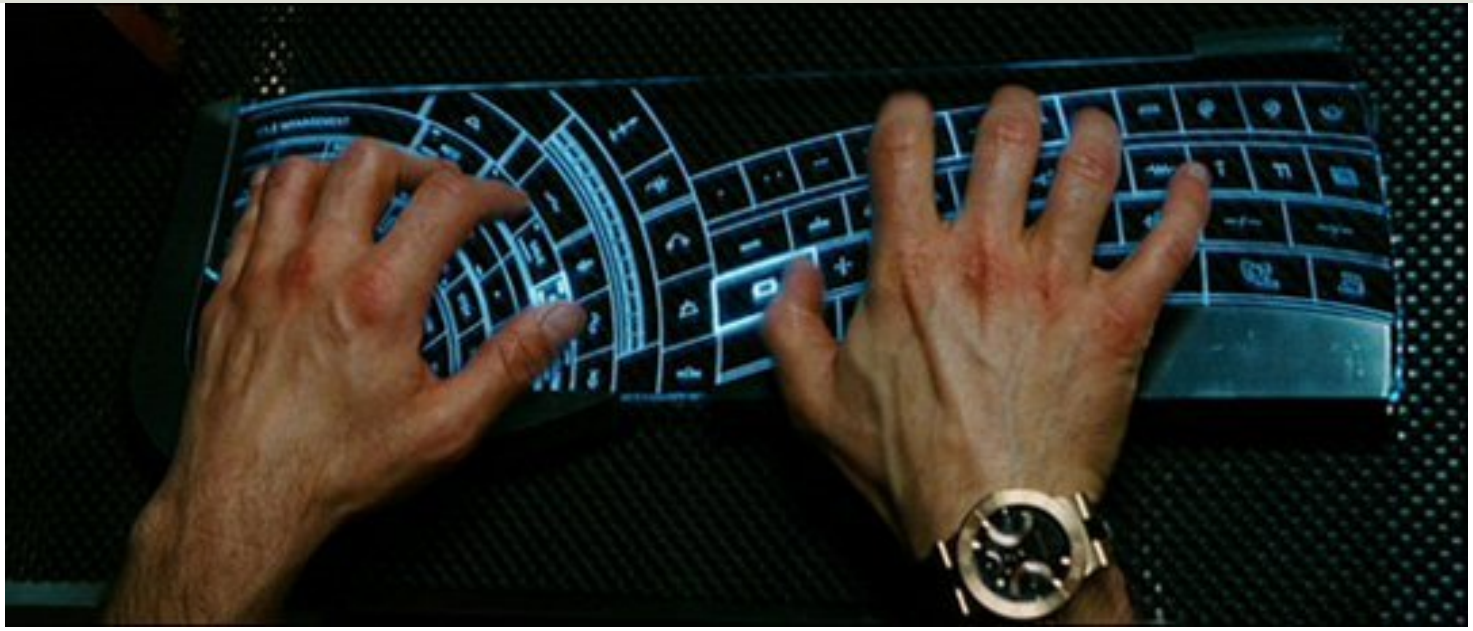


PARTE IX: Dagli Algoritmi ai linguaggi di programmazione

Nozioni

Introduzione ai concetti di:

- *Algoritmo*
- *Programma*
- *Dato*
- *Diagramma di flusso*
- *Linguaggio di programmazione*

Nozioni

Algoritmo = successione di operazioni elementari che possono essere eseguite da un calcolatore

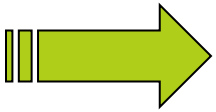
Programma = algoritmo in un linguaggio "comprensibile" dal computer

Dato = informazione da elaborare rappresentata in un formato che consenta al programma di operare su di essa

Diagramma di flusso = descrizione grafica della logica del programma

Nozioni

Linguaggio di programmazione



Linguaggio macchina = basato sul set di istruzioni della macchina, rappresentato da sequenze di 0 e 1

Linguaggio ad alto livello = linguaggio più vicino al linguaggio naturale, rigoroso e non ambiguo

Elaborazione dell'informazione

- Ogni problema legato all'elaborazione di informazioni è caratterizzato da:
  - un insieme di dati di partenza
  - un risultato cercato

- Ogni sua soluzione è:
  - una procedura che genera il risultato cercato a partire dall'insieme dei dati di partenza specificati

La soluzione

- La conoscenza di come si risolve un problema e la capacità di risolverlo sono competenze distinte

Es: ognuno è capace di riconoscere un volto, ma come avviene questo riconoscimento? Come descrivere la procedura per riconoscere uno specifico volto?

La procedura di soluzione

- Può capitare di trovarsi di fronte ad un problema la cui soluzione debba essere attuata non da noi, ma da un altro soggetto
- Il soggetto può non sapere come operare, sebbene possa dichiarare la sua disponibilità ad attuare la soluzione nel momento in cui gli venisse insegnata

La procedura di soluzione

La procedura di soluzione deve allora essere realizzata in fasi distinte e successive:

1. conoscenza di come si risolve un problema
   - ➔ analisi del problema e identificazione di una soluzione da parte del primo soggetto
   - ➔ descrizione della soluzione da parte del primo soggetto in termini comprensibili al secondo soggetto
2. effettiva capacità di risolvere un problema:
   - ➔ interpretazione della soluzione da parte del secondo soggetto
   - ➔ attuazione della soluzione da parte del secondo soggetto

La conoscenza di come si risolve un problema è ciò che ci permette di sviluppare un programma.

L'esecutore

- Il programma sarà poi interpretato ed eseguito da un esecutore
  - La procedura di soluzione deve essere descritta in una forma che l'esecutore sia in grado di interpretare in modo corretto
  - La soluzione deve specificare delle azioni che l'esecutore è in grado di attuare
- Un esecutore è caratterizzato dalle sue capacità di interpretazione e di attuazione

Il calcolatore

- I calcolatori sono degli esecutori di soluzioni che esseri umani hanno precedentemente identificato e descritto

- I calcolatori hanno una notevole velocità di esecuzione e possono ripetere la stessa operazione producendo sempre lo stesso risultato un numero elevato di volte

Il calcolatore

Il calcolatore in quanto esecutore è caratterizzato da:

- il linguaggio che è in grado di interpretare, con il quale devono essere descritte le soluzioni che vuole che esso attui;
- l'insieme delle azioni che è in grado di compiere
- l'insieme delle regole che ad ogni costrutto linguistico sintatticamente corretto associano le relative azioni da compiere

Elaborazione dell'informazione

Per la descrizione della soluzione si utilizza:

- linguaggio naturale - *es. italiano, inglese* – (ambiguo)
- linguaggio formale (non ambiguo):
  - formalismo matematico
  - pseudo-codice
  - diagramma di flusso
  - linguaggio di programmazione

La descrizione rigorosa di un metodo che consente di ottenere un risultato attraverso passi elementari si chiama algoritmo.

Problemi e algoritmi

- Se un problema è particolarmente semplice, l'esecutore potrebbe essere in grado di eseguire la soluzione direttamente.

Es. Determinare la superficie di un cerchio di raggio r.

- Ma se il risolutore non conosce la formula risolutiva la si deve indicare esplicitamente.

Es. $s=\pi r^2$.

- Tuttavia se l'esecutore non conosce come elevare un numero al quadrato, si ha che la soluzione contiene a sua volta un problema la cui soluzione deve essere descritta in modo esplicito.

Gerarchia di problemi

- In generale per giungere alla descrizione della soluzione di un problema si scompone il problema in sottoproblemi
- Ci si ferma quando si giunge ad un problema elementare o primitivo la cui soluzione corrisponde ad un'azione elementare che può essere direttamente compiuta dall'esecutore
- Risolvere un problema equivale a risolvere una opportuna successione di problemi più semplici

Soluzione effettiva

Definiamo effettiva per un esecutore la soluzione di un problema quando:

- l'esecutore è in grado di interpretare la descrizione di tale soluzione e associare ad essa le azioni che deve compiere per eseguirla
- l'esecutore è in grado di compiere tali azioni completandone l'esecuzione in un tempo finito

Ambiguità

- Finché la soluzione di un problema viene descritta in termini informali (come ad es. tra gli esseri umani) può rimanere l'ambiguità circa l'attuabilità della soluzione da parte dell'esecutore (la sua effettività).
- Si ha ambiguità quando due soggetti giudicano come effettiva la stessa soluzione di un problema ma poi compiono azioni che producono risultati differenti.
- Per rimuovere tale ambiguità si deve formalizzare la definizione di esecutore.

Algoritmi e programmi

- Le soluzioni effettive per esecutori caratterizzati formalmente sono chiamate algoritmi
- Quando l'esecutore è un calcolatore, gli algoritmi vengono detti programmi
- Il linguaggio formale per la loro descrizione è detto linguaggio di programmazione

Sviluppo di un programma

Il processo di sviluppo di un programma è organizzato in:

- analisi del problema e identificazione di una soluzione
- formalizzazione della soluzione e definizione dell'algoritmo risolutivo
- programmazione, cioè scrittura dell'algoritmo, in un linguaggio di programmazione di "alto livello"
- traduzione del programma in un "linguaggio macchina" direttamente interpretabile dalla macchina

Linguaggi di alto livello e linguaggi macchina

- I linguaggi di alto livello sono più facilmente comprensibili dagli esseri umani ma sono sempre linguaggi formali
- Il linguaggio macchina è un linguaggio formale comprensibile direttamente da uno specifico calcolatore
- La traduzione da quello di alto livello a quello macchina può essere fatta automaticamente in virtù delle proprietà formali di entrambi

Algoritmo

- Un algoritmo è un insieme finito ed ordinato di passi che determinano un procedimento atto a risolvere in un tempo finito un problema utilizzando i dati iniziali ed ottenendo dei risultati.
- *Esempi*

1. *Algoritmi per eseguire le 4 operazioni che ci sono stati insegnati alle elementari*
   - *Espressi in un linguaggio adatto ai bambini*
2. *Ricette di cucina*
   - *Espresse nel linguaggio dei libri di cucina*

Algoritmo: esempio

Algoritmo per accedere al proprio account sul computer del laboratorio:

1. Accendere lo schermo se è spento.
2. Scrivere il proprio <username> nella riga in cui compare la scritta login.
3. Scrivere la propria <password> nella riga in cui compare la scritta password.
4. Se il sistema risponde con la frase: <<utente non abilitato>> ritornare al punto 2. e riprovare.
5. Se il sistema continua a rispondere con la frase: <<utente non abilitato>> allora chiamare il tutor.

Determinazione del maggiore di due numeri interi

- Occorre definire quali problemi sono elementari,cioè quali problemi hanno una soluzione che può essere eseguita direttamente senza dover ricorrere ad altre scomposizioni
- Supponiamo che la differenza tra due interi e la valutazione del segno positivo o negativo di un numero siano problemi elementari

Determinazione del maggiore di due numeri interi

a. Leggi un valore dall'esterno e inseriscilo nella variabile x
b. Leggi un valore dall'esterno e inseriscilo nella variabile y
c. Calcola la differenza d ← x-y
d. Se d ha segno positivo vai al passo e. altrimenti al passo f.
e. Stampa "il massimo è" e il valore di x e vai al passo g.
f. Stampa "il massimo è" e il valore di x e vai al passo g.
g. Termina

Scomposizione in sottoproblemi

- Per problemi più complessi il numero dei passi cresce notevolmente
- Per semplificare la scrittura di un algoritmo lo si può scrivere in funzione di sottoproblemi non elementari purché di essi sia nota la scomposizione in problemi elementari
- Tali problemi dalla soluzione nota sono detti problemi terminali

Scomposizione in sottoproblemi

- L'esecuzione di un algoritmo può essere pensata in termini di soluzione per un insieme di problemi terminali
- In un linguaggio di programmazione:
  - alla soluzione dei problemi terminali elementari corrisponde il concetto di istruzione
  - alla soluzione dei problemi terminali non elementari corrisponde il concetto di sottoprogramma (procedura o funzione)

Determinazione del maggiore tra tre numeri interi

- Possiamo considerare terminale l'algoritmo per la soluzione del problema del massimo tra due interi
- Il presente problema può dunque essere scomposto come segue:
  a. Se x è maggiore di y allora esegui b. altrimenti esegui c.
  b. La soluzione è il maggiore tra x e z
  c. La soluzione è il maggiore tra y e z

Determinazione del maggiore tra n numeri interi

- Si può generalizzare il procedimento ottenendo:
  a. Trova il maggiore tra i primi due numeri
  b. Trova il maggiore tra i il terzo e il risultato del passo precedente
  c. Trova il maggiore tra i il quarto e il risultato del passo precedente
  d. …

Determinazione del maggiore tra n numeri interi

- Più elegantemente:
  a. Trova il maggiore tra i primi due numeri
  b. Finché ci sono numeri esegui c. altrimenti esegui d.
  c. Trova il maggiore tra il nuovo numero e quello trovato al passo precedente
  d. La soluzione è l'ultimo numero trovato al passo c.

Nota

- Il passo b. mostra una struttura usata spesso nella descrizione dei problemi ripetitivi: "finché *condizione* ripeti *azione*"
- Tale struttura indica che l'azione deve essere eseguita ripetutamente valutando ogni volta la condizione
- In questo modo si ottiene una formulazione molto concisa e indipendente da ogni specifico valore di n
- Un problema che ammette una soluzione di questo tipo si dice che ha una soluzione di tipo iterativo
- I linguaggi di programmazione hanno modi molto compatti (specifici comandi) per esprimere diverse strutture iterative per controllare il flusso dell'elaborazione

Proprietà degli algoritmi

- Perché una descrizione formale di una soluzione sia un algoritmo devono essere soddisfatti i seguenti requisiti:
  - Finitezza
  - Generalità
  - Non ambiguità

Proprietà degli algoritmi

- Finitezza
  - Il numero delle istruzioni è finito
  - Ogni soluzione è eseguita in un intervallo finito di tempo
  - Ogni istruzione è eseguita un numero finito di volte

Proprietà degli algoritmi

Generalità

- Un algoritmo fornisce la soluzione ad una classe di problemi
  - dato un insieme di definizione o *dominio*
  - dato un insieme di arrivo o *codominio*
  - l'algoritmo può operare su tutti i dati appartenenti al dominio per fornire una soluzione all'interno del codominio

Proprietà degli algoritmi

Non ambiguità

- Le istruzioni sono definite in modo univoco
- Non ci sono paradossi, contraddizioni, ambiguità
- A parità di dati su cui lavorare, il risultato dell'algoritmo è identico indipendentemente da chi lo sta eseguendo

Descrizione degli algoritmi

- Le proposizioni usate da un linguaggio formale descrivono due entità:
  - Le operazioni che devono essere eseguite (*istruzioni*)
  - Gli oggetti (*dati*) sui quali si devono eseguire le operazioni

Algoritmi e programmi: sviluppo

- Introdurremo la nozione di
  - contenitore di dati (variabile)
  - come astrazione della nozione di zona della memoria utilizzata da un computer per i dati
- Descriveremo gli algoritmi mediante:
  - diagrammi di flusso
  - un linguaggio di programmazione (***VBA***)

Algoritmi e programmi: sviluppo

Per costruire un programma conviene procedere con metodo:

- passando da un'analisi del problema da risolvere,
- all'algoritmo della soluzione rappresentato in un "linguaggio" adatto all'uomo ma non troppo lontano dai linguaggi di programmazione
- ed infine al programma scritto nel linguaggio di programmazione prescelto

Processo di sviluppo

A. Diamo un nome al problema e partiamo dall'analisi del problema.

B. Scriviamo la specifica funzionale.

C. Outline dell'algoritmo.
   1. Si introducono i contenitori di dati necessari e le relative operazioni elementari.
   2. Si disegna un diagramma di flusso che indica in modo preciso e non ambiguo la successione di operazioni da eseguire.

D. Traduciamo il diagramma di flusso in un programma.

A. Problema e analisi del problema

- L'analisi del problema è il primo passo e deve fornire:
  - un nome e una breve descrizione di cosa si vuol fare
  - un elenco di requisiti cioè di richieste che il programma deve soddisfare

Esempio di analisi del problema

- Problema: RADICI
- Descrizione: vogliamo trovare le soluzioni reali di un'equazione di secondo grado.
- Requisiti: l'equazione può non avere soluzioni, avere due soluzioni coincidenti o due soluzioni distinte; a seconda dei casi, si vuole il messaggio:
  - "nessuna radice x"
  - "radici coincidenti = *r*" dove *r* è il valore reale delle radici
  - "due radici distinte r_1, r_2", dove r_1 e r_2 sono i valori reali delle due radici

B. Specifica funzionale

- La specifica funzionale indica:
  - quali sono i dati iniziali (dati in input), cioè quelli da elaborare, detti anche *ingressi* all'algoritmo
  - qual è il risultato atteso (dati in output), in funzione degli ingressi, detto anche *uscita* dell'algoritmo

Esempio di specifica funzionale

- RADICI: specifica funzionale
- Argomenti o ingressi:
  - *a*,*b*,*c*: numeri reali, coefficienti dell'equazione da elaborare
- Risultati o uscite:
  - "nessuna radice"
  - "$x_1=x_2=r$" se l'equazione ax^2+bx+c ha radici coincidenti uguali a r
  - "$x_1=r_1$, $x_2=r_2$" se l'equazione ax^2+bx+c ha radici distinte uguali a r_1, r_2

C. Outline dell'algoritmo

- Descrive brevemente l'idea dell'algoritmo, cioè i passi da eseguire per giungere alla soluzione a partire dagli ingressi.
- Il primo outline non deve necessariamente essere molto dettagliato: si procede per raffinamenti successivi.

Esempio di outline dell'algoritmo

- RADICI: outline dell'algoritmo
  - Risolvo il problema calcolando il discriminante delta dell'equazione
  - Analizzo i vari casi del delta:
    - < 0
    - = 0
    - > 0
  - Caso per caso costruisco il messaggio da inviare in uscita
- Successivamente definisco le variabili coinvolte e dettaglio l'algoritmo grazie ad un diagramma di flusso

Linguaggi di programmazione: Linguaggio Macchina

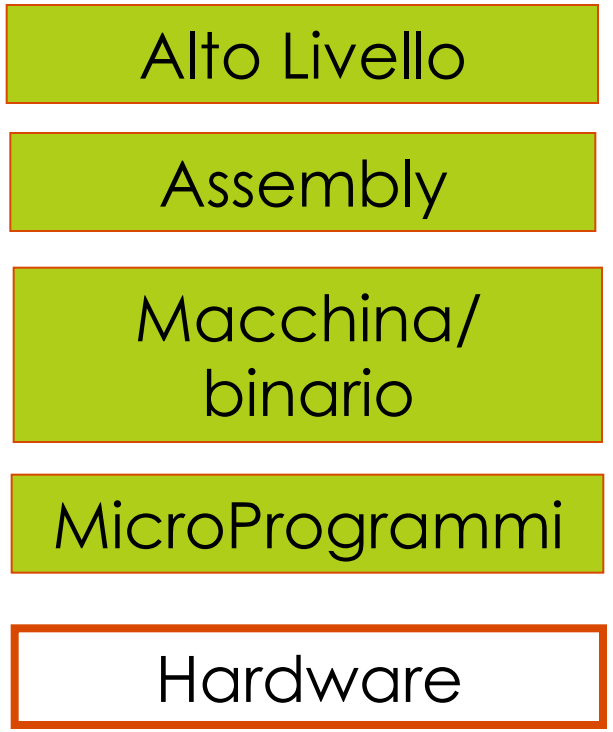
- **I primi linguaggi di programmazione coincidevano con l'insieme delle istruzioni eseguibili dall'hardware.**
- **Le istruzioni hardware sono codificate in codice binario: ogni informazione è rappresentata, all'interno della macchina, come una sequenza di bit.**
- **LOAD 8 potrà essere rappresentata in una macchina reale come 00110010, dove 0011 è la rappresentazione interna del codice operativo LOAD**
- **Enorme sforzo richiesto per codificare algoritmi semplici.**

Linguaggi di programmazione: Assembly

- **Dalla nascita dei primi calcolatori, si è cercato di diminuire lo sforzo e aumentare la produttività dell'utente, anche a costo di caricare la macchina di maggiori compiti.**
- **E' meglio risparmiare il tempo dell'uomo anche a costo di sprecare tempo-macchina (traduzione di programmi e scarsa efficienza del programma).**
- **La prima evoluzione dei linguaggi di programmazione ha portato ad una codifica di tipo simbolico, anziché binaria, dei programmi.**
- **Tali versioni simboliche dei linguaggi hardware sono note come linguaggi assembly**

Linguaggi di programmazione: Evoluzione

- **Il passo successivo nell'evoluzione dei linguaggi di programmazione tese a rendere la codifica degli algoritmi il più possibile "vicina" al problema da risolvere anziché all'architettura della macchina destinata all'esecuzione del programma.**
- **Non a caso i primi due linguaggi di alto livello, FORTRAN e COBOL, hanno costrutti fortemente ispirati alla notazione usata per descrivere i problemi di maggior rilievo per il calcolo automatico degli anni '50, cioè l'elaborazione numerica e quella gestionale.**



Linguaggi di programmazione: Evoluzione

- **Gli anni '70 vedono il fiorire dei linguaggi <u>Strutturati</u>:**
  - **il SIMULA 67, l'ALGOL 68, ma soprattutto il PASCAL e il C**
- **Negli anni '80 si ha l'affermarsi dei linguaggi <u>Object-Oriented</u> (OO):**
  - **C++, Visual BASIC, Java, C#**

La Tecnica di Programmazione

- **Non è legata ad un linguaggio, anche se alcuni linguaggi sono ideali per essere abbinati ad una specifica tecnica di programmazione**
- **Esistono tre tecniche:**
  a) Programmazione NON Strutturata, che produce lo Spaghetti Code
  b) Programmazione Strutturata: Sequenza, Selezione, Ripetizione
  c) Programmazione ad Oggetti: Classe, Oggetto, Metodo

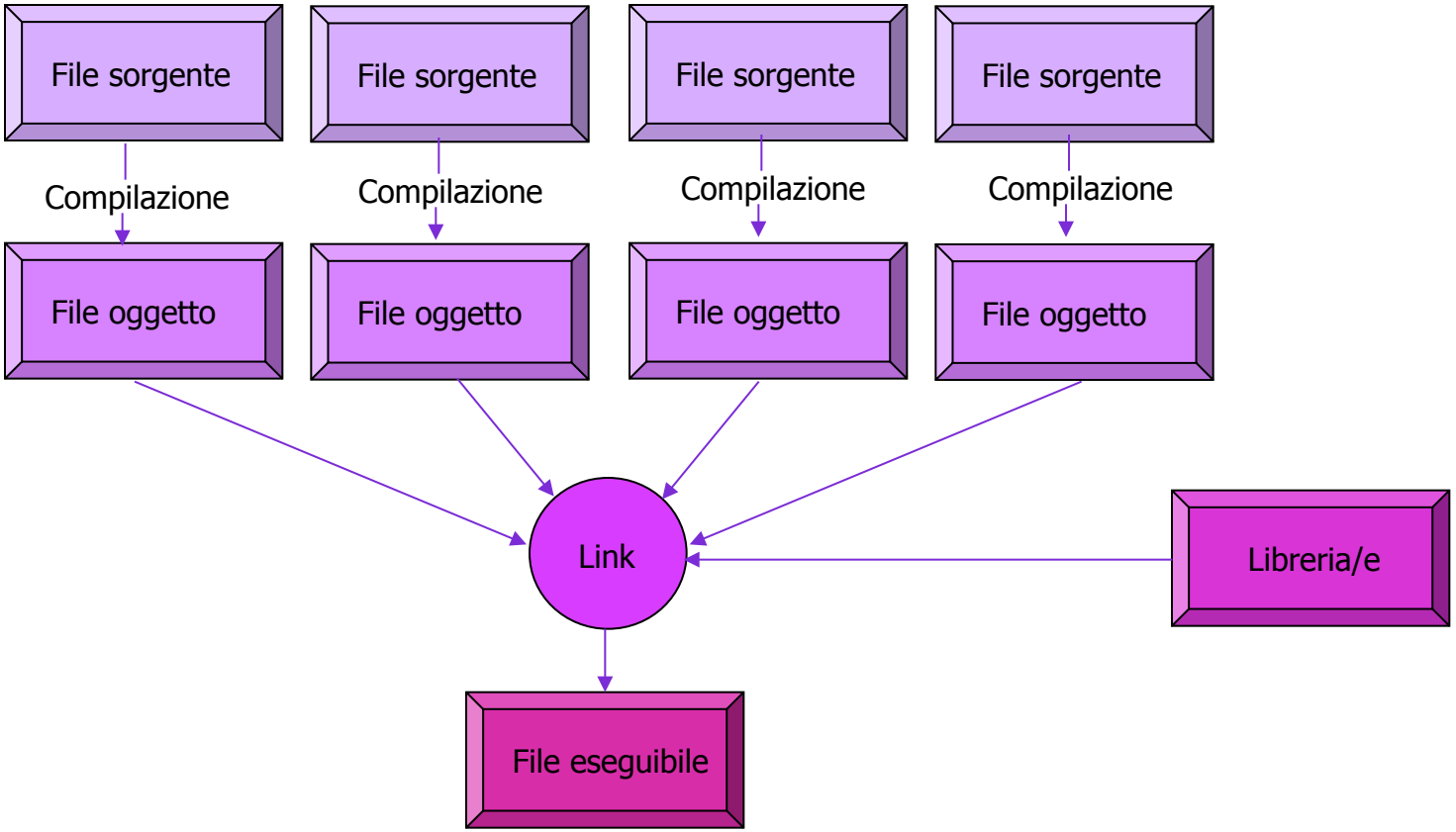
L'Esecuzione dei Programmi

- **Il programma scritto in un qualunque linguaggio deve essere tradotto in linguaggio binario**
- **Esistono due soluzioni:**
  - ✓ Interpretazione
  - ✓ Compilazione

L'Interpretazione

- Funzionamento di un interprete:
  - Preleva un'istruzione I del programma P scritto nel linguaggio L
  - Decodifica I
  - Traduce I in una serie di istruzioni in linguaggio macchina M_1, M_2, … , M_n
  - Esegue M_1, M_2, … , M_n
  - Passa all'istruzione successiva di P fino a quando non si sia raggiunta una istruzione di terminazione

Lo sviluppo dei programmi basato sulla Compilazione



- I codici sorgente ed oggetto possono essere suddivisi in più file, il codice eseguibile di un programma risiede in un unico file

Le Funzioni di Libreria

- In molti linguaggi (quali il C), molte operazioni vengono delegate a delle librerie
- Le funzioni sono divise in gruppi, quali I/O, gestione della memoria, operazioni matematiche e manipolazione di stringhe
- Per ogni gruppo di funzioni esiste un file sorgente, chiamato *file header*, contenente le informazioni necessarie per utilizzare le funzioni (Dichiarazione costanti, funzioni, parametri delle funzioni, etc.)
- I nomi dei file header terminano, per convenzione, con l'estensione ".h" (ad es., *stdio.h* è il file header dello standard I/O nel linguaggio C)

La compilazione dei file sorgente

- **Un file sorgente contiene un main e un insieme di routine, chiamate funzioni, ognuna delle quali risolve una piccola parte del problema di programmazione**
- **Un file sorgente può contenere chiamate a funzioni di libreria e chiamate a funzioni contenute in altri file sorgenti**
- **Il compilatore è esso stesso un programma (o un gruppo di programmi) che deve essere eseguito**
- **Durante la compilazione avviene:**
  - Controllo del codice sorgente da parte di un preprocessore: interpretare speciali direttive di precompilazione,
  - Controllo di eventuali errori sintattici (NON LOGICI),
  - Produzione di un file in codice Assembly,
  - Trasformazione del file in codice Assembly in un altro file in codice OGGETTO (Macchina): **estensione .obj**

Diagrammi di flusso

Il diagramma di flusso è una rappresentazione grafica di un algoritmo costituita da un insieme di simboli (figure geometriche standard) collegate da frecce.

Questo metodo è stato messo a punto negli Stati Uniti da Larry Constantine ed Edward Yourdon a metà degli anni ’80.

Esso presenta, rispetto alla descrizione verbale, una maggiore concisione, una minore possibilità di ambiguità e una comprensione più immediata.

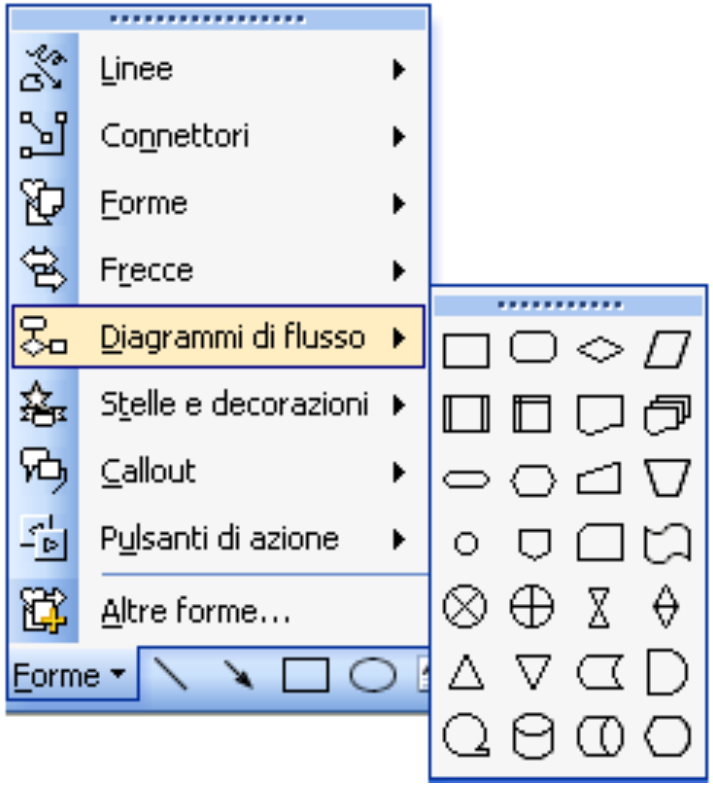
Per consentire una comprensione uniforme dei diagrammi di flusso, è bene disegnarli secondo alcune convenzioni grafiche standard.

I simboli dei diagrammi di flusso si possono ottenere in modo semplice con il programma

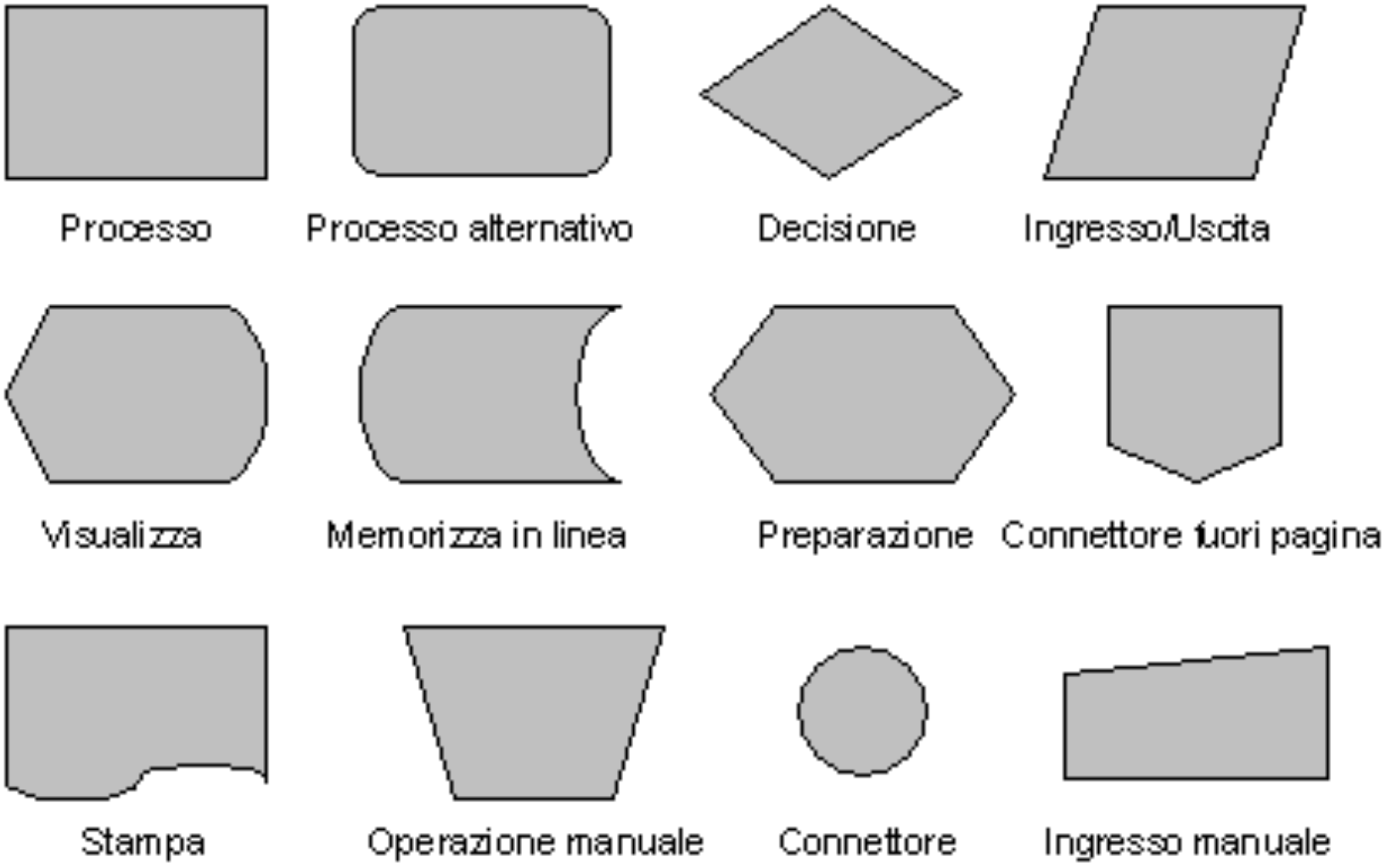
Microsoft Office Word 2003

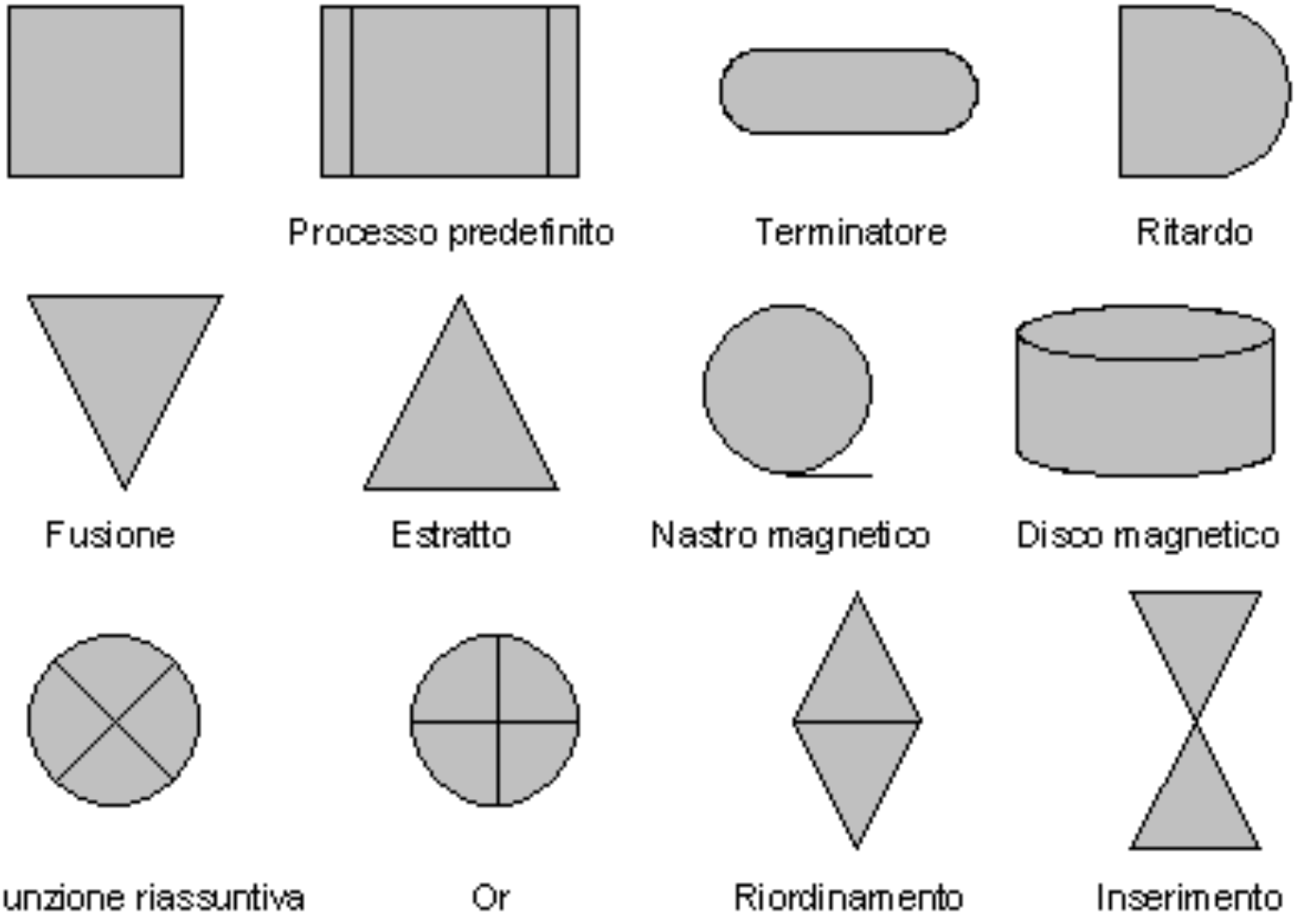
cliccando sull' icona del disegno

e quindi sul menu Forme → Diagrammi di flusso.

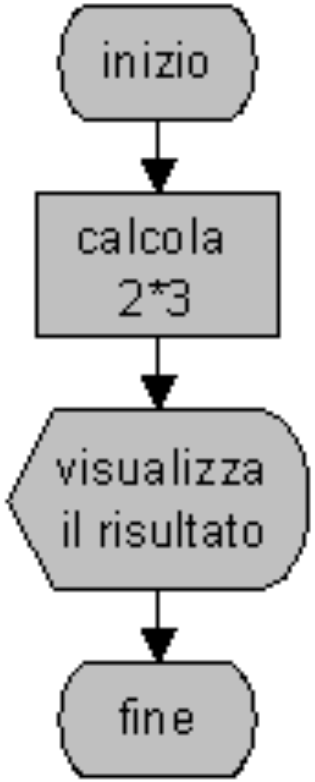


I simboli in uso sono riportati nelle due figure successive.




Processo predefinito
Terminatore
Ritardo
Fusione
Estratto
Nastro magnetico
Disco magnetico
Giunzione riassuntiva
Or
Riordinamento
Inserimento

La figura successiva riporta un semplice diagramma di flusso, che indica la sequenza di passi necessari per calcolare il prodotto **2*3** e visualizzarne il risultato.



Questo esempio usa i tre simboli:

terminatore	indica l'inizio e la fine dell'elaborazione; deve essere presente in ogni diagramma di flusso, con la fine non necessariamente come ultimo elemento
processo	rappresenta una funzione operativa, quale un calcolo matematico o un'assegnazione di valore a una variabile
visualizzazione	indica di mostrare sullo schermo il risultato di una elaborazione

Naturalmente i diagrammi di flusso sono sempre più complessi di questo, e uno solo di essi può occupare diverse pagine.

Per questa ragione va posta grande attenzione alle frecce che uniscono i blocchi del diagramma, e che indicano l'ordine in cui saranno eseguite le varie istruzioni.

Come regola generale, il flusso delle operazioni procede *dall'alto in basso* e *da sinistra a destra*; tuttavia si potrebbe talvolta dover variare questa regola, e a tale fine si utilizzano appunto le frecce nel modo opportuno.

Osserviamo anche che le frasi scritte nei blocchi di un diagramma non devono seguire alcuna regola grammaticale o sintattica, ma possono avere una forma libera, purché indichino chiaramente l' operazione da eseguire.

Nella fase successiva della programmazione, quando cioè il diagramma viene tradotto in programma, si deve invece tenere conto scrupoloso delle regole previste dal linguaggio di programmazione scelto.

Gli errori che si possono commettere in questa seconda fase sono, tutto sommato, più facili da evidenziare e da correggere di quelli in cui si può incorrere disegnando un diagramma di flusso.

Controllo del programma

L'ordine con cui le diverse operazioni devono essere eseguite è specificato da particolari costrutti linguistici, detti *strutture di controllo*.

Alla metà degli anni '60 i due studiosi italiani G. Jacopini e C. Boehm hanno dimostrato che, dato un algoritmo idoneo a risolvere un problema, ne esiste sempre uno equivalente in cui compaiano esclusivamente le tre seguenti strutture fondamentali:

sequenza
selezione (o scelta)
iterazione (o ciclo)

La **sequenza** indica una successione di operazioni che devono essere eseguite una dopo l'altra, ciascuna una sola volta; ne è un esempio il diagramma di flusso precedente.

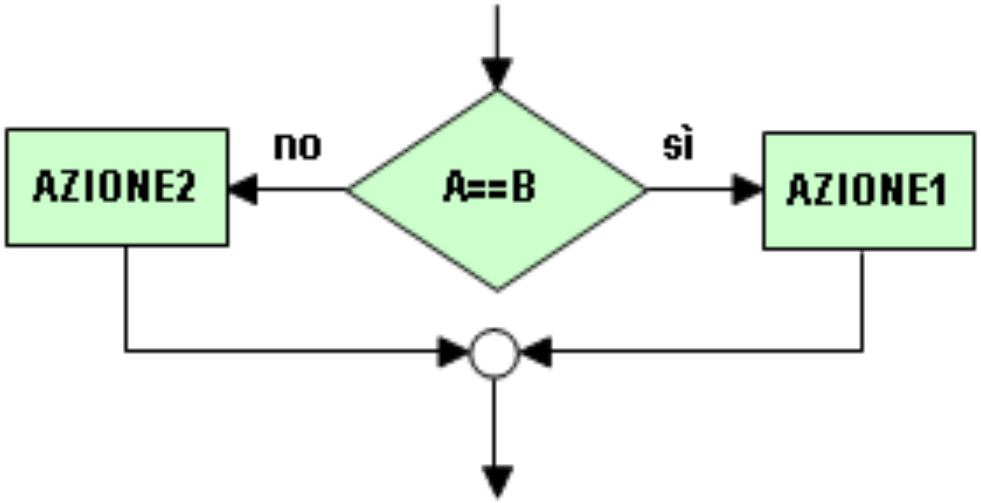
In molti linguaggi di programmazione, tra i quali il C, non esiste una forma sintattica specifica per la sequenza in quanto, in assenza di istruzioni di controllo di flusso, le istruzioni del programma vengono eseguite nell'ordine in cui sono scritte.

La **selezione** (o scelta) permette a un programma di proseguire secondo uno tra due (o più) flussi di istruzioni alternative, a seconda del risultato di un test o del verificarsi di una condizione.

L'**iterazione** (o *ciclo*, o *loop*) consiste nella ripetizione di una o più istruzioni, e si può ottenere collegando il flusso in uscita da un blocco con il flusso in entrata nel blocco stesso o in uno precedente.

In tal modo si possono eseguire compiti ripetitivi senza specificare uno per uno un gran numero di singoli passi, ma scrivendo per esempio un'istruzione del tipo: "esegui il prossimo passo 1.000 volte".

Selezione binaria. Nella selezione il test o la condizione sono tipicamente costituiti da una variabile logica, scritta dentro il simbolo di decisione, dal quale escono due frecce. Queste indicano le possibili azioni da compiersi a seconda del valore della variabile, come mostra la figura.



La selezione tra due alternative viene realizzata nel linguaggio C dal costrutto **if...else...** dove

- i puntini dopo la parola **if** indicano la condizione di cui si esamina la verità e le azioni da compiere se la condizione risulta vera,
- quelli dopo la parola **else** le azioni da compiere se risulta falsa.

Il diagramma di flusso precedente si traduce quindi nel seguente costrutto:

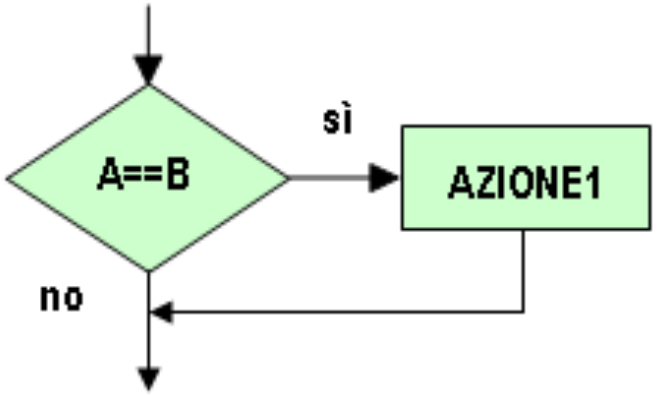
```
if (a == b)
  azione 1;
else
  azione 2;
```

Ad esempio, se vogliamo determinare il massimo di due numeri, **x**, **y** e assegnare tale valore alla variabile **maxnum** scriveremo:

```
if (x >= y)
  maxnum = x;
else
  maxnum = y;
```

Si può anche volere compiere una certa azione se il test o la condizione hanno un valore **vero**, e nessuna azione nel caso contrario.

In tale caso la clausola **else** viene omessa, e si impiega lo schema seguente, che può essere considerato una forma abbreviata del precedente, dove manca l' **azione2**.



Per esempio, se si vogliono leggere due numeri e indicare con **M** il maggiore e **m** il minore, si può usare lo schema seguente, privo della clausola **else**

leggi M, m;
if (M < m)
scambia M con m;

Osserviamo che l'istruzione "**scambia M con m**;" non è una istruzione elementare (dal punto di vista del computer), e quindi potrebbe non essere disponibile nel linguaggio di programmazione usato.

Essa può comunque essere realizzata sostituendola con le seguenti tre istruzioni elementari:

t = M;
M = m;
m = t;

Esempio: Massimo Comun Divisore (1). Un esempio di algoritmo contenente due selezioni binarie con una sola azione è fornito da quello che calcola il Massimo Comun Divisore (MCD) di due interi positivi secondo il seguente metodo di Euclide**(*)**

Per calcolare il MCD di due interi positivi:

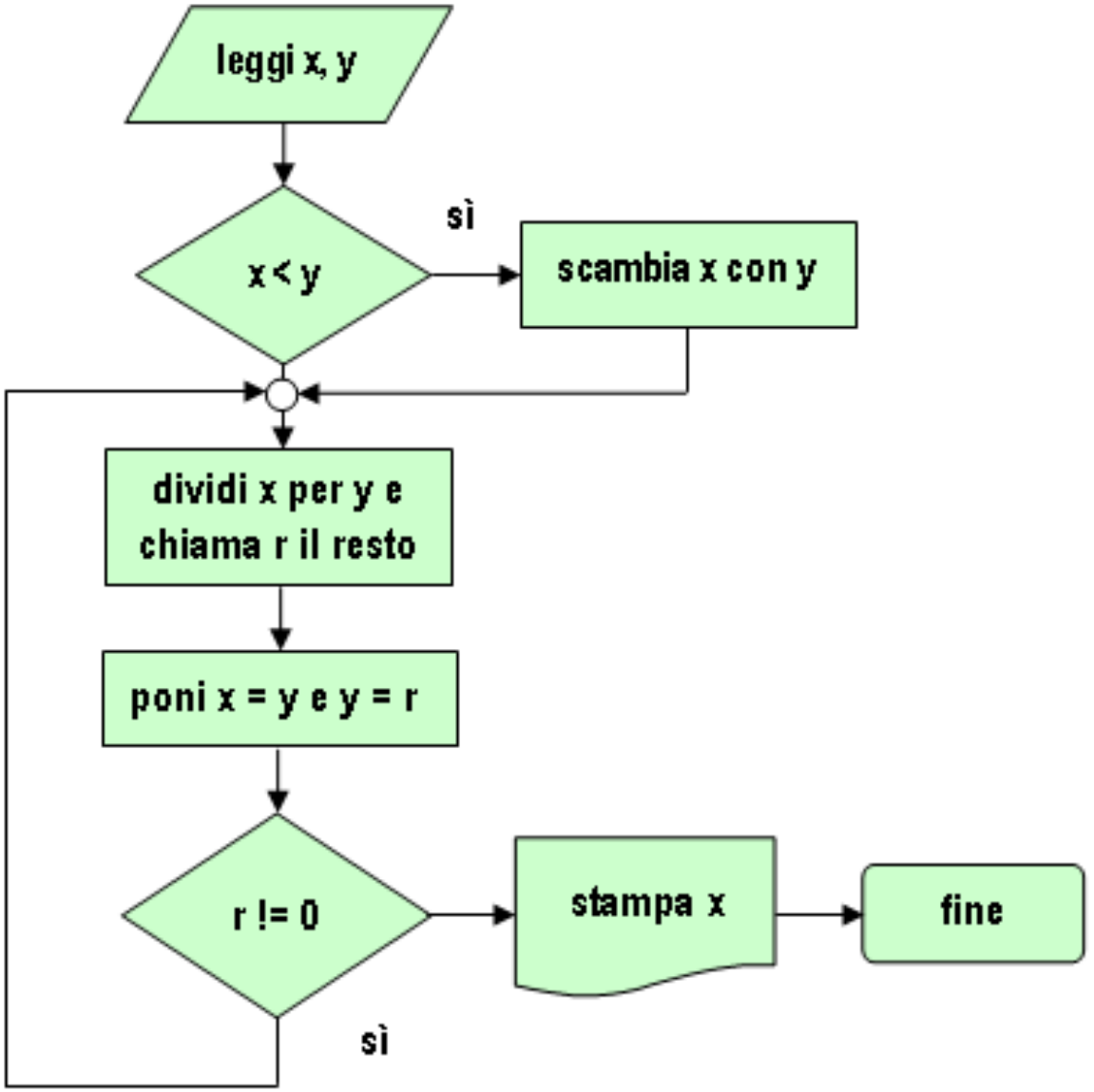
- *si divide il maggiore per il minore*
- *se il resto è 0 il minore rappresenta il MCD cercato*
- *altrimenti il minore sostituisce il maggiore, il resto sostituisce il minore e il procedimento ricomincia da capo.*

(*) Questo algoritmo è enunciato, in termini equivalenti a quelli sopra riportati, nelle proposizioni 1 e 2 del libro VII degli "Elementi" di Euclide.

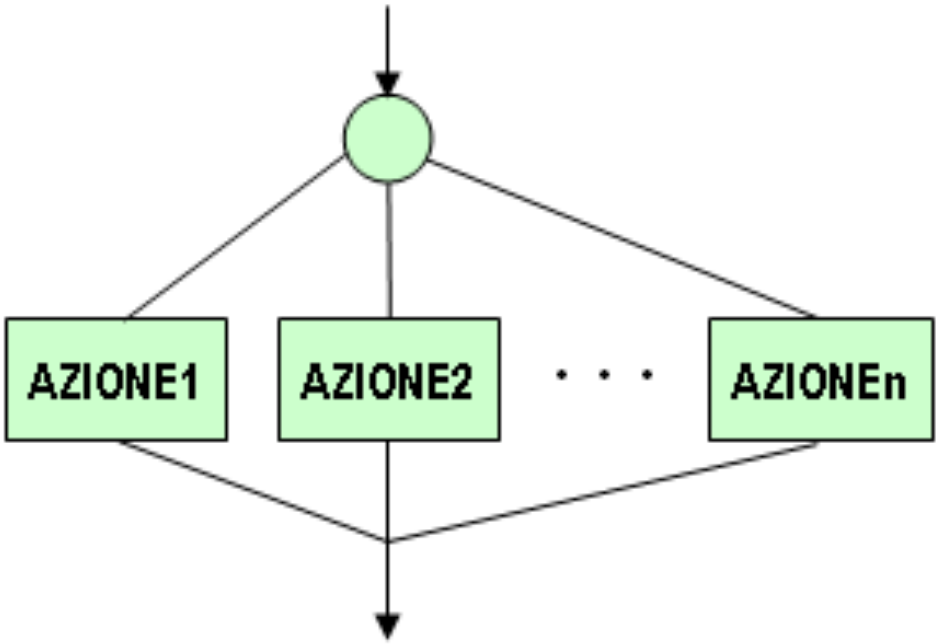
La corrispondente espressione in pseudo istruzioni è la seguente:

1. **leggi x, y;**
2. **if (x < y) scambiali;**
3. **dividi x per y e chiama r il resto;**
4. **poni x = y;**
5. **poni y = r;**
6. **if (r == 0) stampa x ; fine**
7. **vai a 3);**

Il relativo diagramma di flusso è indicato in figura.



Selezione multipla. Non sempre la scelta prevede due sole alternative; se queste sono più di due è necessaria una *selezione multipla* (o *n-aria*), secondo il seguente schema:

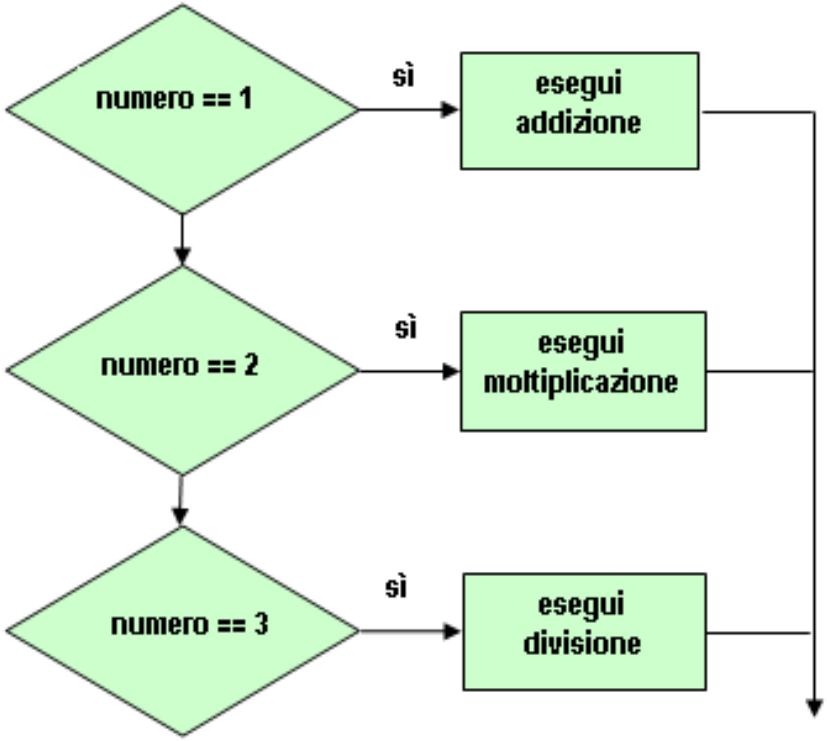


Osserviamo che anche tale selezione multipla si potrebbe eseguire tramite la selezione binaria, disponendo in cascata più selezioni binarie.

Se, ad esempio, si vuole eseguire

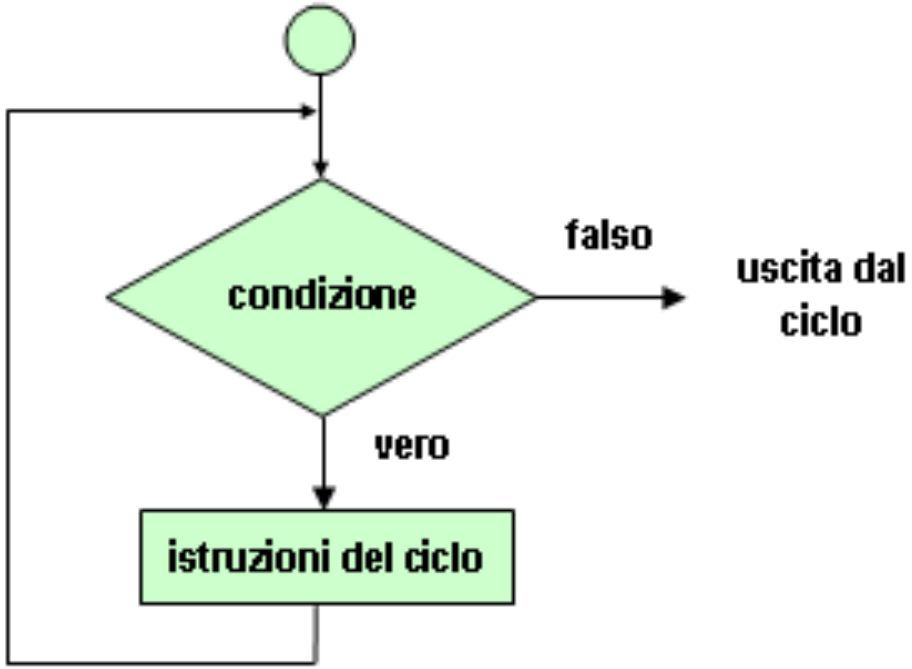
- una addizione tra due numeri se è stato digitato **1**;
- una moltiplicazione se è stato digitato **2**;
- una divisione se è stato digitato **3**,

si potrebbero disporre in cascata tre selezioni binarie come indicato in figura

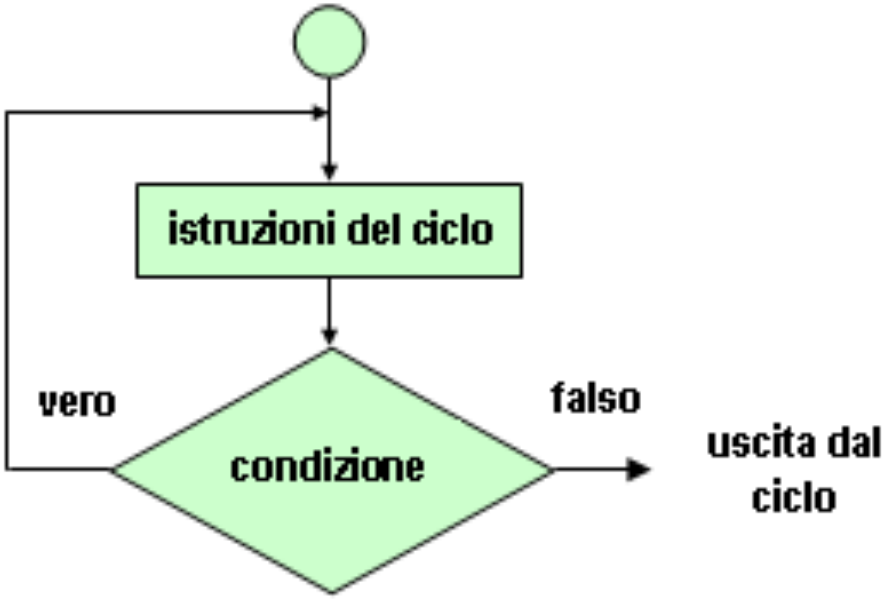


Iterazione. Affinché l'iterazione sia costituita da un numero finito di passi (caratteristica n. 3 degli algoritmi), è necessario che nella linea di collegamento sia inserito un simbolo di decisione che contenga la condizione di uscita dal ciclo.

Tale blocco di controllo si dice *guardia*, e si può trovare *prima* del gruppo di istruzioni che costituiscono il ciclo



o *dopo* di esso.



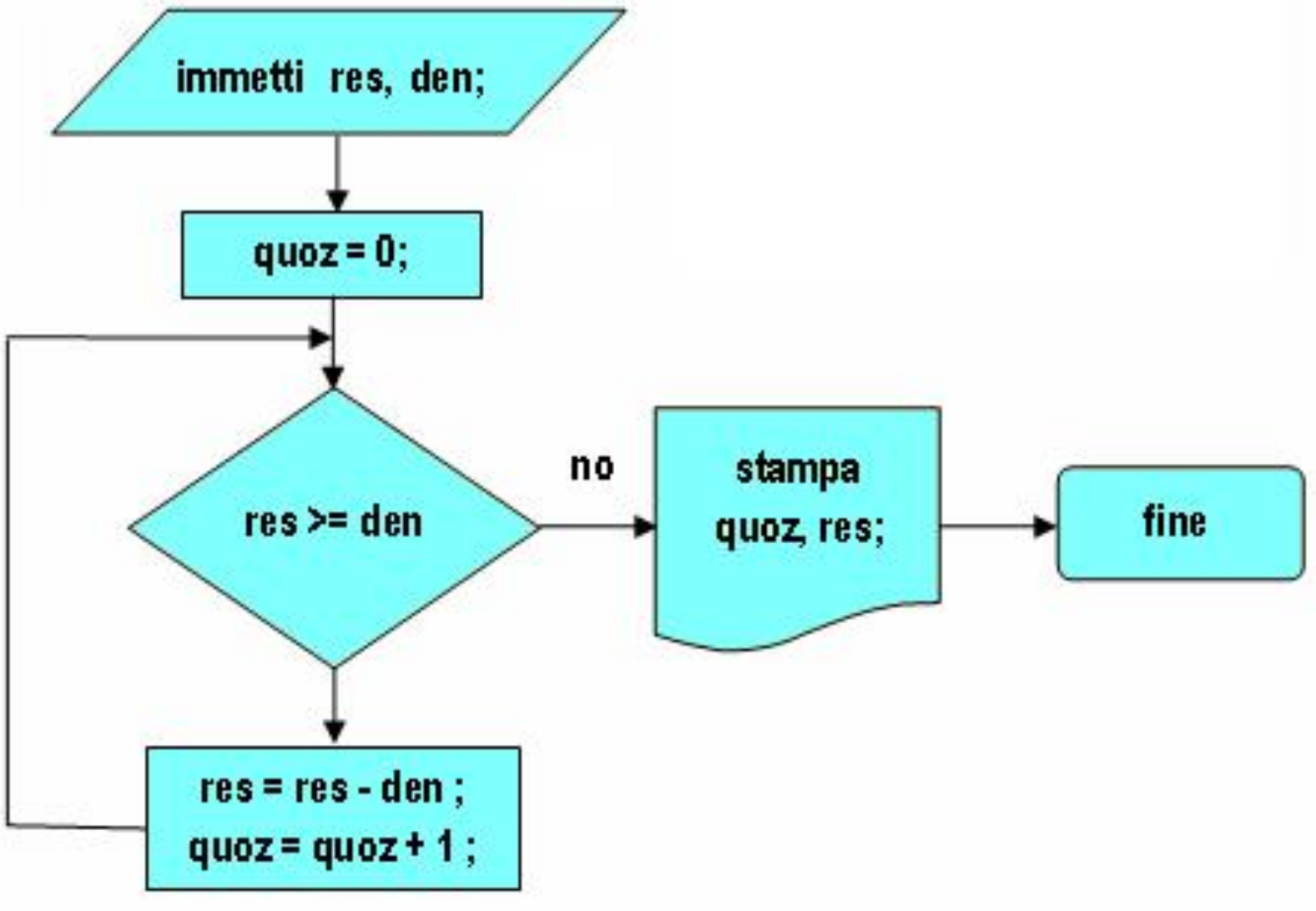
Esempio: divisione con sottrazioni ripetute. Un esempio di ciclo con guardia all'inizio è costituito dall'algoritmo che esegue la divisione intera fra due numeri naturali con il seguente metodo delle *sottrazioni ripetute*:

Per dividere tra loro due numeri naturali:

- *si sottrae il divisore dal dividendo;*
- *si continua a sottrarlo dal risultato dell'ultima sottrazione fino a che tale risultato sia maggiore o uguale al divisore*
- *il numero di sottrazioni effettuate fornisce il quoziente*
- *il risultato dell'ultima fornisce il resto della divisione*

Questo algoritmo si traduce facilmente nel diagramma di flusso seguente, dove la variabile **res** contiene inizialmente il dividendo, la variabile **den** il divisore. **res** contiene poi i risultati delle successive sottrazioni, l'ultimo dei quali fornisce appunto il resto.

Il quoziente è fornito dal contatore **quoz**, che inizialmente vale **0** e viene incrementato di **1** a ogni sottrazione eseguita.

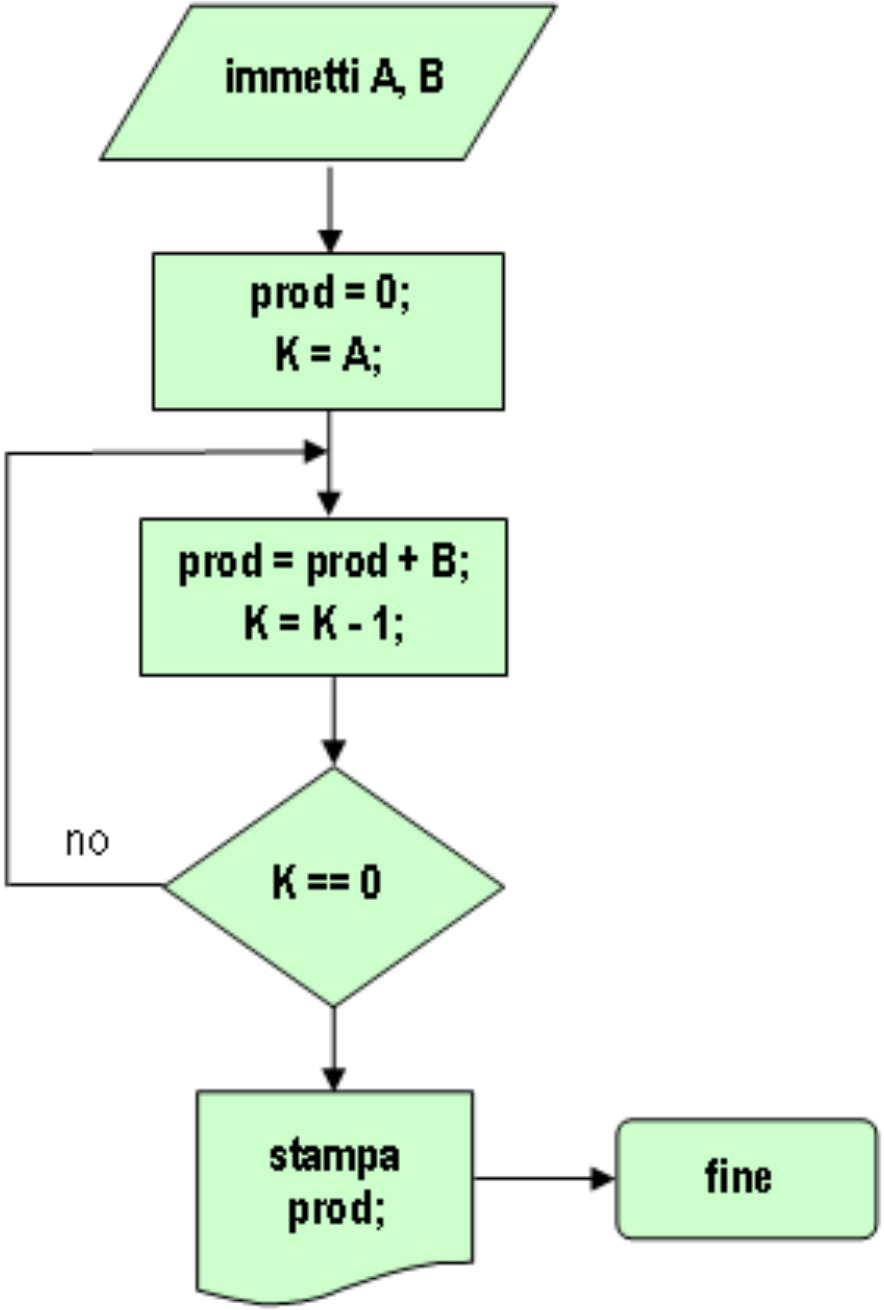

immetti res, den;
quoz = 0;
res >= den
no
stampa
quoz, res;
fine
res = res - den ;
quoz = quoz + 1 ;

Esempio: moltiplicazione con addizioni ripetute. Un esempio di ciclo con guardia alla fine è costituito dall'algoritmo che esegue la moltiplicazione di due numeri naturali con il metodo delle *addizioni ripetute*:

per moltiplicare tra loro due numeri naturali si somma uno di essi a se stesso un numero di volte uguale all'altro.

Questo algoritmo si traduce facilmente nel seguente diagramma di flusso dove indichiamo con

- **A**, **B** i due numeri da moltiplicare
- **prod** il prodotto
- **K** una variabile contatore, posta inizialmente uguale a uno dei due numeri. **K** viene diminuita di **1** ogni volta che si esegue un'addizione, e quando raggiunge il valore **0** determina la fine del ciclo.


immetti A, B
prod = 0;
K = A;
prod = prod + B;
K = K - 1;
no
K == 0
stampa
prod;
fine

Riferimenti

- I materiali di questa lezione sono stati elaborati dalle seguenti sorgenti:
- docenti.lett.unisi.it/files/77/2/3/1/ Teoria_della_Programmazione_1.ppt (di Paolo Bertetti)
- www.diit.unict.it/users/scava/dispense/.../ FondamentiProgrammazione.pp
- www.uniroma2.it/didattica/fondaminf/deposito/Fonda5.ppt