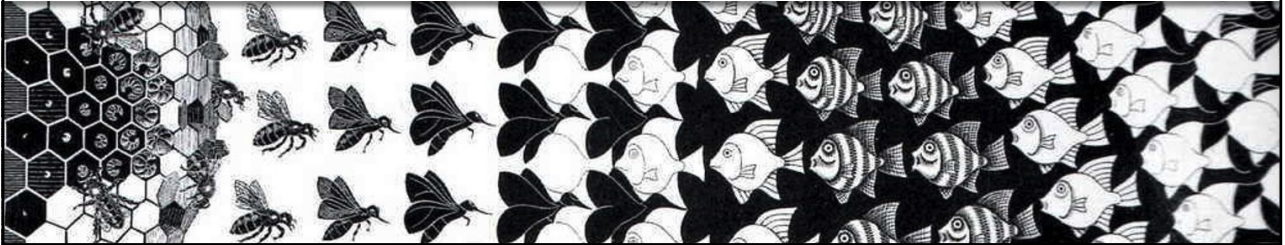


Appunti del corso di Informatica 1 (IN110)
2 – Algoritmi e diagrammi di flusso

Prof. Marco Liverani

(liverani@mat.uniroma3.it – <http://www.mat.uniroma3.it/users/liverani/IN110>)



Sommario

- Caratteristiche dell'esecutore delle procedure di calcolo
- Compiti del progettista/programmatore
- Competenze ed abilità del progettista/programmatore
- Capacità del calcolatore
- Un esempio elementare
- Linguaggi procedurali
- Diagrammi di flusso
- Programmazione strutturata
- Algoritmi ed implementazione
- Un esempio: il calcolo del massimo fra 2 , 3 , n numeri

Caratteristiche dell'esecutore

1

- **È una macchina**, costituita da circuiti elettronici digitali e da componenti elettromeccaniche, ottiche e magnetiche.
- **È velocissimo**, essendo una macchina elettronica è molto rapido nel compiere le operazioni per cui è stato progettato.
- **È puntuale** nell'applicare le regole che conosce (è *preciso*, ma non nel senso matematico del termine).
- **È duttile** e si adatta bene ad eseguire nuove tecniche, purché questo gli venga spiegato in modo *dettagliato e privo di ambiguità*.
- **Ha una buona memoria**, estremamente ampia ed organizzata in modo razionale, ma parcellizzato.

Caratteristiche dell'esecutore

2

- **Non è intelligente**: qualunque sia l'accezione di questo termine, non è adatta a descrivere le caratteristiche di un computer
- **Non è in grado di compiere deduzioni** o ragionamenti di altro tipo in modo autonomo
- **Non è in grado di capire un problema**
- **Non è in grado di capire la soluzione** di un problema, né è in grado di capire in modo autonomo se il risultato raggiunto è la soluzione del problema

Compiti del programmatore

- **Analizzare il problema** riducendolo in termini astratti, eliminando ogni componente non indispensabile e **formulando un modello** del problema
- Individuare una **strategia risolutiva** e ricondurla ad un *algoritmo*
- **Codificare l'algoritmo** in modo tale da renderlo comprensibile al calcolatore
- **Analizzare il risultato** dell'elaborazione evidenziando eventuali errori nella formulazione del problema, nella strategia risolutiva, nella codifica dell'algoritmo

Competenze e abilità del programmatore

- Deve essere in grado di **capire i problemi** e **schematizzarli**, distinguendone le diverse **componenti** (dati in input, parametri del problema, dati in output)
- Deve essere in grado di risolvere problemi mediante un **approccio algoritmico**, individuando gli aspetti del problema che possano essere risolti **reiterando** più volte **operazioni simili**
- Deve conoscere i **metodi fondamentali** di risoluzione dei problemi, gli approcci più comuni, le strade notoriamente meno convenienti
- Deve conoscere a fondo le **caratteristiche** e le **capacità del calcolatore**
- Deve essere in grado di comunicare con il calcolatore: ne deve **conoscere il linguaggio**

Capacità del calcolatore

- Sa **memorizzare le informazioni**
- Sa **distinguere** in modo pignolo **tra informazioni di tipo numerico ed altri tipi di informazione** (parole, immagini, ecc.). Anche i numeri vengono trattati in modo differente a seconda dell'insieme a cui appartengono (naturali, relativi, razionali, ecc.)
- **Sa eseguire alcune operazioni elementari**: addizione, sottrazione, prodotto e rapporto fra numeri, concatenazione di parole
- **Sa eseguire il confronto** fra informazioni dello stesso tipo: confronto fra numeri (es.: $a > b$, $a = b$, $a \leq b$) e sa verificare l'uguaglianza fra due parole
- **Sa «leggere»** le informazioni dall'esterno (*input*)
- **Sa «scrivere»** le informazioni all'esterno (*output*)
- **Sa memorizzare sequenze di istruzioni** elementari (programma) e **le sa eseguire secondo un ordine stabilito** dal programma stesso

Un esempio elementare

1

- **Problema**: *Scrivere i primi 10 multipli di 7*
- Il problema è di una semplicità disarmante!
- Soluzione:

1. scrivi 7
2. scrivi 14
3. scrivi 21
4. scrivi 28
5. scrivi 35
6. scrivi 42
7. scrivi 49
8. scrivi 56
9. scrivi 63
10. scrivi 70
11. stop

Ma questa soluzione è inaccettabile!

- siamo noi, i **progettisti** della soluzione, che abbiamo calcolato la soluzione, non **l'esecutore**
- noi dobbiamo **progettare un metodo** per risolvere il problema (un algoritmo) non dobbiamo eseguire il calcolo
- il metodo proposto è troppo rigido: se devo scrivere i primi 5 o i primi 60 multipli di 7 devo modificare l'algoritmo; se devo scrivere i multipli di 3 e non di 7 devo modificare l'algoritmo...

Occorre trovare una diversa strategia!

Un esempio elementare

2

- **Problema:** *stampare i primi 10 multipli di x .*
- Il problema è semplice e chiaro:
 - un solo **dato in input** (un dato che caratterizza l'istanza del problema): la "base" costituita dal numero x
 - una **costante** del problema: il numero di multipli da stampare, 10.
- La **strategia risolutiva** è la seguente: *iniziando dal numero x letto in input, per 10 volte verrà calcolato e stampato il multiplo di x , moltiplicando x per una variabile i il cui valore verrà incrementato di una unità ad ogni passo, da 1 fino a 10.*

Un esempio elementare

3

Possiamo ricondurre la strategia al seguente **algoritmo**:

1. Leggi in input un numero e chiama x il numero letto
2. Assegna il valore 1 alla variabile i
3. Calcola $x \times i$ e assegna alla variabile y il risultato
4. Stampa in output y
5. Calcola $i+1$ e assegna alla variabile i il risultato
6. Se $i \leq 10$ allora vai al passo 3 altrimenti prosegui
7. Fermati

Algoritmi

- Un **algoritmo** è una descrizione dettagliata, per **passi elementari** successivi, di una strategia utile per risolvere un determinato problema
- Ogni algoritmo è costituito da un insieme **finito** di passi e il procedimento che descrive deve **terminare** dopo un numero finito di iterazioni
- Nella progettazione di un algoritmo il programmatore inizia a porsi problemi relativi alla **rappresentazione delle informazioni** che deve essere **efficiente** (senza sprechi inutili) ed **efficace** (non si deve perdere traccia di dati importanti)
- Naturalmente l'aspetto fondamentale è la progettazione di un **algoritmo corretto**, ma anche **efficiente**
 - Un algoritmo corretto *termina sempre* e produce una soluzione per il problema, non ripete all'infinito delle operazioni senza mai giungere ad un risultato
 - Un algoritmo efficiente consente all'esecutore di individuare la soluzione nel minor tempo possibile (eseguendo il minor numero di operazioni)
- Le istruzioni che lo compongono sono **elementari**: questo significa che devono poter essere eseguite dall'esecutore utilizzando le sue capacità di base; le istruzioni che possiamo utilizzare per comporre un algoritmo dipendono quindi dalle capacità dell'esecutore

Linguaggi procedurali

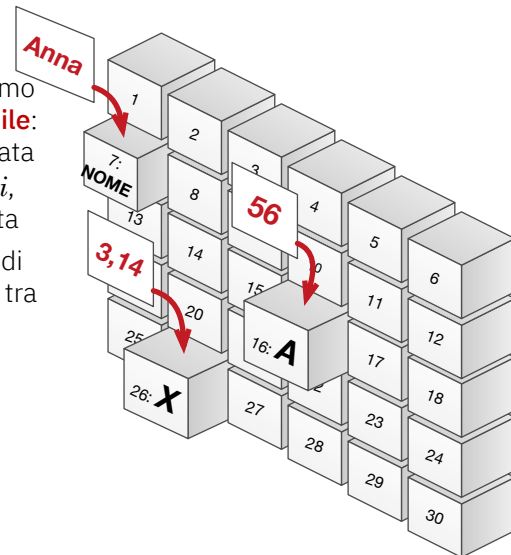
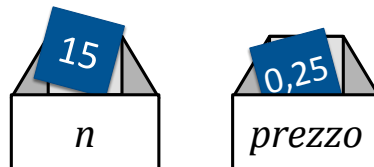
1

- Il nostro studio e l'approccio che adotteremo è focalizzato sull'uso di un *linguaggio procedurale imperativo* (il linguaggio "C")
- Questi linguaggi sono basati su **sei istruzioni fondamentali**:
 - **Assegna**: assegna ad una variabile (ad una locazione di memoria) il valore di una espressione.
 - **Leggi**: legge in input dall'esterno un valore e lo memorizza in una variabile (locazione di memoria).
 - **Scrivi**: scrive in output il valore di una espressione o di una variabile (locazione di memoria).
 - **Se ... allora ... altrimenti ...**: modifica il "flusso" del programma sulla base del valore di una espressione logica.
 - **Vai al passo ...**: modifica il "flusso" del programma incondizionatamente.
 - **Fermati**: termina l'esecuzione del programma.

Linguaggi procedurali

2

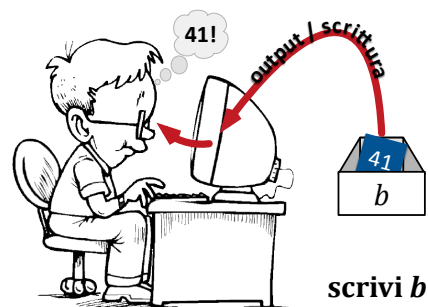
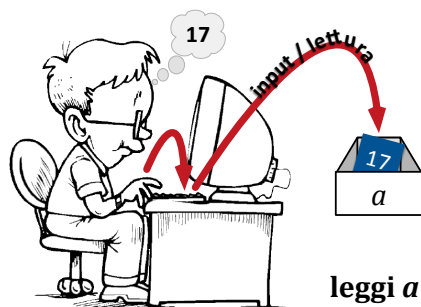
- Gli algoritmi sono costituiti da istruzioni che esprimono delle operazioni che devono essere eseguite dal calcolatore (esecutore)
- Lo strumento fondamentale nella codifica di un algoritmo con un linguaggio procedurale è costituito dalla **variabile**: è una porzione della memoria della macchina identificata da un'etichetta che ne costituisce il **nome** (es.: a , b , x , i , *risultato*, ecc.) e può contenere un solo **valore** per volta
- L'algoritmo è costituito da una sequenza di operazioni di assegnazione di un valore ad una variabile o confronto tra i valori di due variabili o espressioni (es.: $a=5$, $b=a+1$, $c = 17/24.2-b$)



Linguaggi procedurali

3

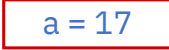




- Anche le operazioni di input (*leggi*) e di output (*scrivi*) consentono all'esecutore dell'algoritmo (il calcolatore) di compiere delle operazioni con le variabili di memoria:
 - **scrivi x** : estrae dalla variabile di memoria x il suo valore e lo presenta in output all'utente (es.: lo visualizza sullo schermo del computer)
 - **leggi y** : acquisisce in input un dato dall'utente (es.: l'utente inserisce un dato digitandolo sulla tastiera) e l'esecutore lo memorizza nella variabile di memoria denominata y



Diagrammi di flusso

1

Per rappresentare in modo efficace un algoritmo sono stati sviluppati dei *modelli grafici* (i **diagrammi di flusso**) che associano alle istruzioni del programma dei simboli grafici:

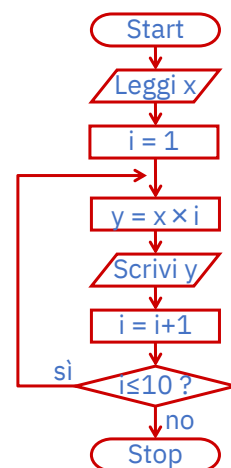
Assegnazioni	
Input/Output	
Condizioni	
Salti (vai al passo...)	
Start/Stop	

Diagrammi di flusso

2

- **Esempio:** «stampare i primi 10 multipli di x ».

1. Leggi x
2. $i = 1$
3. $y = x \times i$
4. Scrivi y
5. $i = i + 1$
6. Se $i \leq 10$ vai al passo 3 altrimenti prosegui
7. Fermati

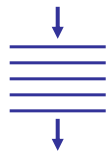


Programmazione strutturata

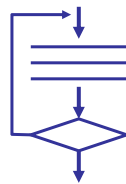
1

- I programmatori inesperti tendono ad “aggrovigliare” il programma introducendo numerosi salti privi di regole (*spaghetti programming*).
- È stato dimostrato (**Teorema fondamentale della programmazione strutturata** di Giuseppe Jacopini e Corrado Böhm) che ogni programma può essere codificato attenendosi esclusivamente a tre strutture fondamentali:

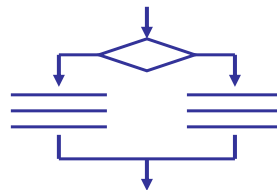
Sequenziale



Iterativa



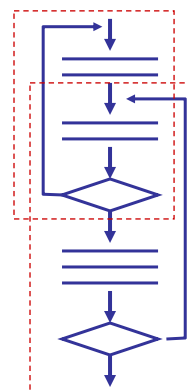
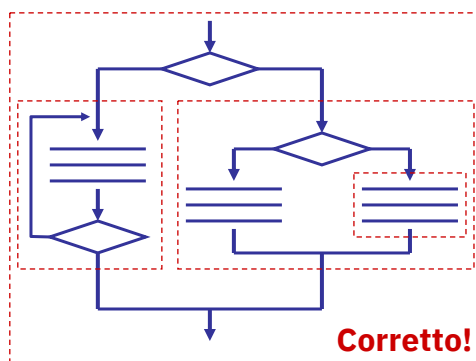
Condizionale



Programmazione strutturata

2

- Le tre strutture possono essere **concatenate** una di seguito all'altra oppure **nidificate** una dentro l'altra.
- Non possono essere “intrecciate” o “accavallate”.



Sbagliato!

È un tipico esempio di *spaghetti programming*

Programmazione strutturata

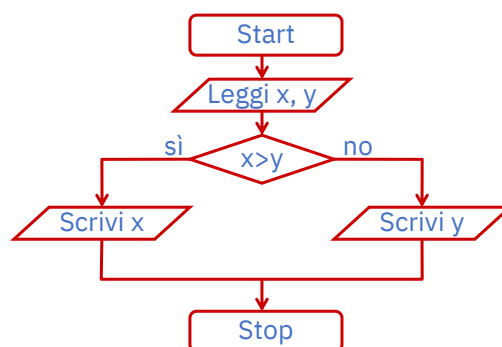
3

- L'istruzione di *salto incondizionato* (“**vai al passo ...**”) viene quindi rimpiazzata da un'istruzione di *salto condizionato* (“**se ... allora vai al passo ...**”) realizzato dalla struttura iterativa (più spesso come “**fintanto che la condizione ... è verificata vai al passo ...**”)
- Il salto incondizionato infatti, oltre ad essere **inutile** (è una delle conseguenze indirette del Teorema di Giuseppe Jacopini e Corrado Böhm) crea problemi nella **comprensione** e nella **manutenzione** del software

Algoritmi e implementazione

1

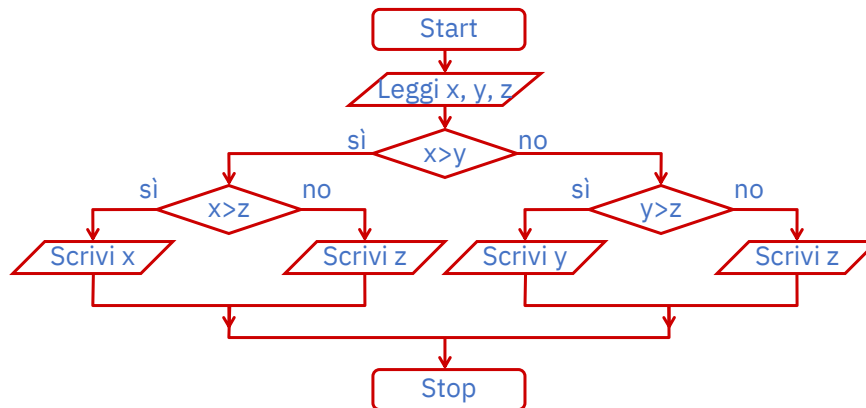
- Esempio: letti in input 2 numeri stampare il massimo.



Algoritmi e implementazione

3

- Esempio: letti in input 3 numeri stampare il massimo.



Algoritmi e implementazione

4

- Esempio: letti in input n numeri stampare il massimo. In questo caso non è possibile adottare la stessa strategia: dovremmo utilizzare troppe variabili (quante?) ed avremmo un algoritmo troppo articolato.
- È necessario adottare una diversa strategia: individuare una **operazione semplice** che **ripetuta più volte** porti alla soluzione.

