazione**
NOT, AND, OR	Logico	Connettivi logici
=	“	Uguaglianza
≠	“	Diversità
<	“	Minoranza
>	“	Maggioranza
≤	“	Minore o uguale
≥	“	Maggiore o uguale

Costanti

- Dati il cui valore viene definito inizialmente e non varia per tutta l'esecuzione del programma
  - Accessibili solo in lettura
- Garanzia nell'uso
  - Impossibile eseguire assegnazioni sugli identificatori corrispondenti

Strutture di Controllo

- Obbligatorie
  - Sequenza
  - Selezione binaria
  - Una Iterazione illimitata
- Opzionali
  - Selezione multipla
  - L'altra iterazione illimitata
  - Iterazione limitata

Tipo di dato

- Un **dato** indica un valore che una variabile può assumere
- Un **tipo di dato** è un modello matematico che sta ad indicare una collezione di valori sui quali sono ammesse certe operazioni

Tipo di dato

- Attributo di una variabile, che ne specifica ed individua:
  - L'insieme dei valori che può assumere
  - L'insieme di operazioni effettuabili su di essa
  - Il modo con cui ci si può riferire ad essa

  - Esempio:
    Età: numero intero

Tipo di dato

- Si parla di tipo astrato di dato come di un oggetto matematico costituito dalla tripla:

  T = <D, C, O> dove

  – D rappresenta il dominio
  – C rappresenta le costanti
  – O rappresenta gli operatori
    - Funzioni
    - Predicati

Tipo di dato
Esempio

- Tipo dei complessi
  - Dominio
    - Sottoinsieme dei numeri complessi
  - Costanti
    - Parte reale
    - Parte immaginaria
  - Operazioni
    - Addizione +: complesso × complesso → complesso
  - Non ci interessiamo della rappresentazione interna dei valori e delle operazioni

Dichiarazione di Tipo

- Le istruzioni operative di un programma operano su rappresentazioni delle entità del programma.
- Occorrono delle istruzioni dichiarative che consentono di definire come rappresentare tali entità in termini di variabili mediante una totale caratterizzazione espressa in termini di
    - un nome
    - un tipo

Dichiarazione di Tipo
Utilità

- Definizione del dominio di applicazione del programma
- Comprensione del funzionamento di un algoritmo
- Verifica della correttezza del programma
  - Traduttore
  - Programmatore
- Definizione dello spazio di memoria necessario
- Rappresentazione interna
  - Esempio: 18 e 18.0 (intero o reale)

Definizione di Tipo
Meccanismi Linguistici

- Istruzioni e costrutti linguistici necessari per informare l'esecutore su
  - dominio della variabile
  - insieme di operazioni effettuabili su di essa
  - modo attraverso cui ci si può riferire ad essa
- A volte si ricorre all'indicazione esplicita dei valori permessi per ogni variabile
  - *Costanti di tipo*

Definizione di Tipo
nei Linguaggi di Programmazione

- I moderni linguaggi di programmazione mettono a disposizione
  - Un insieme di tipi di uso più comune
    - Tipi predefiniti
  - Gli strumenti per poter costruire qualunque tipo di dati

Tipi Standard

- Tipi più comuni di variabili
  - Interi
  - Reali
  - Logici
  - Caratteri
- Valori rappresentabili limitati
  - Dimensioni della memoria che dovrà ospitarne le variabili
    - Tipo numerico
    - Tipo alfanumerico

Tipo degli Interi

- Valori nell'insieme dei numeri interi
  - Stringa di cifre, eventualmente preceduta dal segno
    - Positivi
    - Negativi
- Operazioni basilari
  - somma, prodotto, differenza, quoziente, resto, elevamento a potenza

Tipo degli Interi

- I valori non sono in numero infinito nell'aritmetica dei calcolatori
  - Per ogni macchina esistono
    - il più grande intero
    - il più piccolo intero

    rappresentabile in una locazione di memoria
- Non valgono alcune proprietà dell'aritmetica (limiti fisici)
  - Risultato di un'operazione non rappresentabile nell'unità di memoria
    - *Overflow*
  - $x \oplus y = x + y$ solo se $|x + y| < max$ $\oplus$ indica l'addizione eseguita dal computer
  - Legge associativa $(x+y)+z=x+(y+z)$ solo se $|x + y| < max$ e $|y + z| < max$

Tipo degli Interi

- Sistema in base 10

  0 1 2 3 4 5 6 7 8 9

- Sistema in base 2

  0 1

- Sistema in base 16

  0 1 2 3 4 5 6 7 8 9 A B C D E F

  $n_b = d_p d_{p-1} \ldots d_1 d_0$

Come passare da base 10 ad un'altra base b?

Mediante divisioni successive per la base b fino ad arrivare a quella che produce quoziente nullo

$1972_{10} = 11110110100_2 = 7B4_{16}$

Tipo degli Interi
Rappresentazione dei Valori Negativi

- Rappresentare il segno nel primo bit
  - 2 rappresentazioni per lo zero
  - Differente gestione per somma e sottrazione

Tipo degli Interi

DEC	BIN	DEC	BIN
0	0000	-8	1000
1	0001	-7	1001
2	0010	-6	1010
3	0011	-5	1011
4	0100	-4	1100
5	0101	-3	1101
6	0110	-2	1110
7	0111	-1	1111

La regola meccanica per trovare il negativo di un binario è cambiare i bit 0 in 1 e i bit 1 in 0 e poi aggiungere 1

Tipo degli Interi
Rappresentazione in Complemento a 2

- Cambiare tutti gli 0 in 1 e viceversa
- Aggiungere 1
  - Esempio: $4_{10} = 0100_2$
    $0100 \rightarrow 1011 + 1 = 1100 \equiv -4$

- Valori rappresentabili con *n* bit: 2^n
  - Da -2^{n-1} a $+2^{n-1} - 1$
- I valori negativi iniziano sempre per 1
- Allineamento automatico per la somma

Tipo degli Interi

- Overflow nell'addizione
  - Esempi corretti

+3	0011
–6	1010
–3	1101

–8	1000
+7	0111
–1	1111

  - Esempio con overflow

+2	0010
+7	0111
–7	1001

$\neq +9$!!!

Tipo dei Reali

- Valori nell'insieme dei numeri reali
  - Parte intera
  - Parte decimale
    - Differenza tra il numero e la sua parte intera
    - < 1
    - Sequenza potenzialmente infinita di cifre
      - In alcuni casi esiste un'ultima cifra diversa da zero, seguita da una successione infinita di zeri

Tipo dei Reali

- Non formano un continuo nell'aritmetica dei calcolatori
  - Occorre discretizzare l'asse continuo dei valori reali
  - Ciascuno rappresenta un intervallo del continuo
    - Insieme di infiniti valori reali
  - Ad ogni numero reale è associata una rappresentazione
    - Intervallo in cui ricade
- Dominio = sottoinsieme finito e limitato dei numeri reali definito dall'implementazione

Tipo dei Reali

- Rappresentazione ottenuta per troncamento o arrotondamento
  - Possibili errori di precisione anche consistenti
- Esiste un valore massimo
  - Overflow
    - Rappresentazione indefinita per tutti i valori maggiori

Tipo dei Reali
Rappresentazione in Virgola Fissa

- r = n,m
  - *N* bit per la parte intera *n*
  - *M* bit per la parte decimale *m*
- Naturalmente allineati per addizione e sottrazione
- Cattiva gestione della memoria
  - Possibile spreco di memoria
  - Aumento delle probabilità di overflow
    - Parte intera
    - Parte decimale

Tipo dei Reali
Rappresentazione in Virgola Fissa

ESEMPIO

4 cifre per la parte intera
4 cifre per la parte decimale

$r_1 = 12{,}34$ | + | *0* | *0* | 1 | *2* | *3* | 4 | *0* | *0* |

$r_2 = 12456{,}34$ | + | *2* | *4* | *5* | *6* | *3* | 4 | *0* | *0* |

$r_3 = -0{,}000034$ | - | *0* | *0* | 0 | *0* | *0* | 0 | *0* | *0* |

Posizione della virgola ↑

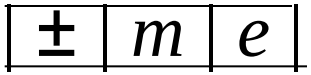
Tipo dei Reali
Rappresentazione in Virgola Mobile

- $r = \pm m \cdot b^e \rightarrow$ | ± | *m* | *e* | (*b* fissata)

  m mantissa
  - $13{,}18 * 10^0$
  - $1{,}318 * 10^1$
  - $0{,}1318 * 10^2$
  - $0{,}01318 * 10^3$

mantissa normalizzata: $1/b <= |m| < 1$

Tipo dei Reali
Rappresentazione in Virgola Mobile

- $z = \pm m \cdot b^e \quad \rightarrow \quad$  (b fissata)
  - 1 bit per il segno
  - N bit per il valore assoluto della mantissa m
    - Cifre più significative non nulle
    - Numero reale ($1/b \leq m < 1$)
  - M bit per l'esponente e
    - Ordine di grandezza della prima cifra significativa
    - Riferito alla base b (= 2)
    - Numero intero
      - Rappresentazione corrispondente

Tipo dei Reali
Rappresentazione in Virgola Mobile

- Ottimizzazione della gestione della memoria
  - Cifre significative
- Allineamento manuale per addizione e sottrazione
  - Riportare i valori allo stesso esponente
- Comoda per la moltiplicazione
  - Prodotto delle mantisse
  - Somma degli esponenti

Reali in Virgola Mobile
Esempio

$$\pi = 3{,}14159265\ldots$$
$$= +0{,}314159265\ldots \cdot 10^{1}$$

- Rappresentazione con 5 cifre significative
  - Troncamento
    3,1415 → | + | 31415 | +1 |
  - Arrotondamento
    3,1416 → | + | 31416 | +1 |

Reali in Virgola Mobile
Esempio

- Rappresentazione con 4 cifre significative

$211.5 = +0.2115 \cdot 10^3 \rightarrow |+|2115|+3|$

$37.592 = +0.37592 \cdot 10^2 \rightarrow |+|3759|+2|$

- Allineamento per la somma

$$\begin{array}{rl} & |+|2115\ \ |+3| \\ |+|3759|+2| \rightarrow & |+|03759|+3| \\ \hline & |+|2490\ \ |+3| \end{array} = +249.0$$

Reali in Virgola Mobile
Esempio

- Rappresentazione con 2 cifre significative
  - Con arrotondamento

$1{,}36 = +0.136 \cdot 10^1 \rightarrow |+|14|+1|$

$-1{,}34 = -0.134 \cdot 10^1 \rightarrow |-|13|+1|$

- Errore di approssimazione nella sottrazione

$$1{,}36 - 1{,}34 = 0{,}02$$

$$\begin{array}{l} |+|14|+1| \\ |-|13|+1| \\ \hline |+|01|+1| \end{array} = 0{,}1 \quad \textbf{!!!}$$

Tipo dei Reali
Proprietà

- Commutatività di somma e prodotto

  $x + y = y + x \qquad x \cdot y = y \cdot x$

  $x \geq y \geq 0 \Rightarrow (x - y) + y = x$

- Simmetria rispetto allo zero

  $x - y = x + (-y) = -(y - x)$

  $(-x) \cdot y = x \cdot (-y) = -(x \cdot y)$

  $(-x) \div y = x \div (-y) = -(x \div y)$

- Monotonia

  $0 \leq x \leq a \quad 0 \leq y \leq b$

  $x + y \leq a + b \qquad x - b \leq a - y$

  $x \cdot y \leq a \cdot b \qquad x \div b \leq a \div y$

Tipo dei Reali
Proprietà Mancanti

- Associativa
  - Può accadere che $(x + y) + z \neq x + (y + z)$
- Distributiva
  - Può accadere che $x \cdot (y + z) \neq (x \cdot y) + (x \cdot z)$

Tipo dei Valori Logici

- Rappresentano valori di verità
  - Falso, Vero
    - Falso < Vero
- Usati tipicamente nelle condizioni
  - Ottenibili come risultato di confronti

Tipo dei Valori Logici
Operatori

- Per priorità decrescente:
  - Not
  - And
  - Or

x	***y***	**not *x***	***x* and *y***	***x* or *y***
F	*F*	V	F	F
F	*V*	V	F	V
V	*F*	F	F	V
V	*V*	F	V	V

- Sottoinsieme completo
  - Può simulare tutti gli altri operatori logici

Tipo dei Caratteri

- Insieme finito ed ordinato di simboli
  - Lettere dell'alfabeto
  - Cifre decimali
  - Punteggiatura
  - Simboli speciali
    - Spaziatura (*blank*), Ritorno carrello, A capo, Separatore di linea (EOL), …
- Costanti di tipo Carattere racchiuse tra apici
  - Esempi: 'a' '8' '?' '@'
  - Per assegnare il valore ? alla variabile x si usa l'istruzione x='?'

Tipo dei Caratteri
Rappresentazione

- Corrispondenza biunivoca tra l'insieme dei caratteri e un sottoinsieme degli interi
  - Standard ASCII
    (American Standard Code for Information Interchange)
    - 7 bit $\rightarrow$ 128 simboli (2^7)
- Funzioni di trasferimento
  - ord(c)
    - numero d'ordine del simbolo *c* nella tavola di codifica
  - chr(i)
    - Simbolo il cui numero d'ordine è *i*

Tipo dei Caratteri
Proprietà

- ord(chr(i)) = i chr(ord(c)) = c
  - $c_1 < c_2 \leftrightarrow ord(c_1) < ord(c_2)$
- Relazione d'ordine totale
  - Coerente con i sottoinsiemi delle lettere e delle cifre
    - ord('A') < ord('B') < … < ord('Z')
    - ord('a') < ord('b') < … < ord('z')
    - ord('0') < ord('1') < … < ord('9')

Tipo dei Caratteri

Note

- Alcuni caratteri non sono stampabili
  - Esempio: CR (carriage return)
- I caratteri delle cifre sono diversi dalle cifre
  - Hanno come rappresentazione interna un valore diverso dalla cifra che rappresentano
- Relazione d'ordine totale
  - Proprietà riflessiva, antisimmetrica, transitiva

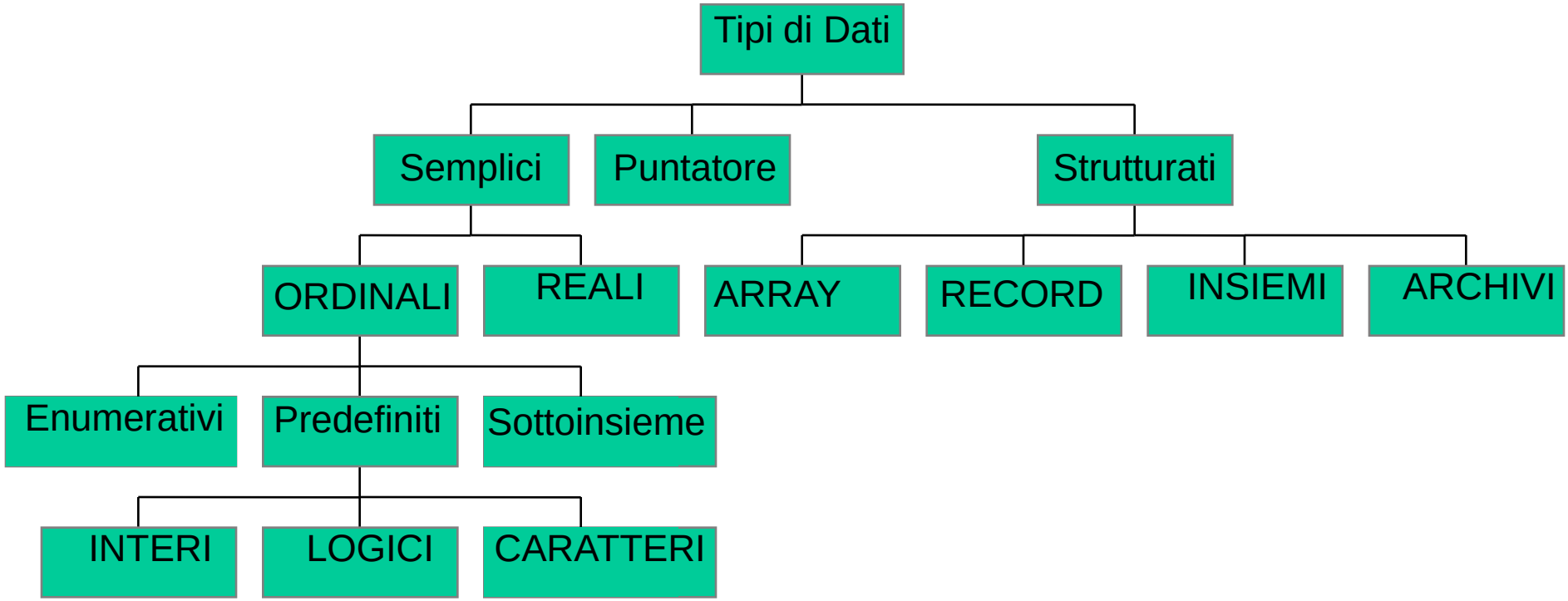
$\forall\ x: x \leq x$

$\forall x, y: \quad x \leq y \ \wedge \ y \leq x \longrightarrow x = y$

$\forall x, y, z: x \leq y \wedge y \leq z \longrightarrow x \leq z$

$\forall x, y: x \leq y \vee y \leq x$ (relazione d'ordine totale)

Tipi di Dati
Tassonomia



Definizione di Tipo

- Fornisce, tramite le istruzioni dichiarative:
  - L'indicazione di tutti i valori caratterizzanti il tipo
  - L'eventuale strutturazione di insiemi di valori
- Esempio: tipo t : T
  - *t* è il nome che indica il tipo
  - *T* ne è la descrizione

Tipi Semplici

- I cui elementi sono singoli valori

Tipi Ordinali

- Corrispondenza uno-a-uno fra i valori e un intervallo di interi
  - *Ordinali* dei relativi valori
- Operazioni consentite:
  - Confronto
  - Assegnamento
  - Funzioni predecessore-successore
    - pred(X)
    - succ(Y)

Tipi Ordinali

- Sono tipi ordinali quelli usati più di frequente
  - Solitamente *predefiniti* in ogni linguaggio
    - Interi, Logici, Caratteri
- Altri sono definibili dal programmatore
  - Enumerativi (il più spontaneo)
    - Elenco dei valori
  - Sottoinsieme (subrange) di un tipo scalare
    - Valori estremi

Tipi Ordinali
Enumerativi

- Definiti elencando l'insieme dei valori che ad essi competono
  - tipo T : (v_1, v_2, …, v_n)
- Gli insiemi di valori sono definiti ed ordinati.
- Per ogni tipo *T* deve valere:
  - I valori sono distinti ($\forall\ i \neq j: v_i \neq v_j$)
  - L'ordine dipende dall'elencazione ($i<j: v_i < v_j$)
  - Gli unici valori del tipo sono quelli elencati (solo v_i, i=1…n appartiene al tipo *T*)

Tipi Ordinali
Subrange

Quando un dato assume un intervallo di valori che è un *subrange* dei valori descritti da un certo tipo *ordinale* esistente il suo tipo può essere definito come un subrange di quel tipo ospite

- Basati su un tipo ordinale preesistente
  - Tipo *Ospite*
- Definiti specificando *il più piccolo valore* e *il più grande valore* dell'intervallo dei valori rappresentati dal subrange
  - tipo T : [v_{min} … v_{max}]
  - $v_{min} <= v_{max}$ e devono essere dello stesso tipo

Tipi Ordinali
Esempio

- Enumerativo
  - tipo giorno : (lun, mar, mer, gio, ven, sab, dom)
    - x: giorno (dichiarativa della variabile con identificatore x e di tipo giorno)
    - x ← gio (assegnazione del valore gio alla variabile x)
    - se x < sab allora …
    - pred(gio) = mer
    - succ(gio) = ven
- Subrange
  - tipo feriale : [lun … sab]

N.B. Non è permesso definire un subrange del tipo reale poichè esso non è un tipo ordinale

Corso di Programmazione

Tipi strutturati

Prof.ssa Teresa Roselli

teresa.roselli@uniba.it

Dati Strutturati

- Insiemi di valori correlati
  - Indicati collettivamente da un unico nome
  - Si presuppone che tra essi esista una struttura, legata
    - All'organizzazione
    - Al tipo di valori che compongono l'insieme
    - Alle operazioni per estrarre i dati dall'insieme
  - È fondamentale il modo in cui i valori componenti vengono individuati

Dato strutturato
Esempi

- Tabelle
  - Orario delle Lezioni
  - Orario Ferroviario
  - Matrici

Dato strutturato

Esempi

	1° ora	2° ora	3° ora	4° ora
Lunedì	Italiano	Ed. Fisica	…….	…….
Martedì	Italiano	Matematica	…….	…….
Mercoledì	…….	…….	…….	…….

Struttura di Dati

Una struttura di dati è un insieme di dati correlati che possono non essere tutti dello stesso tipo.

I dati sono legati da una organizzazione ed è fondamentale il modo in cui i dati componenti vengono individuati.

- Esempio:
  - Schede
    - Documenti di riconoscimento
    - Schede di biblioteca

Variabili Strutturate

I dati strutturati e le strutture di dati vengono memorizzati in varibili strutturate.

Una variabile di tipo strutturato possiede più di una componente

- Ia dichiarazione prevede l'indicazione di
    - Nome delle struttura
    - Tipo di struttura
    - Tipo delle componenti
- Strutture più comuni già previste dai linguaggi di programmazione
    - Operatori già definiti
        - Alcuni linguaggi consentono operazioni su intere variabili strutturate
- Strutture più complesse definibili attraverso il costrutto di tipo

Variabile strutturata

- Agglomerato (significativo) in cui sono riuniti dati da elaborare
  - Particolare tipo di dato
    - Caratterizzato *più* dall'organizzazione imposta agli elementi componenti che dal tipo degli elementi stessi
- Prevede
  - Un modo sistematico di organizzare i dati
  - Un insieme di operatori per
    - Manipolare elementi della struttura
    - Aggregare elementi per costruire altre strutture

Strutture di Dati

- I moderni linguaggi di programmazione mettono a disposizione
  - Un insieme di strutture di uso più comune (predefinite)
    - Sufficiente l'indicazione di
      - Dimensione
      - Tipo delle componenti
  - Gli strumenti per poter costruire qualunque tipo di struttura
    - Costruttori di tipo

Strutture di Dati
Classificazione

- Disposizione dei dati componenti
  - Lineari
    - Dati disposti in sequenza
      - Primo elemento, secondo elemento, …
  - Non lineari
    - Non è individuata una sequenza
- Numero di dati componenti
  - A dimensione fissa
    - Il numero di elementi della struttura rimane costante nel tempo
  - A dimensione variabile
    - Il numero di elementi può variare nel tempo

Meccanismi di Strutturazione

- Trasformazione Diretta
  - Vettore o array
- Prodotto Cartesiano
  - Record
- Insieme Potenza
  - Set
- Sequenze
  - File

Vettore
Array monodimensionale

- Tabella monodimensionale
  - Struttura lineare
  - A dimensione fissa
- Sequenza di elementi dello stesso tipo
  - Operazioni consentite:
    - Lettura (selezione)
      - Reperimento del valore di un elemento
    - Scrittura (sostituzione)
      - Sostituzione del valore di un elemento con un nuovo valore

Vettore
Array monodimensionale

- Numero fissato di componenti
  - Tutte dello stesso tipo (array = struttura omogenea)
    - Tipo *base*
  - Ciascuna esplicitamente denotata ed indirizzata tramite un selettore (*indice)*
    - Non si è legati ad uno specifico tipo di indice
  - Memorizzate in celle adiacenti di memoria
- Definito da:
  - Tipo degli elementi
  - Numero degli indici (un indice nel caso di array monodimensionale)
  - Tipo degli indici

Vettore
Array monodimensionale

array (*tipo_indice*) di *tipo_base*

- Accesso a qualunque componente
  - Specificandone la posizione
    - Nome della variabile array seguito dall'indice
  - In un tempo indipendente dal valore dell'indice

    (si ottiene mediante il calcolo di una funzione che si basa sull'indirizzo della prima posizione)
    - Accesso diretto (*random*)
- Un elemento di un array può essere a sua volta di un tipo strutturato

Array monodimensionale
Rappresentazione

- Componenti allocate in posizioni di memoria contigue
  - Ordinatamente
  - Consecutivamente
- Occupazione totale di memoria *d* * *n*
  - Tipo base *d* parole di memoria
    - Dipendente dal tipo di componenti
  - Vettore *n* elementi
- Nota la posizione della prima componente: I_0
  - la *j*-esima componente ha posizione $I_j = I_0 + (j - 1) * d$

Array

- Dipendono dal linguaggio di programmazione:
  - Modalità di
    - Dichiarazione
    - Definizione dell'intervallo di variabilità dell'indice
    - Scrittura degli indici
  - Possibilità di operazioni multiple
    - Agiscono su tutti gli elementi della struttura
      - Purché abbiano uguali dimensioni
    - Se non definite, necessarie iterazioni

Array

- Solo tipo strutturato disponibile in alcuni linguaggi
- Indici solo interi positivi in alcuni linguaggi di programmazione
- Nei linguaggi a tipizzazione forte esistono
  - Dichiarativa di *tipo* array
  - Dichiarativa di *variabile* array (diretta)
- In altri linguaggi la dichiarativa di array è diretta e con notazioni implicite

Array
Allocazione

- Nessuno spazio è allocato quando è dichiarato il tipo di array, il tipo di array descrive soltanto la struttura di un array, lo spazio di memoria è allocato quando è dichiarata una variabile di quel tipo

Array
Elaborazione

- Sequenziale
- Su tutte le componenti
  - Uso di strutture iterative
    - Numero di ripetizioni noto a priori
      - For
    - Controllo della condizione sull'indice della struttura, rispetto alla dimensione massima dell'array
      - While
      - Repeat

Array Multidimensionali

- Array di Array
  - Esempio:

    tipo materia : (italiano, matematica, religione)

    orario : array(lun … sab) di array (8 … 13) di materia

    - Abbreviazione:

      orario : array(lun … sab, 8 … 13) di materia
  - Uso di tipi intermedi

    tipo lezioni :array(8 … 13) di materia

    orario : array(lun … sab) di lezioni

Array Multidimensionali
Rappresentazione

- Linearizzazione
  - Componenti memorizzate in sequenza
    - Si inizia da quelle più interne ovvero si fissano gli indici più esterni e si fa variare quello più interno
- Esempio
  - A(1…*m*, 1…*n*) 2 dimensioni
  - Memorizzazione per righe
    - Ind(A(i, j)) = (i – 1) * n *d + (j – 1) * d
  - Memorizzazione per colonne
    - Ind(A(i, j)) = (j – 1) * m *d + (i – 1) * d

Array Multidimensionali
Rappresentazione

- Esempio
  - A(1…*l*, 1…*m*, 1…*n*) 3 dimensioni
  - Memorizzazione per righe
    - A(i, j, k)

      (i – 1) * m * n *d + (j – 1) * n * d + (k – 1) * d

Array
Esempio

- In matematica, matrici
  - A una dimensione (*vettori*) o a più dimensioni
    - Sono fondamentali le dimensioni
  - Variabili sottoscritte o con indici
    - Considerate come un tutto unico
      - Operazioni tra matrici (algebra matriciale)
    - Indici interi
      - Rappresentano la posizione che quella variabile occupa in una struttura di variabili
      - Servono ad indicare univocamente quella variabile

Record

- Registra in una *n*-pla di dati le principali caratteristiche di un’entità
  - Struttura non lineare
  - A dimensione fissa
- Insieme di dati non omogenei
  - Operazioni consentite:
    - Lettura (selezione)
      - Reperimento del valore di un elemento
    - Scrittura (sostituzione)
      - Sostituzione del valore di un elemento con un nuovo valore

Record

- Numero fissato di componenti
  - Tipi potenzialmente diversi
  - Ciascuna esplicitamente denotata ed indirizzata tramite un selettore (*campo)*
    - Paragonabile ad una variabile ordinaria
    - Denotato da un identificatore
- Definito dalla descrizione, per ogni singola componente, di:
  - Tipo
    - Limiti di variabilità del valore che può assumere
  - Identificatore per accedervi

Record

record *identificatore* : *tipo*;

…

identificatore : *tipo*

- Accesso a qualunque componente
  - Specificandone il campo
    - Nome della variabile record seguito dall'identificatore del campo
- Un elemento di un record può essere a sua volta di un tipo strutturato

Record
Rappresentazione

- Componenti allocate
  - In posizioni di memoria contigue
  - Nell'ordine in cui sono specificate nella dichiarazione
- Occupazione di memoria complessiva
  - Somma dell'occupazione di ciascun campo
    - Note in fase di compilazione
- Posizione di memoria di un campo
  - Somma della posizione iniziale del record e della somma delle dimensioni dei campi precedenti

Record

- Variabile strutturata a molte componenti
  - Aggregazioni di dati
  - Tipi potenzialmente differenti
  - Accesso alle componenti tramite nome
- Astrazione delle modalità di memorizzazione dei dati usate a livello di linguaggio macchina

Record
Esempi

- Data
  - Giorno, Mese, Anno
- Scheda bibliografica
  - Autore, Titolo, Prezzo, Anno, Prestito
- Indirizzo
  - Via, N. civico, CAP, Città, Provincia
- Scheda anagrafica
  - Nome, Cognome, Data di nascita, Stato civile

Array vs. Record

- Dimensione fissa
- Tipi omogenei
- Sequenza
  - Ordine
- Accesso diretto
  - Indice
    - Uso di espressioni
      - Flessibilità

- Dimensione fissa
- Tipi diversi
  - Più generale
- Insieme
- Accesso diretto
  - Identificatore di campo

Array & Record

- Spesso si ha a che fare con strutture formate da array di record
  - Sequenza di schede
    - Simili fra loro
    - Distinguibili in base ad un sottoinsieme dei campi che le formano
      - *Chiave*
  - Esempi
    - Schedario di dipendenti (Chiave: Codice Fiscale)
    - Orario Ferroviario (Chiave: Numero Treno)

Corso di Programmazione

Sottoprogrammi

Procedure e funzioni

Prof.ssa Teresa Roselli

roselli@di.uniba.it

Programmazione Modulare

- Tecnica basata sul metodo di scomposizione di un problema in sottoproblemi logicamente indipendenti tra loro
  - Ad ogni sottoproblema corrisponde un modulo
    - Codificati separatamente
    - Compilati separatamente (talvolta)
    - Integrati solo alla fine per formare il programma complessivo

Programmazione Modulare

- Un problema caratterizzato da
  - un algoritmo *A*
  - che opera sull'insieme dei dati di partenza *D*
  - per produrre l'insieme dei risultati *R*

  viene suddiviso in un insieme finito di *n* sottoproblemi a differenti livelli caratterizzati dalla tripla

$$(D_i, A_i, R_i)$$

  - Interazione e ordine di esecuzione degli algoritmi secondari per ottenere la soluzione del problema originario gestita da un algoritmo coordinatore

Programmazione Modulare

- Algoritmo coordinatore → programma principale o main
- Algoritmi secondari → sottoprogrammi
  - Diversi livelli
  - Si costruisce una gerarchia di macchine astratte, ciascuna delle quali
    - Realizza un particolare compito in modo completamente autonomo
      - Proprie definizioni di tipi, dichiarazioni di variabili e istruzioni
    - Fornisce la base per il livello superiore
    - Si appoggia su un livello di macchina inferiore (se esiste)

  Il programma è visto come un *nuovo operatore* disponibile sui dati

  L*'astrazione funzionale* è la tecnica che permette di ampliare il repertorio di operatori disponibili

Programmazione Modulare
Tecniche per individuare i sottoproblemi

- Basate sul metodo di soluzione di problemi consistente nello scomporre un problema in sottoproblemi più semplici
  - Sviluppo top-down
    - Approccio step-wise refinement
  - Sviluppo bottom-up
  - Sviluppo “a sandwich”

Raffinamento per passi successivi

- Basato su
  - Raffinamento di un passo della procedura di soluzione
    - Legato alle modalità di esecuzione conseguenti una certa suddivisione in sottoproblemi
      - Necessario concentrarsi sul “cosa” piuttosto che sul “come”
  - Raffinamento della descrizione dei dati
    - Definizione della struttura e tipo
    - Definizione delle modalità di comunicazione
      - Come renderli comuni a più sottoproblemi

Sviluppo Top-Down

- Costruzione del programma per livelli successivi
  - Corrispondenza con la scomposizione del problema cui è relativo
    - Strumento concettuale per la costruzione di algoritmi
  - Dettagliamento successivo delle parti in cui viene scomposto (sottoprogrammi) fino al codice finale
    - Strumento operativo per l'organizzazione e lo sviluppo di programmi complessi
- Metodo *trial and error*
  - Prova e riprova alla ricerca della scomposizione ottimale

Sviluppo Bottom-Up

- Partendo dalle istruzioni del linguaggio
  - Costruzione di programmi molto semplici
  - Collegamento successivo in programmi più complessi

  fino ad ottenere il programma finale
- Usato soprattutto nell'adattamento di algoritmi codificati già esistenti a nuove situazioni

Metodo a Sandwich

- Basato su una cooperazione fra le tecniche top-down e bottom-up
  - Necessità di raffinare via via la soluzione del problema principale
    - Scomposizione in algoritmi che ne risolvono delle sottoparti
  - Disponibilità di algoritmi di base per problemi semplici
    - Raggruppamento in algoritmi via via più complessi

Sottoprogramma

- Corrisponde all'algoritmo secondario che risolve un sottoproblema
- Insieme di istruzioni
  - Individuate da un nome
  - Che concorrono a risolvere un problema
    - Ben definito
      - Sensato
    - Non necessariamente fine a se stesso
      - è di supporto per la risoluzione di problemi più complessi
      - rappresenta una funzionalità a se stante, una unità concettuale con un significato più ampio (prescinde dal problema presente)
- Esempi:
  - Scambio, Ricerca del Minimo, Ordinamento, …

Sottoprogrammi
Utilità

- Un programma viene strutturato in sottoprogrammi:
  - Per rispettare la decomposizione ottenuta con il metodo di progettazione dell'algoritmo
  - Per strutturare in maniera chiara l'architettura del programma
  - Perché lo stesso gruppo di istruzioni deve essere ripetuto più volte in diversi punti del programma (blocchi ripetibili)

Sottoprogrammi
Utilità

- Risponde alla necessità di risolvere uno stesso problema
  - Più volte
    - All'interno dello stesso programma
    - In programmi diversi
  - Su dati eventualmente diversi
- Unicità dello sforzo creativo

Sottoprogrammi
Utilità

- Stile e qualità del software
  - Leggibilità
  - Manutenibilità
  - Trasportabilità
  - Modularità
  - Reuso

  I sottoprogrammi giocano un ruolo fondamentale nella tecnica della programmazione

Sottoprogrammi

- SOTTOPROGRAMMA è una astrazione funzionale che consente di individuare gruppi di istruzioni che possono essere *invocate* esplicitamente e la cui chiamata garantisce che il *flusso di controllo* ritorni al punto successivo all'invocazione

Astrazioni Funzionali

- Fornite dai linguaggi di programmazione ad alto livello
  - Consentono di creare unità di programma (macchine astratte)
    - Dando un nome ad un gruppo di istruzioni
    - Stabilendo le modalità di comunicazione tra l'unità di programma creata ed il resto del programma in cui essa si inserisce
  - Assumono nomi diversi a seconda del linguaggio di programmazione
    - Subroutine
    - Procedure
    - Sub program
    - …

Astrazioni Funzionali

- Paragonabili a nuove istruzioni che si aggiungono al linguaggio
  - Definite dall'utente
    - Specifiche per determinate applicazioni o esigenze
  - Più complesse delle istruzioni base del linguaggio
    - Analogia con il rapporto fra linguaggi ad alto livello e linguaggio macchina
  - Ciascuna risolve un ben preciso problema o compito
    - Analogia con un programma

Astrazioni Funzionali

- Struttura risultante di un programma:

  Intestazione di programma
    Definizione di tipi
    Dichiarazioni di variabili
    Dichiarazione di macchine astratte (sottoprogrammi)
  Corpo di istruzioni operative del programma principale

  – La dichiarazione di una macchina astratta rispecchia le regole di struttura di un programma

Sottoprogramma

- Indipendentemente dalle regole sintattiche del particolare linguaggio di programmazione
  - Individuabile con un nome
    - Identificatore
  - Prevede l'uso di un certo insieme di risorse
    - Variabili, costanti, …
  - Costituito da istruzioni
    - Semplici o, a loro volta, composte (altre macchine astratte)
  - Differisce da un programma nelle istruzioni di inizio
    - Specificano che (e come) altri pezzi di programma possono utilizzarlo

Chiamata di Sottoprogrammi

- Provoca l'esecuzione delle istruzioni del sottoprogramma
  - Modalità: deve essere comandata dal programma chiamante
    - Specifica del nome associato
  - Effetto: si comporta come se il sottoprogramma fosse copiato nel punto in cui è stato chiamato
    - Eliminazione di ridondanza

Chiamata di Sottoprogrammi

- All'atto dell'attivazione (su chiamata) dell'unità di programma
  - Viene sospesa l'esecuzione del programma (o unità) chiamante
  - Il controllo passa all'unità attivata
- All'atto del completamento della sua esecuzione
  - L'attivazione termina
  - Il controllo torna al programma chiamante

Sottoprogrammi
Nidificazione

- Le risorse di cui fa uso un sottoprogramma possono includere altri sottoprogrammi
  - Completa analogia con i programmi
- Si viene a creare una gerarchia di sottoprogrammi
  - Struttura risultante ad albero
    - Relazione padre-figlio riferita alla dichiarazione

Sottoprogrammi
Comunicazione

- Definizione
  - Titolo o intestazione
    - Identificatore
    - Specificazione delle risorse usate (talvolta)
  - Corpo
    - Sequenza di istruzioni denotata dal nome del sottoprogramma
- Comunicazione
  - Come si connettono i sottoprogrammi tra di loro?
  - Come si scambiano dati?
  - Come comunicano col programma principale?

Sottoprogrammi
Comunicazione

- Un sottoprogramma può comunicare
  - Con l'ambiente esterno
    - Istruzioni di lettura e/o scrittura
  - Con l'ambiente chiamante
    - Implicitamente
      - Tramite le variabili non locali (secondo le regole di visibilità del linguaggio)
    - Esplicitamente
      - Attraverso l'uso di parametri
        - » rappresentano le variabili che il sottoprogramma ha in input dal programma chiamante e che, opportunamente elaborate, vengono tramutate in output del sottoprogramma

Vista di un Sottoprogramma

- Rappresenta l'insieme delle risorse a cui il sottoprogramma ha accesso
  - Dati
  - Altri sottoprogrammi
- E' definita da
  - Nidificazione nella dichiarazione dei sottoprogrammi
    - *Vista Statica*
  - Sequenza di chiamata dei sottoprogrammi
    - *Vista Dinamica*
- Utile per limitare l'accesso alle risorse soltanto ai sottoprogrammi interessati

Vista di un Sottoprogramma
Sottoprogrammi

- Un sottoprogramma può richiamare soltanto i sottoprogrammi
  - che esso dichiara direttamente
  - che sono stati dichiarati dallo stesso sottoprogramma che lo dichiara
    - Incluso se stesso
      - Ricorsione
- Visibilità definita esclusivamente in base alla nidificazione
  - Figli e fratelli nella struttura ad albero

Vista di un Sottoprogramma
Variabili

- Ciascun sottoprogramma può usare esclusivamente
  - Le proprie variabili
  - Le variabili dichiarate dai sottoprogrammi attualmente in esecuzione
    - Visibilità dipendente
      - Dalla struttura della gerarchia di dichiarazione dei sottoprogrammi (statica)
      - Dall'ordine di chiamata dei sottoprogrammi precedenti (dinamico)

Vista di un Sottoprogramma
Shadowing (oscuramento)

- Sottoprogrammi diversi possono dichiarare risorse con lo stesso nome
  - Oggetti diversi, totalmente scorrelati
    - Possono essere di tipi differenti
- Sottoprogrammi attivi in un certo istante possono aver dichiarato risorse con lo stesso nome
  - Ciascun sottoprogramma attivo ha accesso solo al sinonimo "più vicino"
    - Visibilità dipendente esclusivamente dall'ordine di chiamata dei sottoprogrammi precedenti

Sottoprogrammi
Tipi di Variabili

- Variabili locali al sottoprogramma
  - Interne al sottoprogramma
  - Temporanee
    - Create quando il sottoprogramma entra in azione
    - Distrutte quando il sottoprogramma è stato eseguito
      - Liberazione del relativo spazio di memoria
- Variabili non locali al sottoprogramma
  - Definite nel resto del programma, al di fuori del sottoprogramma
    - Dette *globali* se definite nel programma principale

Sottoprogrammi
Tipi di Variabili

MAIN

Risorse globali

SOTTOPROGRAMMA P1

Risorse locali a P1 e non locali a P1.1

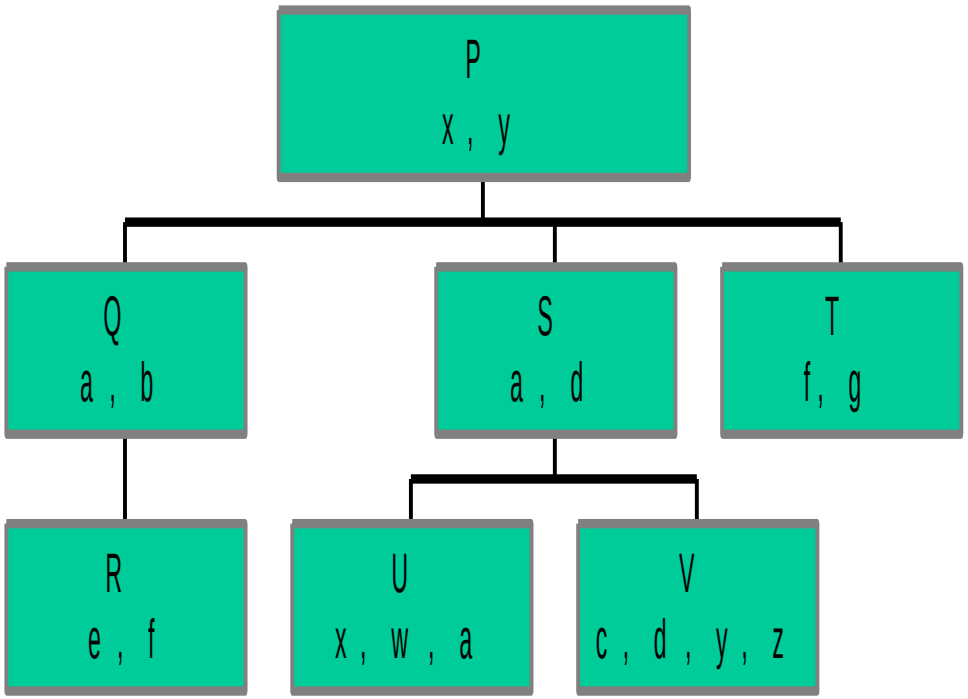
SOTTOPROGRAMMA P1.1

Risorse locali a P1.1

Sottoprogrammi
Regole di Visibilità

- Un identificatore è visibile nel programma o sottoprogramma in cui è dichiarato e in tutti i sottoprogrammi locali ad esso nei quali non è stato ridichiarato.
- Tutte le risorse di un programma o sottoprogramma devono essere dichiarate prima di essere usate
- Una *risorsa globale* è visibile ovvero accessibile ovvero usabile sempre e da tutti (main e sottoprogrammi) a meno che non venga oscurata (shadowing)
- Una *risorsa locale* ad un sottoprogramma P è visibile solo dalle istruzioni di P e dagli eventuali sottoprogrammi definiti in P

Vista di un Sottoprogramma
Esempio



Vista di un Sottoprogramma

- Delimitazione spaziale di una risorsa
  - Le regole di visibilità degli identificatori stabiliscono l'*ambito* o *campo di visibilità* o *scopo degli identificatori* ovvero la zona di programma in cui è possibile fare riferimento a quell'identificatore.
- Delimitazione temporale di una risorsa
  - Le regole di visibilità definiscono anche la durata o il tempo di vita di una variabile ovvero l'intervallo di tempo in cui una variabile esiste (è allocata una area della RAM per essa)

Attributi delle variabili

- VARIABILE
  - NOME
  - VALORE
  - INDIRIZZO
  - TIPO
  - AMBITO
  - DURATA

Corso di Programmazione

Sottoprogrammi

Procedure e funzioni

Prof.ssa Teresa Roselli

roselli@di.uniba.it

Durata e ambito di una variabile

- In alcuni linguaggi il tempo di vita di una variabile coincide con il *tempo di esecuzione* del programma o del sottoprogramma in cui la variabile è dichiarata
- La fase in cui è definita un'area di memoria per una variabile è detta *fase di allocazione*.
  - L'allocazione può essere eseguita:
    - dal compilatore (allocazione statica) in base alla struttura lessicale del programma
    - in esecuzione (allocazione dinamica)

Effetti Collaterali

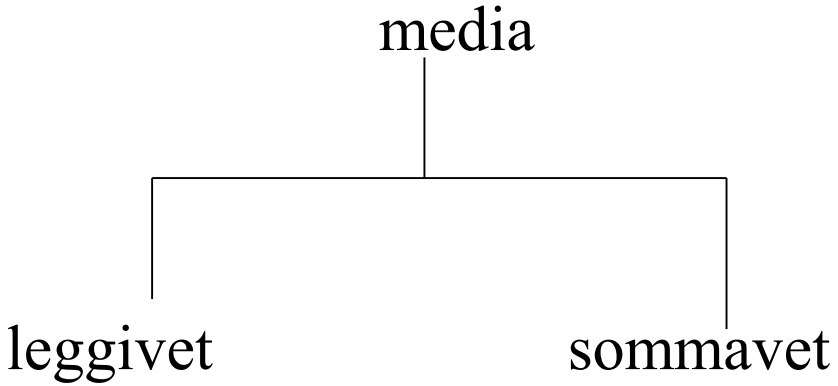
- Effetti di un sottoprogramma che altera il valore di una variabile non locale
  - La presenza di tali variabili impedisce che il sottoprogramma possa essere considerato come un'entità completa e autoconsistente
    - Non si riferisce esclusivamente alle sue costanti, variabili e parametri
- Attenzione: occorre valutare attentamente l'uso, all'interno di un sottoprogramma, di variabili non locali
    - Chiarezza
    - Sicurezza

Sottoprogramma
il countour model

```
PROGRAMMA media
    TIPO vettore=ARRAY(1,100) OF real
    vet:vettore
    n:integer
    m:real
    s:real
    SOTTOPROGRAMMA leggivet
            x:real
            i:integer
            BEGIN
                DO VARYING i FROM 1 TO n
                        leggi x
                        vet(i)=x
                REPEAT
            END
     SOTTOPROGRAMMA sommavet
            i:integer
            BEGIN
                s=0
                DO VARYING i FROM 1 TO n
                        s=s+vet(i)
                REPEAT
            END
    BEGIN
    leggi n
    leggivet
    sommavet
    m=s/n
    stampa m
    END
```

Sottoprogramma
il countour model

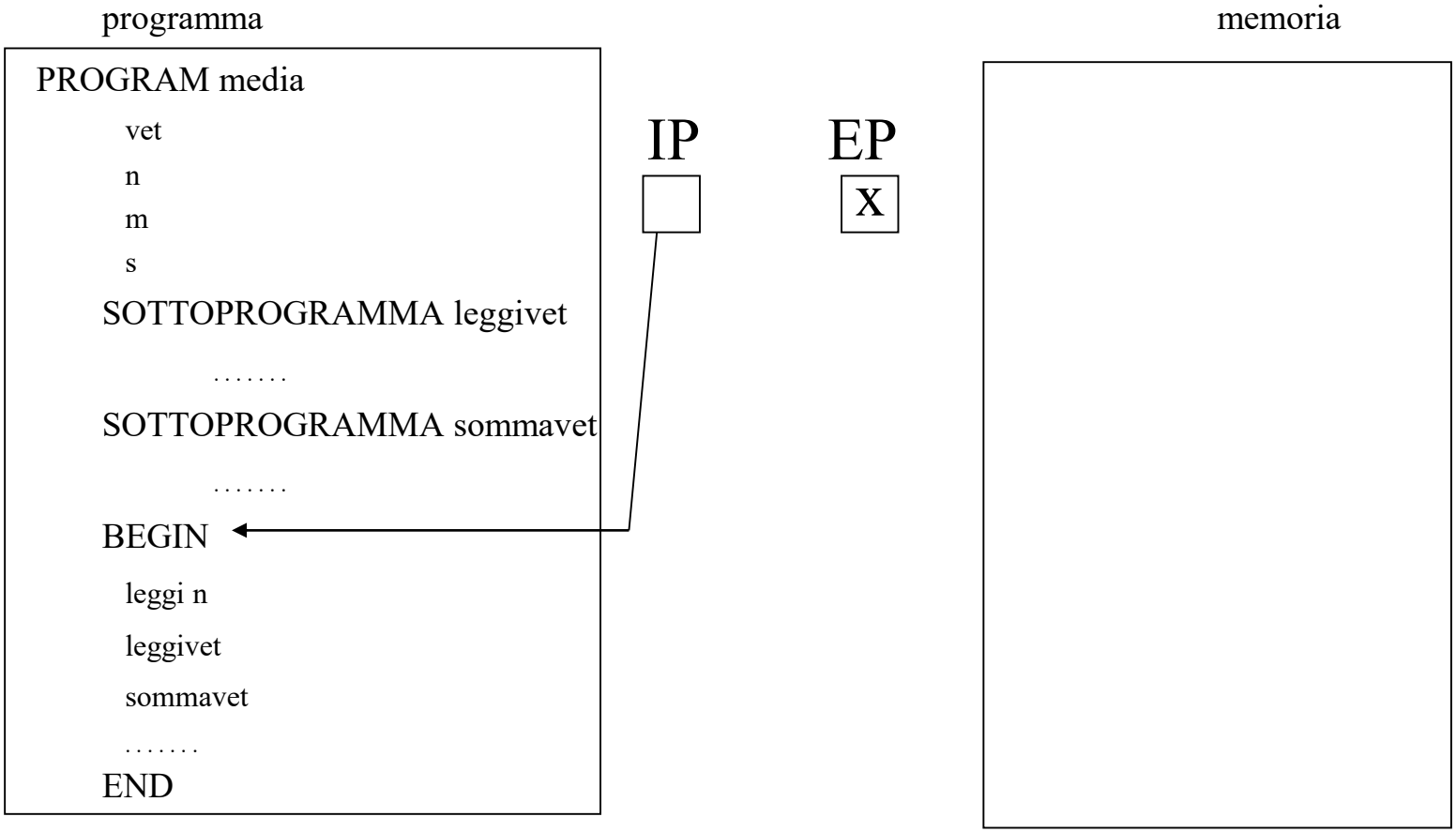
- La gerarchia di macchine è



  - media cede ai sottoprogrammi il diritto di accesso a tutte le sue variabili
  - ogni sottoprogramma ha le proprie variabili

Sottoprogramma
il countour model

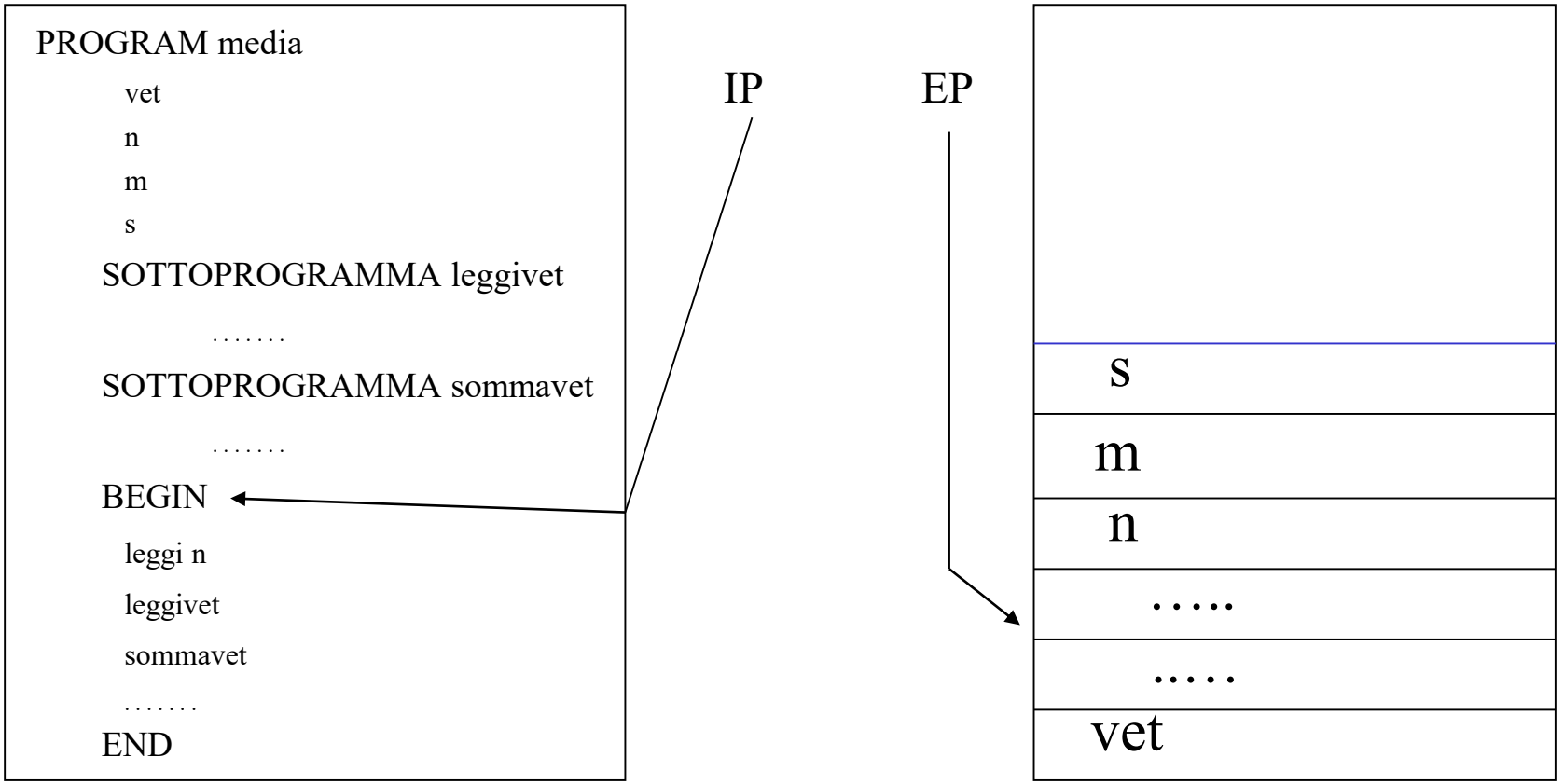
Il modello associato all'esecuzione del programma è detto *countour model*



IP instruction pointer (puntatore all'istruzione) EP environment pointer (zona della memoria per le variabili)

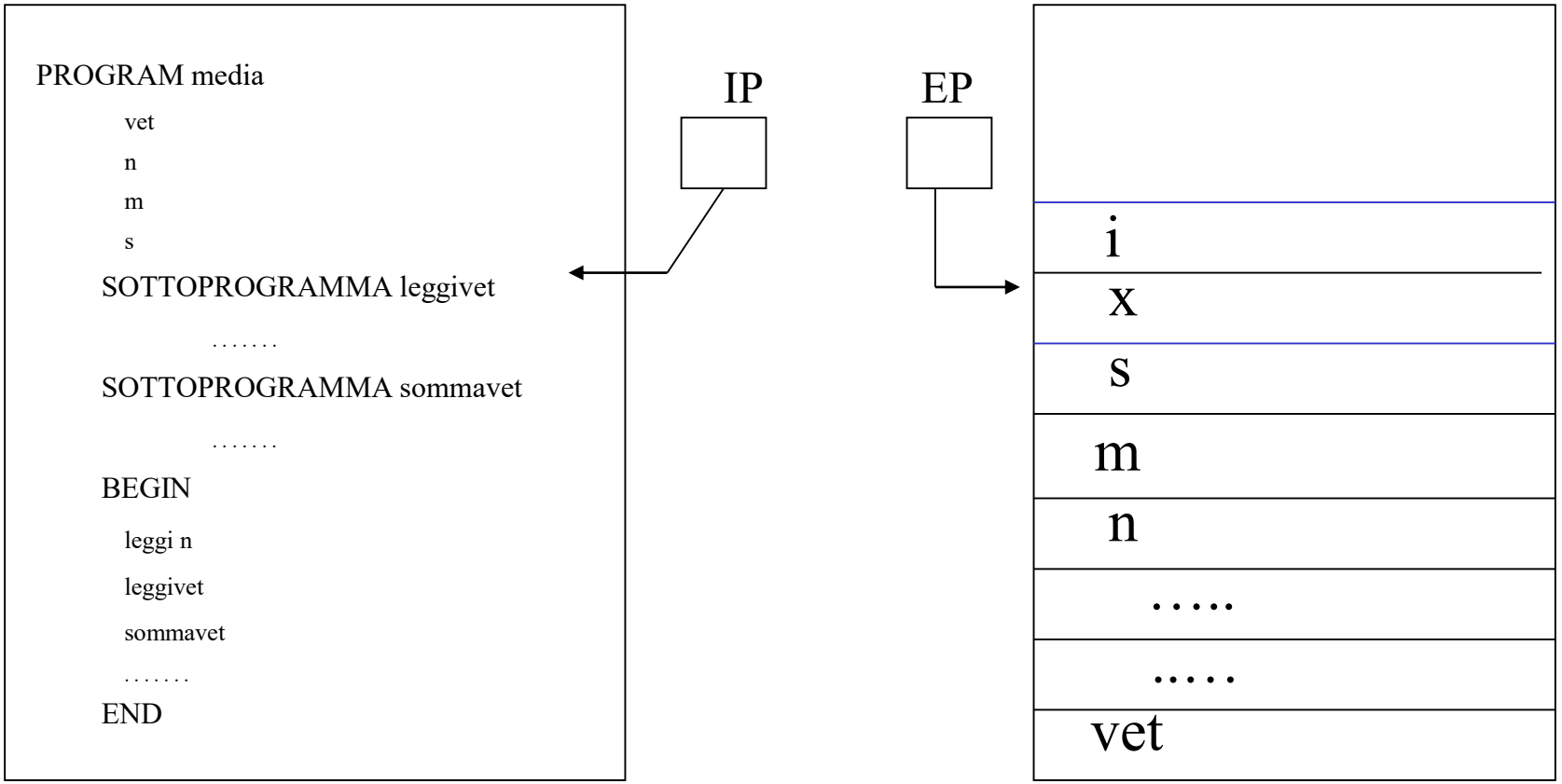
Sottoprogramma
il countour model

All'inizio dell'esecuzione vengono allocate le variabili del programma principale



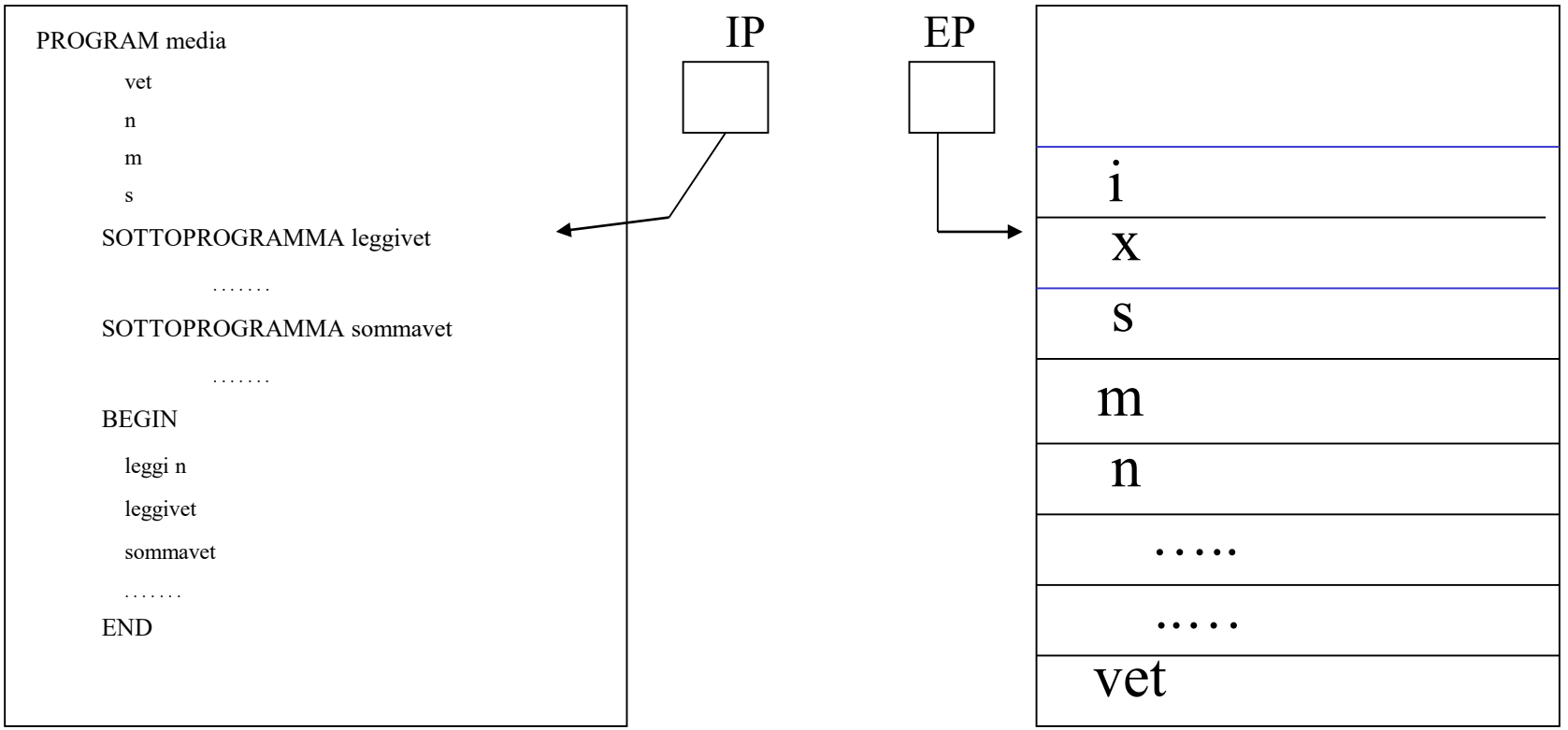
Sottoprogramma
il countour model

Quando viene richiamato il sottoprogramma leggivet vengono allocate nuove variabili in un nuovo ambiente a cui punterà EP



Sottoprogramma
il countour model

Leggivet userà le variabili i e x puntate da EP attuale mentre, non trovando n e vet nell'ambiente puntato da EP, risalirà negli ambienti precedenti sino a trovare la prima loro occorrenza. Questo è il meccanismo che consente ad un sottoprogramma di ereditare il diritto di accesso al programma che lo richiama



Sottoprogramma
il countour model

Terminata l'esecuzione di leggivet, l'ambiente puntato da EP non serve più quindi viene rilasciata l'area di memoria occupata e EP torna a puntare all'ambiente che aveva richiamato leggivet.

Sottoprogramma esempio

```
sottoprogramma quadrato
    i: integer
    quad: integer
    begin
        do varying i from 1 to 10
            quad ← i*i
            stampa quad
        repeat
    end
```

dichiarazione variabili

Per stampare i primi 15 quadrati?
Per stampare i primi 35 quadrati?

Sottoprogramma esempio

```
sottoprogramma quadrato (n: integer)
    i: integer
    quad: integer
    begin
        do varying i from 1 to n
            quad      i*i
            stampa quad
        repeat
    end
```

dichiarazione variabili

```
quadrato(10)
quadrato(m)
```

chiamate

Sottoprogramma

Nella dichiarazione del sottoprogramma:

- l'*intestazione* introduce il *nome* e gli *argomenti* del sottoprogramma
- Il *corpo* descrive le regole di comportamento del sottoprogramma

Nell'intestazione del sottoprogramma appaiono i *parametri formali* che all'atto della chiamata vengono sostituiti dai *parametri effettivi* (*attuali*) ovvero dai dati su cui il sottoprogramma deve operare

Parametri Formali

- Segnaposto per indicare simbolicamente
  - Gli oggetti su cui il sottoprogramma lavora
  - Il loro tipo e struttura
- I loro nomi appaiono nell'intestazione del sottoprogramma
  - Non hanno alcuna connessione con nomi usati altrove
- All'atto della chiamata vengono sostituiti dai parametri *effettivi* (o *reali*)
  - Dati su cui effettivamente il sottoprogramma deve operare

Parametri Formali

- Specificati all'atto della definizione del sottoprogramma
  - Legati al sottoprogramma
  - Simbolici
  - Consentono di definire
    - Quale tipo di dato deve essere passato alla procedura
    - Quale argomento deve essere trasmesso alla funzione

  quando queste sono invocate

Parametri Effettivi

- Alla chiamata di un sottoprogramma, vanno specificati i dati effettivi su cui esso dovrà operare
  - Valori, Espressioni, Variabili, …
  - Coincidenza con i parametri formali
    - Numero, tipo e ordine
- L'esecuzione dell'istruzione di chiamata comporta la sostituzione dei parametri formali con quelli reali

Parametri
Tipi di Passaggio

- La sostituzione può essere:
  - Per valore
    - Si calcola il valore del parametro reale e lo si sostituisce al corrispondente parametro formale (assegnazione)
  - Per referenza
    - Il parametro effettivo è una variabile ed ha a disposizione una locazione di memoria il cui indirizzo viene "passato" al parametro formale
  - Per nome (o *valore-risultato*)
    - Il nome del parametro formale, all'occorrenza, viene sostituito col nome del parametro reale

Parametri
Tipi di Passaggio

- *P*(*pf*) attivata con parametro reale *pe*
  - Per valore:
    - Il parametro formale si comporta come una variabile locale a *P*
  - Per riferimento:
    - Al momento dell'attivazione di *P* viene calcolato l'indirizzo di *pe* e *pf* viene creato con riferimento alla stessa locazione di memoria
  - Per nome:
    - Al momento dell'attivazione di *P* il valore di *pe* viene calcolato e memorizzato in una nuova locazione di indirizzo *pf*
    - Al termine dell'esecuzione di *P* il contenuto di *pf* viene trasferito in *pe* e la memoria riservata per *pf* viene rilasciata

Passaggio di Parametri
Esempio

```
program legaparametri (…,…);
    var n: integer;
    sottoprogramma P (? x : integer)
      begin
      x := x + 1;
      writeln(n);    {1}
      writeln(x)     {2}
      end;
    begin
    n := 3;
    P(n);
    writeln(n)       {3}
    end.
```

- Per valore
  1. 3
  2. 4
  3. 3
- Per referenza
  1. 4
  2. 4
  3. 4
- Per nome
  1. 3
  2. 4
  3. 4

Passaggio di Parametri
Pro (+) e Contro (-)

- Copia di valori
  - \+ permette la trasmissione del valore di un parametro dal chiamante
  - \+ permette la separazione tra programma chiamante e programma chiamato
  - – aumenta l'occupazione di memoria ed il tempo
  - – rende difficile la gestione di parametri di dimensione variabile
- Trasmissione per riferimento
  - \+ evita problemi di passaggio perché il trattamento degli indirizzi è gestito direttamente dal compilatore
  - \+ non occupa memoria aggiuntiva
  - – causa effetti collaterali spesso imprevedibili

Passaggio di Parametri
per Valore

- Generalmente usato per parametri che
  - Rappresentano un argomento e non il risultato di un sottoprogramma
    - Inutile consentirne la modifica

Passaggio di Parametri
per Referenza

- Usato più spesso quando il parametro
  - Rappresenta un risultato
    - Necessità di conoscere la modifica
  - Ha dimensioni notevoli
    - Pesante ricopiarlo interamente
- Potenziale fonte di errori
  - Stessa variabile usata sotto diverse denominazioni
  - Errori nel sottoprogramma irrecuperabili

Procedure

- Sottoprogrammi il cui compito è quello di produrre un effetto
  - Modifica del valore di variabili
  - Comunicazione di informazioni all'utente
- L'intestazione di procedura (e la sua chiamata) includono
  - Nome della procedura
    - Paragonabile ad una nuova istruzione del linguaggio
  - Lista di parametri

Corso di Programmazione

Sottoprogrammi

Procedure e funzioni

Prof.ssa Teresa Roselli

roselli@di.uniba.it

Funzione

Sottoprogramma che ha come risultato il calcolo di un valore:

- è dotata di nome (*designatore* di funzione) a cui viene associato un "valore di ritorno"
- per il motivo precedente deve essere dotata di *tipo* che va dichiarato nell'intestazione della funzione
- per lo stesso motivo, nel corpo della funzione, il nome deve comparire come parte sinistra di un assegnazione per attribuire ad esso il valore da trasmettere al programma chiamante
- La chiamata di una funzione avviene inserendo il suo nome in una espressione (non può mai trovarsi a sinistra di una assegnazione nel programma chiamante)
- Una funzione ha *parametri* come una procedura

Funzione

Il nome di una funzione

- a destra di una assegnazione rappresenta il *valore* della funzione
- a sinistra di una assegnazione individua la *locazione di memoria* (corrisponde al concetto di variabile e può apparire solo all'interno della funzione stessa

Funzioni

- Indipendentemente dal linguaggio di programmazione una dichiarazione di funzione prevede un costrutto linguistico del tipo

funzione <identificatore> <lista di argomenti> **:** <tipo risultato>

  - Gli argomenti rappresentano i parametri di entrata
    - Andrebbero sempre passati per valore al fine di evitare effetti collaterali
  - Come nel caso delle procedure, è possibile definire all'interno della funzione una sezione dichiarativa con variabili locali

Funzioni
esempio

- Dichiarazione della funzione *resto*

```
FUNZIONE resto(a:integer;b:integer):integer
        BEGIN
                resto ⟵ a - (a DIV b) * b
        END
```

- Chiamata della funzione *resto* nel programma chiamante

```
        .
        .
        .
residuo ⟵ resto(x,y)
        .
        .
        .
```

Procedure vs. Funzioni

	Procedure	**Funzioni**
Tipo associato		X
Parametri	X	X
Possibilità di modificare il valore dei parametri	X	
Valore di uscita		X
Chiamata	autonoma	in un'espressione

SIDE EFFECT
effetti collaterali in procedure e funzioni

- Il side effect si verifica quando a seguito dell'attivazione di un sottoprogramma si ha una modifica delle variabili non locali al sottoprogramma (i parametri passati per riferimento e le variabili non locali possono esportare valori al di fuori del sottoprogramma)
- Il side effect è accettabile nelle procedure ma deve essere evitato nelle funzioni poiché produrrebbe più di un valore di ritorno.
- L'utilizzo di variabili globali impedisce di considerare il sottoprogramma come una entità completa e autoconsistente poiché non fa riferimento esclusivamente alle sue variabili, tipi e costanti.

Sottoprogrammi come Parametri

- Nella classe dei parametri di sottoprogrammi rientrano anche procedure e funzioni
  - Un sottoprogramma *F* può essere usato come parametro di un altro sottoprogramma *G*, quando *F* deve essere eseguito durante l'esecuzione di *G*
- Il parametro formale corrispondente ad un sottoprogramma
  - Riporta l'intestazione
    - I nomi del sottoprogramma e dei parametri possono cambiare
  - Deve avere parametri passati esclusivamente per valore

Sottoprogrammi come Parametri

- All'invocazione di un sottoprogramma avente come parametri altri sottoprogrammi
  - Il corrispondente parametro effettivo deve essere l'identificatore di una procedura o di una funzione con i medesimi requisiti riguardo a parametri o tipo del risultato
  - Durante l'esecuzione del corpo del sottoprogramma invocato, ogni occorrenza del parametro formale implica l'uso corrispondente del sottoprogramma fornito come parametro effettivo

Sottoprogrammi come Parametri esempio

Il sottoprogramma

```
sottoprogramma s(sottoprogramma p(x,a))
    begin
        p(x,a)
    end
```

può essere invocato come segue

```
s(pinco(m,n))
```

Attivazione di Sottoprogrammi

- Nei linguaggi tipo Pascal l'attivazione di sottoprogrammi è gestita tramite *pila*
  - Ad ogni attivazione di un'unità di programma
    - Viene creato un *record di attivazione*
    - Il record viene messo in cima alla pila
  - Al termine dell'attivazione
    - Il record è tolto dalla pila
    - La memoria viene rilasciata
    - Si perdono i legami tra parametri

Record di Attivazione
Informazioni contenute e dimensioni

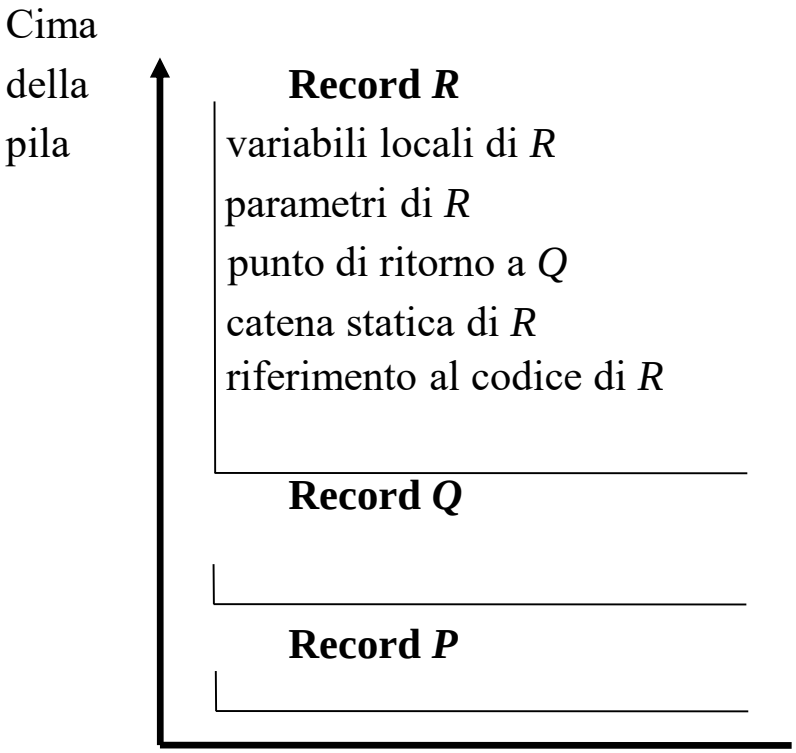
- Nome dell'unità di programma
- Riferimento all'area di memoria in cui è memorizzato il corpo di istruzioni da eseguire
- Punto di ritorno
  - Riferimento all'istruzione a cui tornare al termine dell'attivazione
- Gerarchia creata al momento della dichiarazione (*catena statica*)
- Parametri formali e loro legame con i corrispondenti parametri effettivi (dipende dal tipo di passaggio: il passaggio per referenza di un array fa occupare meno memoria rispetto al passaggio per valore)
- Variabili locali con riferimento alle aree di memoria allocate (dipende dal numero e dal tipo di variabili)

Concatenazione

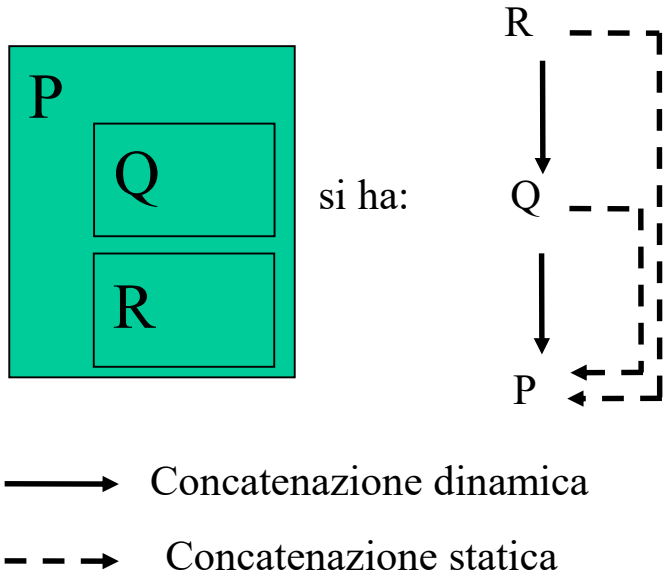
- Statica (può essere determinata guardando il testo del programma)
  - Definita dalla nidificazione nella dichiarazione delle procedure
    - non può variare
    - fornisce i riferimenti nella pila alle variabili non locali
- Dinamica (dipende dall'esecuzione)
  - Realizzata attraverso la sequenzializzazione delle attivazioni nella pila
    - Cambia a seconda dell'evolversi dell'esecuzione

Concatenazione
Esempio

- *P* ha attivato *Q*
  che ha attivato *R*



- Nel caso di dicharazioni nidificate



Gestione dell'Esecuzione

- Instruction pointer *IP*
  - Individua l'istruzione in corso di esecuzione nel programma
    - Riferita all'area di memoria associata al programma
- Environment pointer *EP*
  - Individua l'ambiente di lavoro
    - Risorse della porzione di programma attualmente in esecuzione

Gestione dell'Esecuzione

- Inizialmente:
  - *IP* punta al **begin**
  - *EP* non ha un ambiente cui puntare
- Nel momento in cui inizia l'esecuzione
  - Vengono allocate le variabili richieste dal programma principale
  - Le variabili vengono reperite ed usate nell'ambiente puntato da *EP*

Gestione dell'Esecuzione

- Al momento dell'attivazione di un sottoprogramma
  - Vengono allocate nuove variabili
    - Fanno parte di un nuovo ambiente
  - *EP* punta a questo ambiente
    - Il sottoprogramma userà le variabili puntate dall'*EP* attuale
    - Se non le trova, risalirà negli ambienti attivati precedentemente fino a trovare la prima occorrenza delle variabili

Gestione dell'Esecuzione

- Al termine dell'esecuzione di un sottoprogramma
  - *EP* punta all'ambiente precedente nella catena di attivazioni
  - Le variabili locali relative al sottoprogramma vengono distrutte

Ricorsione

- Si dimostra che
  - Ogni problema ricorsivo è computabile per mezzo di un programma

  e, viceversa,
  - Ogni problema computabile per mezzo di un programma è esprimibile in forma ricorsiva
- Le scomposizioni ricorsive implicano, a livello di codice, programmi in grado di invocare se stessi
  - procedure o funzioni ricorsive

Ricorsione

- Gestita come una normale attivazione di procedura o funzione
  - Disciplina a stack
  - Per ogni variabile *v* locale alla procedura *R*
    - Una chiamata di *R* che dia vita alla generazione di *k* chiamate ricorsive produrrà *k*+1 distinte istanze della variabile *v*
    - Il tempo di vita di ciascuna di esse è contenuto (innestato) in quello delle altre che la precedono

Ricorsione
Esempio

- Calcolo del fattoriale

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

– esprimibile ricorsivamente come

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n - 1)! & \text{se } n > 0 \end{cases}$$

Ricorsione
Esempio

- Sia nfatt(n) la funzione che calcola il fattoriale di *n* per ogni intero $n \geq 0$

  Se n = 0

  allora nfatt(n) = 1

  altrimenti nfatt(n) = n * nfatt(n – 1)

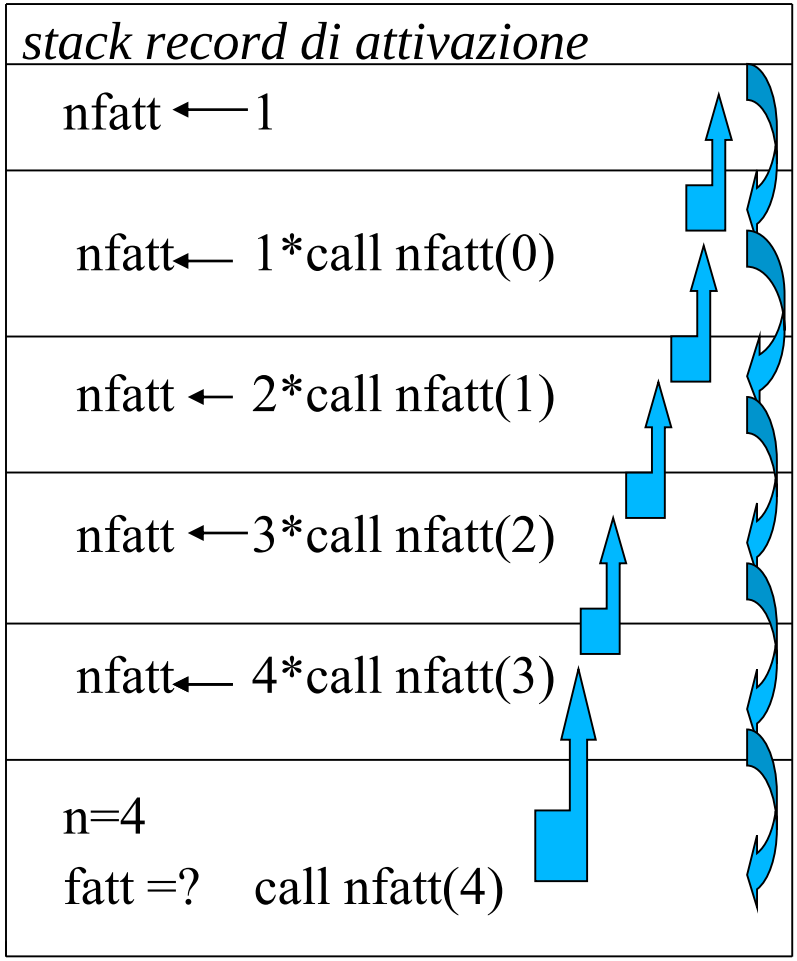
- L'esecuzione della funzione *nfatt* causa chiamate ricorsive della stessa funzione
  - Sovrapposizione di ambienti di programmazione nello stack in fase di esecuzione

Ricorsione
Esempio

- 4!: poiché 4 > 0, 4! = 4 * 3! = 4 * ?…
  - 3!: poiché 3 > 0, 3! = 3 * 2! = 3 * ?…
    - 2!: poiché 2 > 0, 2! = 2 * 1! = 2 * ?…
      - 1!: poiché 1 > 0, 1! = 1 * 0! = 1 * ?…
        - » 0!: è noto che 0! = 1
      - … = 1 * 1 = 1
    - … = 2 * 1 = 2
  - … = 3 * 2 = 6
- … = 4 * 6 = 24

Ricorsione
Esempio

memoria istruzioni	
FUNZIONE	p=6
nfatt(n:integer):integer	
BEGIN	p=5
if n=0 then nfatt←1	
else nfatt← n*nfatt(n-1)	p=4
END	
BEGIN (del main)	p=3
leggi n	
fatt← nfatt(n)	p=2
scrivi fatt	
END	p=1
	p=passo



Corso di Programmazione
Algoritmi Fondamentali

Prof.ssa Teresa Roselli
roselli@di.uniba.it

Minimo fra 3 valori

- Trovare il minore fra tre numeri reali a, b, c
  - Esempi:
    - $a = 1$, $b = 2$, $c = 3$
    - $a = 20$, $b = 50$, $c = 55$
    - $a = 2$, $b = 1$, $c = 3$
    - $a = 7$, $b = 34$, $c = 13$
- Operatore di confronto $x \leq y$
  - È verificato se il valore associato alla variabile x è minore di quello associato alla variabile y

Minimo fra 3 valori

Algoritmo

se $a \leq b$

allora se $a \leq c$

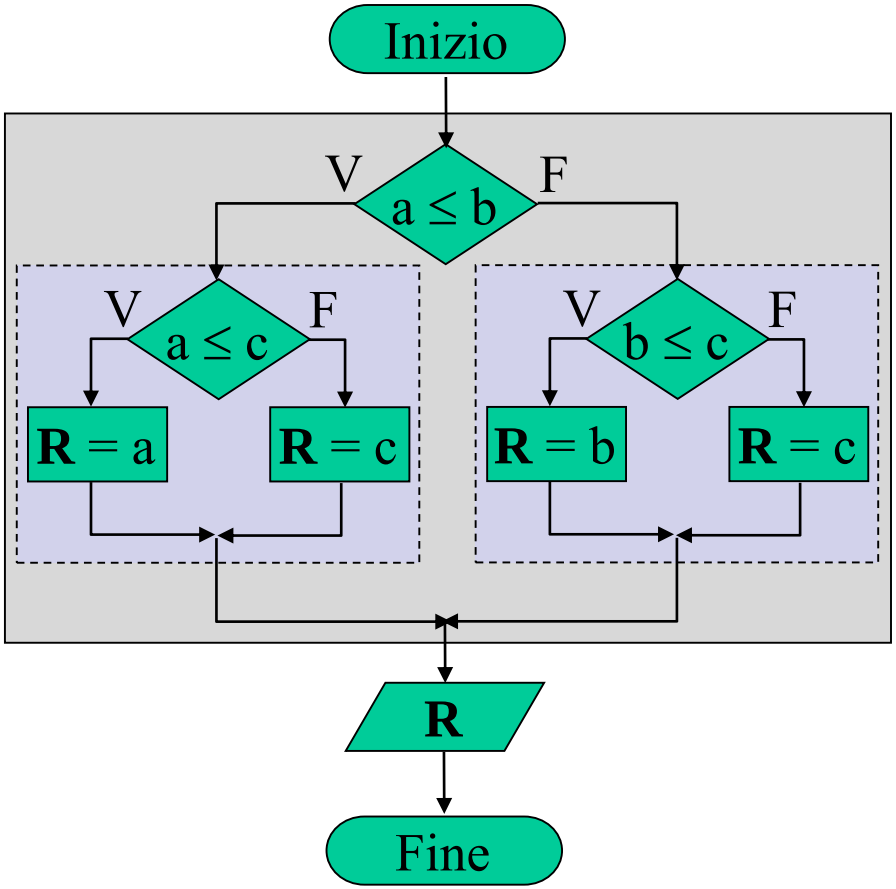
allora soluzione = a

altrimenti soluzione = c

altrimenti se $b \leq c$

allora soluzione = b

altrimenti soluzione = c



Minimo fra 3 valori
Considerazioni

- Analogo per qualunque dominio ordinale
  - Necessità di un operatore di confronto
  - Esempi:
    - Date 3 lettere, stabilire qual è la prima in ordine alfabetico
    - Dati 3 giorni della settimana, stabilire quale viene prima dal lunedì alla domenica

Scambio di valori

- Date due variabili *a* e *b*, scambiare i valori ad esse assegnati
  - Esempio:
    - $a = 12, b = 54 \rightarrow a = 54, b = 12$
- Operatore di assegnamento $x \leftarrow y$
  - Copia il valore associato alla variabile *y* nella memoria associata alla variabile *x*
    - Al termine i valori di *x* ed *y* coincidono
    - Il vecchio valore di *x* viene perso

Scambio di valori
Algoritmo (tentativo)

Assegna il valore di *b* ad *a*

Assegna il valore di *a* a *b*

– Trace:
  - *a* = 12, *b* = 54
  - *a* = 54, *b* = 54
  - *a* = 54, *b* = 54

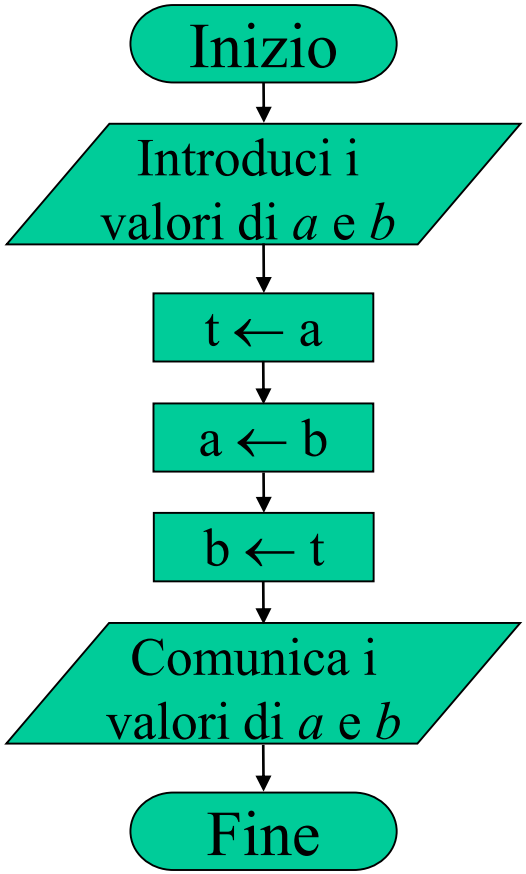
- ***Non funziona!***
  – Dopo il primo passo il valore originario di *a* viene perso
  – Serve una variabile temporanea *t* in cui copiare il valore di *a* prima che sia sovrascritto

Scambio di valori
Algoritmo (corretto)

Assegna il valore di *a* a *t*

Assegna il valore di *b* ad *a*

Assegna il valore di *t* a *b*



– Trace:
  - *a* = 12, *t* = ??, *b* = 54
  - *a* = 12, *t* = 12, *b* = 54
  - *a* = 54, *t* = 12, *b* = 54
  - *a* = 54, *t* = 12, *b* = 12

Scambio di valori
Considerazioni

- Ad ogni passo dell'algoritmo una variabile assume sempre il valore definito dalla assegnazione più recente per essa
- L'applicazione di un algoritmo a casi particolari può aiutare a scoprire gli errori
  - Scelta oculata degli esempi critici
    - Non valori uguali per le due variabili
- Quante variabili temporanee servono per una rotazione dei valori contenuti in *n* variabili?

Conteggio

- Dato un insieme di *n* valori, contare il numero di valori maggiori di una soglia *s*
  - Esempio:
    Studenti che hanno superato un esame (*s* = 17)
    - Voti 22 30 17 25 4 18 27
- Accumulatore
  - Area di memoria interna contenente i risultati parziali dei calcoli

Conteggio

- Ad ogni passo, se il voto supera la soglia, si compiono le seguenti azioni:

  conteggio attuale ← conteggio precedente + 1

  conteggio precedente ← conteggio attuale

  – Alla destra del primo assegnamento si ha un'espressione e non una variabile
    - Il risultato andrà nell'accumulatore in attesa di assegnarlo alla variabile a sinistra

Conteggio
Considerazioni

- L'assegnamento è diverso dalla relazione matematica di uguaglianza
  - Il simbolo a sinistra dell'assegnamento deve essere un identificatore
    - Lo stesso simbolo, posto a destra, indica il valore prima dell'assegnamento
      - $x \leftarrow x + 1$ ha senso
      - $x + 1 \leftarrow x$ non ha senso

Conteggio
Algoritmo

leggi il numero *n* di voti da elaborare

inizializza il contatore a 0

mentre ci sono ancora voti da esaminare **esegui**

    leggi il voto successivo

    se voto > 17

     allora somma 1 al contatore

comunica il numero totale di promossi

Somma

- Dato un insieme di *n* numeri, progettare un algoritmo che li sommi e restituisca il totale risultante
  - Ipotesi: $n \geq 0$
  - Esempio:
    - Sommare 46, 2 e 284
      - somma ← 46 + 2 + 284
        Troppo specifico sui valori!
      - somma ← a + b + c
        Troppo specifico sul numero di valori!

Somma

- Non è possibile scrivere un'assegnazione che valga per qualunque insieme di *n* valori, ma:
  - Gli elaboratori sono adatti ai compiti ripetitivi
  - Il sommatore ha capienza di 2 numeri per volta

Somma

- Si può usare un accumulatore per contenere le somme parziali:
  - $s \leftarrow a_1 + a_2$
  - $s \leftarrow s + a_3$
  - …
  - $s \leftarrow s + a_n$

- Compattando i vari passi:

$s \leftarrow a_1 + a_2$

finché ci sono ancora numeri da sommare

leggi il numero successivo a_{i+1}

$s \leftarrow s + a_{i+1}$

- Non funziona per la somma di 0 o 1 valore

Somma
Algoritmo

considera il numero *n* di
addendi

$s \leftarrow 0$

finché ci sono ancora numeri
da sommare esegui

leggi il numero successivo
a_{i+1}

$s \leftarrow s + a_{i+1}$

comunica che la somma
finale è s

Somma
Considerazioni

- Per sommare n numeri sono necessarie $n - 1$ addizioni
  - Se ne usano n per generalità
- Ad ogni iterazione s rappresenta la somma dei primi i valori
  - Per $i = n$ si ha la somma totale
- Il valore di i cresce ad ogni iterazione
  - Prima o poi $i = n$ e si uscirà dal ciclo (Terminazione)
- Non è assicurata la precisione della somma finale
  - Sommare i numeri per valore assoluto crescente

Successione di Fibonacci

- Generare i primi *n* termini della successione di Fibonacci (n ≥ 1)
  - I primi due termini sono 0 e 1
  - Ogni termine successivo è ottenuto come somma degli ultimi 2 termini
    - 1, 2, 3, 5, 8, 13, 21, 34, …

Successione di Fibonacci

- Necessario un ciclo di generazioni
  - In ogni istante, bisogna conoscere gli ultimi due termini generati per generare il successivo
    - Ultimo
    - Penultimo
  - Dopo la generazione
    - Il termine appena generato diventa l'ultimo
    - Il termine che prima era l'ultimo diventa penultimo

Successione di Fibonacci
Algoritmo

Acquisire il numero di termini da generare *n*

primo (e penultimo) termine ← 0

secondo (e ultimo) termine ← 1

termini generati ← 2

fintantoché termini generati < *n* **esegui**

    nuovo termine ← penultimo + ultimo

    penultimo ← ultimo

    ultimo ← nuovo termine

    termini generati ← termini generati + 1

Inversione delle Cifre di un Intero

- Progettare un algoritmo che accetta un intero positivo e ne inverte l'ordine delle cifre
  - Esempio
    $27953 \rightarrow 35972$
- Operatori DIV e MOD
  - Calcolano, rispettivamente, il quoziente intero ed il resto di una divisione fra interi

Inversione delle Cifre di un Intero

- Serve accedere alle singole cifre del numero
  - Non è nota a priori la lunghezza del numero
    - Iniziare dalla cifra meno significativa
- Eliminare progressivamente le cifre dal numero originario
  - Procedere da destra verso sinistra
- Accodarle progressivamente al numero invertito
  - Far scalare le precedenti verso sinistra

Inversione delle Cifre di un Intero

- Individuazione della cifra meno significativa
  - c = n MOD 10
- Rimozione della cifra meno significativa
  - n' = n DIV 10
- Scalare verso sinistra l'inversione parziale per aggiungere la nuova cifra
  - m * 10 + c
- Ripetere fino a quando non ci sono più cifre
  - Quoziente della divisione pari a 0

Inversione delle Cifre di un Intero Algoritmo

Stabilire *n*

Predisporre a 0 l'intero rovesciato

Finché l'intero da rovesciare è maggiore di 0

- Estrarre dall'intero da rovesciare la cifra meno significativa come resto della divisione per 10
- Aggiornare l'intero rovesciato moltiplicandolo per 10 e aggiungendo la cifra estratta
- Eliminare dall'intero da rovesciare la cifra estratta dividendolo per 10

Inversione delle Cifre di un Intero
Algoritmo

read *n*
inverso:= 0
while *inverso* > 0 do
    begin
    c:= *n* MOD 10
    inverso:= *inverso* * 10 + *c*
    n:= n DIV 10
    end
write *inverso*

Algoritmi su Array

- Algoritmi di base
- Algoritmi di supporto
  - Partizionamento
  - Fusione
- Ricerca
  - Basata su un ordinamento
- Ordinamento
  - Finalizzato alla ricerca

Acquisizione di un Array

- Dichiarazione del tipo opportuno
  - Dimensione fissa
    - Sufficiente all'uso previsto
    - Indice massimo attualmente usato

inizializza l'indice

fintantoché c'è un nuovo dato e l'array non è pieno

incrementa l'indice

leggi il nuovo dato e inseriscilo nell'array

  - Al termine l'indice specifica il numero di elementi
- N.B.: NON è possibile leggere una stringa in un unico passo

Massimo

- Trovare il massimo valore in un insieme di *n* elementi
  - Il computer non può guardare globalmente i valori nell'insieme
    - Analizzarli uno per volta
      - Necessità di esaminarli tutti per trovare il massimo
    - Ricordare in ogni momento il massimo parziale
      - Guardando solo il primo, è il più alto che si sia visto
      - Guardando i successivi, si aggiorna il massimo ogni volta che se ne trova uno maggiore del massimo parziale

Massimo
Algoritmo

- Inserire il numero di valori
- Leggere il primo
- Porre il massimo al primo
- Mentre non si sono letti tutti gli elementi
  - Leggere il successivo
  - Se è maggiore del massimo
    - Porre il massimo a questo numero
- Comunicare il massimo

Massimo
Considerazioni

- Necessari $n - 1$ confronti
- Si devono analizzare tutti gli elementi fino all'ultimo
  - Noto se l'insieme è conservato in un array
  - Ignoto se la sequenza di valori viene inserita progressivamente
    - Va chiesto quanti elementi compongono la sequenza

Inversione

- Risistemare gli elementi di un array di dimensione *n*, in modo tale che appaiano al contrario

  | a | k | h | e | x | b | h | → | h | b | x | e | h | k | a |

  – Inserirli in ordine inverso in un array di appoggio, quindi ricopiarlo in quello originale?
    - Spreco di memoria
    - Spreco di tempo

Inversione

- Effetto dell'inversione
  - Il primo diventa l'ultimo
  - Il secondo diventa il penultimo
  - …
  - Il penultimo diventa il secondo
  - L'ultimo diventa il primo
- Soluzione: scambiare il primo con l'ultimo, il secondo col penultimo, ecc.

Inversione

- 2 indici
  - Partono dagli estremi dell'array
  - Procedono verso l'interno
    - Ogni volta che il primo viene incrementato, il secondo viene decrementato
  - Si scambiano i valori via via incontrati
- Basta un solo indice
  - Conta la distanza dagli estremi (sia destro che sinistro)
    - (i, n – i)

Inversione

- Proviamo
  - $i = 1 \rightarrow (1, n - 1)$
    - *n* è stato saltato NON FUNZIONA!!!
    - Per considerarlo si può aggiungere 1
- Riproviamo
  - $i = 1 \rightarrow (1, n - 1 + 1) = (1, n)$
  - $i = 2 \rightarrow (2, n - 2 + 1) = (2, n - 1)$
  - …
- OK!

Inversione

- Quando fermarsi?
  - Ogni elemento si scambia col simmetrico
    - Se sono pari, gli scambi sono la metà degli elementi
    - Se sono dispari, quello centrale resterà al centro
      - E' già sistemato, non si scambia

□ $\lfloor n/2 \rfloor$ scambi

Inversione
Algoritmo e Programma Pascal like

- Stabilire l'array di *n* elementi da invertire
- Calcolare il numero di scambi *r* = *n* DIV 2
- Mentre il numero di scambi *i* è minore di *r*
  - Scambiare l'*i*-mo elemento con l'(*n*–*i*+1)-mo
- Restituire l'array invertito

```
begin
    r := n DIV 2;
    for i := 1 to r do
     begin
          t := a(i);
          a(i) := a(n–i+1);
          a(n–i+1) := t
     end
end.
```

- Aggiungere dichiarazioni, acquisizione dell'array e stampa del risultato

Rimozione dei Duplicati

- Dato un array ordinato, compattarlo in modo che i valori duplicati siano rimossi

  | 3 | 3 | 5 | 8 | 10 | 10 | 10 | 14 | 20 | 20 |

  → | 3 | 5 | 8 | 10 | 14 | 20 |

  – I doppi sono adiacenti
    - Saltarli
      – Confrontare elementi a coppie
    - Spostare il successivo valore distinto nella posizione seguente al più recente valore distinto
      – Contatore degli elementi distinti

Rimozione dei Duplicati
Algoritmo

Definire l'array di *n* elementi
Impostare a 2 l'indice di scansione
Finché si trovano valori distinti
    Confrontare coppie di elementi adiacenti
Impostare il numero di valori distinti
Finché ci sono coppie da esaminare
    Se la prossima coppia non ha duplicati
        Incrementa il contatore dei valori distinti
        Sposta l'elemento di destra della coppia in tale posizione

Rimozione dei Duplicati
Note Implementative

- Inizializzazione dell'indice di scansione
  - Prima posizione dell'array
    - Confronti col valore successivo
  - Seconda posizione dell'array
    - Confronti col valore precedente
- Seconda opzione più intuitiva
  - Consente di terminare il ciclo ad *n*

Ricerca

- Determinare se (e dove) un certo elemento *x* compare in un certo insieme di *n* dati
  - Supponiamo gli elementi indicizzati 1 … *n*
    - Il fatto che l'elemento non sia stato trovato è rappresentabile tramite il valore di posizione 0
    - Deve funzionare anche per 0 elementi
  - Possibili esiti:
    - Elemento trovato nell'insieme
      - Restituirne la posizione
    - Elemento non presente nell'insieme

Ricerca Sequenziale

- Scorrimento di tutti gli elementi della sequenza, memorizzando eventualmente la posizione in cui l'elemento è stato trovato
  - Nessuna ipotesi di ordinamento
  - Utilizzabile quando si può accedere in sequenza agli elementi della lista

Ricerca Lineare Esaustiva
Algoritmo

- Scandisce tutti gli elementi dell'elenco
  - Restituisce l'ultima (posizione di) occorrenza
  - Utile quando si vogliono ritrovare tutte le occorrenze del valore

```
j ← 0
posizione ← 0
fintantoché j < n
    j ← j + 1
    se lista(j) = x allora posizione ← j
```

Ricerca Lineare Esaustiva
Programma Pascal like

- Completare con le dichiarazioni

```
begin
    j := 0;
    posizione := 0;
    while j < n do
      begin
            j := j + 1;
            if a(j) = x then posizione := j
      end
end
```

Ricerca Lineare Esaustiva
Considerazioni

- Complessità
  - Basata sul numero di confronti
    - In ogni caso: *n*
      - Si devono controllare comunque tutti gli elementi fino all'ultimo
- Soluzione più naturale
  - A volte non interessa scandire tutto l'elenco
    - Ci si può fermare appena l'elemento viene trovato

Ricerca Lineare con Sentinella
Algoritmo

- Si ferma alla prima occorrenza del valore
  - Restituisce la prima (posizione di) occorrenza
  - Utile quando
    - Si è interessati solo all'esistenza, oppure
    - Il valore, se esiste, è unico

```
j ← 0
posizione ← 0
fintantoché (j < n) e (posizione = 0)
    j ← j + 1
    se lista(j) = x allora posizione ← j
```

Ricerca Lineare con Sentinella
Programma Pascal like

- Completare con le dichiarazioni

```
begin
    j := 0;
    posizione := 0;
    while j < n and posizione = 0 do
     begin
          j := j + 1;
          if a[j] = x then posizione := j
     end
end
```

Ricerca Binaria
o Dicotomica

- Analizzare un valore interno all'elenco, e se non è quello cercato basarsi sul confronto per escludere la parte superflua e concentrarsi sull'altra parte
  - Applicabile a insiemi di dati ordinati
    - Guadagno in efficienza
    - Incentivo all'ordinamento
  - Richiede l'accesso diretto

Ricerca Binaria

- Scelta della posizione da analizzare
  - Più vicina ad uno dei due estremi
    - Caso migliore: restringe più velocemente il campo
    - Caso peggiore: elimina sempre meno elementi
  - Centrale
    - Compromesso che bilancia al meglio i casi possibili
- Necessità di ricordare la porzione valida
  - Prima posizione
  - Ultima posizione

Ricerca Binaria
Algoritmo

Fintantoché la parte di elenco da analizzare contiene più di un elemento e quello cercato non è stato trovato

Se l'elemento centrale è quello cercato

allora è stato trovato in quella posizione

altrimenti se è minore di quello cercato

allora analizzare la metà elenco successiva

altrimenti analizzare la metà elenco precedente

Ricerca Binaria
Esempio

$x = 29$ | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | 38 | 44 | 53 | 61 |

12 elementi

I tentativo ~~| 2 | 4 | 7 | 11 | 24 | 25 |~~ 29 | 32 | 38 | 44 | 53 | 61 |

↑ Eliminati 6

II tentativo | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | ~~38 | 44 | 53 | 61 |~~

↑ Eliminati 4

III tentativo | 2 | 4 | 7 | 11 | 24 | 25 | 29 | 32 | 38 | 44 | 53 | 61 |

↑ Trovato!

- Se fosse stato 31: non trovato in 4 tentativi

Ricerca Binaria
Programma Pascal like

```
begin
    posizione := 0;
    first := 1; last := n;
    while (first <= last) and (posizione = 0) do
     begin
          j := (first + last) DIV 2;     (* per difetto *)
          if lista(j) = x then
               posizione := j
          else
               if lista(j) < x then first := j + 1 else last := j – 1
     end
end
```

Ordinamento

- Disporre gli elementi di un insieme secondo una prefissata relazione d'ordine
  - Dipendente dal tipo di informazione
    - Numerica
      - Ordinamento numerico
    - Alfanumerica
      - Ordinamento lessicografico
  - 2 possibilità
    - Crescente
    - Decrescente
  - Altri criteri personali

Ordinamento

- Possibilità di avere elementi costituiti da più componenti
  - Associazione a ciascuno di una *chiave*
    - Lo identifica univocamente
    - Stabilisce la sua posizione
    - Unica componente rilevante per l'ordinamento
  - Esempio: Record costituito da più campi
- Supponiamo, nel seguito, di richiedere un ordinamento numerico crescente
  - Indicazione della sola chiave

Ordinamento

- Gran varietà di algoritmi
  - Basati su confronti e scambi fra gli elementi
    - Relazione d'ordine, criterio
  - Non esiste uno migliore in assoluto
    - La bontà dipende da fattori connessi ai dati su cui deve essere applicato
      - Dimensione dell'insieme di dati
        - » Numerosità
      - Grado di pre-ordinamento dell'insieme di dati
        - » Già ordinato, parzialmente, ordine opposto, casuale

Ordinamento

- Una delle attività di elaborazione più importanti
  - Stima: 30% del tempo di calcolo di un elaboratore
- Obiettivo: efficienza
  - Sfruttare “al meglio” i confronti ed i conseguenti spostamenti degli elementi
    - Piazzare gli elementi prima possibile più vicino alla loro posizione finale nella sequenza ordinata

Ordinamento

- Algoritmi esterni
  - Usano un array di appoggio
    - Occupazione di memoria doppia
    - Copia del risultato nell'array originale
- Algoritmi interni
  - Eseguono l'ordinamento lavorando sullo stesso array da ordinare
    - Scambi di posizione degli elementi

Ordinamento Esterno

- Enumerativo
  - Ciascun elemento confrontato con tutti gli altri per determinare il numero degli elementi dell'insieme che sono più piccoli in modo da stabilire la sua posizione finale

Ordinamento Interno

- Per Selezione
  - Elemento più piccolo localizzato e separato dagli altri, quindi selezione del successivo elemento più piccolo, e così via
- A bolle
  - Coppie di elementi adiacenti fuori ordine scambiate, finché non è più necessario effettuare alcuno scambio
- Per Inserzione
  - Elementi considerati uno alla volta e inseriti al posto che gli compete all'interno degli altri già ordinati

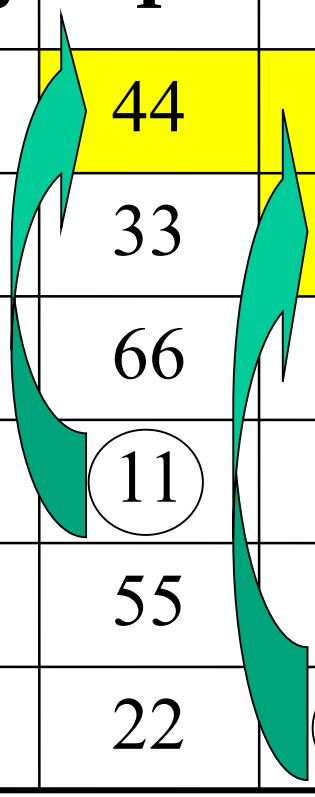
Ordinamento per Selezione

- Minimi successivi
  - Trovare il più piccolo elemento dell'insieme e porlo in prima posizione
    - Scambio con l'elemento in prima posizione
  - Trovare il più piccolo dei rimanenti ($n - 1$) elementi e sistemarlo in seconda posizione
  - …
  - Finché si trovi e collochi il penultimo elemento
    - Ultimo sistemato automaticamente

Ordinamento per Selezione

Esempio

	Inizio	**I**	**II**	**III**	**IV**	**V**
array(1)	44	44	11	11	11	11
array(2)	33	33	33	22	22	22
array(3)	66	66	66	66	33	33
array(4)	11	11	44	44	44	44
array(5)	55	55	55	55	55	55
array(6)	22	22	22	33	66	66



Ordinamento per Selezione
Algoritmo

$i \leftarrow 1$

fintantoché $i < n$ **esegui**

trova il minimo di lista (i … n)

scambia la posizione del minimo con lista(i)

$i \leftarrow i + 1$

- Utilizza algoritmi noti
  - Ricerca del minimo
  - Scambio

Ordinamento per Selezione
Programma Pascal like

```
begin
    for i := 1 to n – 1 do
     begin
          min := a(i]) p := i;
          for j := i + 1 to n do
               if a(j) < min then
                    begin     min := a(j); p := j end;
          a(p) := a(i);
          a(i) := min
     end
end.
```

Ordinamento per Selezione
Complessità

- Confronti
  - Sempre $(n - 1) * n / 2$
    - $n - 1$ al I passo di scansione
    - $n - 2$ al II passo di scansione
    - …
    - 1 all'$(n - 1)$-mo passo di scansione
- Scambi
  - Al più $(n - 1)$
    - 1 per ogni passo

Ordinamento per Selezione
Considerazioni

- Ogni ciclo scorre tutta la parte non ordinata
- Numero fisso di confronti
  - Non trae vantaggio dall'eventuale preordinamento
- Pochi scambi

Algoritmi su File

- Algoritmi basilari di creazione e modifica, ed inoltre
  - Fusione
  - Ordinamento
- Supponiamo di operare su file di record
  - Identificazione tramite campo chiave numerico intero
    - Struttura del record: | *chiave* | …altri dati… |
- Operazione dominante rispetto alla quale valutare la complessità
  - Lettura/scrittura di un elemento di un file

File

- Aggiunta di elementi ad un file esistente
  - Non vanno cancellati gli elementi esistenti
    - Dopo averli scorsi in lettura non si può scrivere
  - Necessità di un file di appoggio

Aprire in lettura il file originale ed in scrittura il file di appoggio
Finché non si è raggiunta la posizione di inserimento ed il file non è finito
    copiare un elemento dal file originale a quello di appoggio
Inserire l'elemento da aggiungere nel file di appoggio
Finché non si è raggiunta la fine del file
    copiare un elemento dal file originale a quello di appoggio
Riversare il file di appoggio nel file originale (copia)

File

- Modifica di un elemento
  - Bisogna individuarne la posizione
    - Dopo averlo scorso il file in lettura non vi si può scrivere
  - Necessità di un file di appoggio

Aprire in lettura il file originale ed in scrittura il file di appoggio
Finché non si è raggiunto l'elemento da modificare
    copiare un elemento dal file originale a quello di appoggio
Inserire l'elemento modificato nel file di appoggio
Finché non si è raggiunta la fine del file
    copiare un elemento dal file originale a quello di appoggio
Riversare il file di appoggio nel file originale (copia)