Cosa intendiamo per programmazione

- I calcolatori sono macchine in grado di eseguire velocemente e con precisione sequenze di operazioni elementari.
- A differenza di altre macchine automatiche (es. lavatrici, calcolatrici tascabili, registratori di cassa, ecc.) i calcolatori sono programmabili: la funzione svolta di volta in volta dipende dal particolare programma con cui l'utente istruisce la macchina per la soluzione di un problema.
- Un programma è una sequenza di operazioni necessarie per risolvere un problema, espressa in un linguaggio di programmazione che il calcolatore è in grado di comprendere ed eseguire.

Cosa intendiamo per programmazione

- Il procedimento che porta alla definizione dei programmi adatti a risolvere problemi è detto programmazione.
- I concetti che stanno alla base della programmazione si possono spiegare e comprendere senza far riferimento al calcolatore.
- La programmazione è tuttavia divenuta una vera e propria disciplina solo con l'avvento dei moderni calcolatori elettronici.

Le fasi della programmazione

Ad un primo livello di astrazione l'attività della programmazione può essere suddivisa in quattro (macro) fasi principali.

1. Definizione del problema (specifica)
2. Individuazione di un procedimento risolutivo (algoritmo)
3. Codifica dell'algoritmo in un linguaggio di programmazione (codifica)
4. Esecuzione e messa a punto (esecuzione)

Specifica

- La prima fase della programmazione consiste nel comprendere e definire (specificare) il problema che si vuole risolvere.
- La specifica del problema può essere fatta in maniera più o meno rigorosa, a seconda del formalismo descrittivo utilizzato.
- La specifica di un problema prevede la descrizione dello stato iniziale del problema (dati iniziali, input) e dello stato finale atteso (i risultati, output).
- La caratterizzazione degli stati iniziale e finale dipende dal particolare problema in esame e dagli oggetti di interesse.

Esempi di specifica informale

1. Dati due numeri, trovare il maggiore.
2. Dato un elenco telefonico e un nome, trovare il numero di telefono corrispondente.
3. Data la struttura di una rete stradale e le informazioni sui flussi dei veicoli, determinare il percorso più veloce da A a B.
4. Scrivere tutti i numeri pari che non sono la somma di due numeri primi (Congettura di Goldbach).
5. Decidere per ogni programma e per ogni dato in ingresso, se il programma C termina quando viene eseguito su quel dato.

Esempi (contd.)

Caratteristiche comuni ai problemi

informazioni in ingresso $\Longrightarrow$ informazioni in uscita

stato iniziale $\Longrightarrow$ stato finale

Osservazioni sulla formulazione dei problemi:

- la descrizione non fornisce un metodo risolutivo (es. 3 ▸)
- la descrizione del problema è talvolta ambigua o imprecisa (es. 2, con Mario Rossi che compare più volte ▸)
- per alcuni problemi non è noto un metodo risolutivo (es. 4 ▸)
- esistono problemi per i quali è stato dimostrato che non può esistere un metodo risolutivo (es. 5 ▸)

Noi considereremo solo problemi per i quali è noto che esiste un metodo risolutivo.

Algoritmi

- Una volta specificato il problema, si determina un procedimento risolutivo dello stesso (algoritmo), ovvero un insieme di azioni da intraprendere per ottenere i risultati attesi.
- Il concetto di algoritmo ha origini molto lontane: l'uomo ha utilizzato spesso algoritmi per risolvere problemi di varia natura. Solo in era moderna, tuttavia, ci si è posti il problema di caratterizzare problemi e classi di problemi per i quali è possibile individuare una soluzione algoritmica e solo nel secolo scorso è stato dimostrato che esistono problemi per i quali non è possibile individuare una soluzione algoritmica.

Algoritmi (cont.)

- Utilizziamo algoritmi nella vita quotidiana tutte le volte che, ad esempio, seguiamo le istruzioni per il montaggio di una apparecchiatura, per la sostituzione della cartuccia di una stampante, per impostare il ciclo di lavaggio di una lavastoviglie, per prelevare contante da uno sportello Bancomat, ecc.

Un algoritmo è una sequenza di passi che, se intrapresa da un esecutore, permette di ottenere i risultati attesi a partire dai dati forniti.

Proprietà di un algoritmo

La descrizione di un procedimento risolutivo può considerarsi un algoritmo se rispetta alcuni requisiti essenziali, tra i quali:

Finitezza: un algoritmo deve essere composto da una sequenza finita di passi elementari.

Eseguibilità: il potenziale esecutore deve essere in grado di eseguire ogni singola azione in tempo finito con le risorse a disposizione

Non-ambiguità: l'esecutore deve poter interpretare in modo univoco ogni singola azione.

Tipici procedimenti che non rispettano alcuni dei requisiti precedenti sono:

- le ricette di cucina: *aggiungere sale q.b.* - non rispetta 3)
- le istruzioni per la compilazione della dichiarazione dei redditi (!)

Codifica

- Questa fase consiste nell'individuare una rappresentazione degli oggetti di interesse del problema ed una descrizione dell'algoritmo in un opportuno linguaggio noto all'esecutore.
- Nel caso in cui si intenda far uso di un elaboratore per l'esecuzione dell'algoritmo, quest'ultimo deve essere tradotto (codificato) in un opportuno linguaggio di programmazione. Il risultato in questo caso è un programma eseguibile al calcolatore.
- Quanto più il linguaggio di descrizione dell'algoritmo è vicino al linguaggio di programmazione scelto, tanto più semplice è la fase di traduzione e codifica. Se addirittura il linguaggio di descrizione coincide con il linguaggio di programmazione, la fase di traduzione è superflua.

Codifica (cont.)

I linguaggi di programmazione forniscono strumenti linguistici per rappresentare gli algoritmi sottoforma di programmi che possano essere compresi da un calcolatore.

In particolare dobbiamo rappresentare nel linguaggio di programmazione

- l'algoritmo $\Longrightarrow$ programma
- le informazioni iniziali $\Longrightarrow$ dati in ingresso
- le informazioni utilizzate dall'algoritmo $\Longrightarrow$ dati ausiliari
- le informazioni finali $\Longrightarrow$ dati in uscita

In questo corso impareremo a codificare algoritmi utilizzando il linguaggio di programmazione denominato C.

Esecuzione

- La fase conclusiva consiste nell'esecuzione vera e propria del programma.
- Spesso questa fase porta alla luce errori che possono coinvolgere ciascuna delle fasi precedenti, innescando un procedimento di messa a punto tale da richiedere la revisione di una o piú fasi (dalla specifica, alla definizione dell'algoritmo, alla codifica di quest'ultimo).
- Nel caso dei linguaggi di programmazione moderni, vengono forniti strumenti (denominati di solito debugger) che aiutano nella individuazione degli errori e nella messa a punto dei programmi.

Esempi

Negli esempi che seguono, utilizziamo un linguaggio pseudo-naturale per la descrizione degli algoritmi.

Tale linguaggio utilizza, tra l'altro, le comuni rappresentazioni simboliche dei numeri e delle operazioni aritmetiche.

Problema 1: Calcolo del prodotto di due interi positivi

Specifica

Input: due valori interi positivi A e B
Output: il valore di A×B

Algoritmo

Se l'esecutore che scegliamo è in grado di effettuare tutte le operazioni di base sui numeri, un semplice algoritmo è il seguente:

Passo 1.	Acquisisci il primo valore, sia esso A
Passo 2.	Acquisisci il secondo valore, sia esso B
Passo 3.	Ottieni il risultato dell'operazione A×B

Problema 1 (cont.)

Codifica

Un esecutore che rispetta le ipotesi precedenti è una persona dotata di una calcolatrice tascabile. La codifica dell'algoritmo in questo caso può essere allora la seguente:

Passo 1.	Digita in sequenza le cifre decimali del primo valore
Passo 2.	Digita il tasto $*$
Passo 3.	Digita in sequenza le cifre decimali del secondo valore
Passo 4.	Digita il tasto $=$

Esecuzione

Problema 1 (cont.)

- Supponiamo ora che l'esecutore scelto sia in grado di effettuare solo le operazioni elementari di somma, sottrazione, confronto tra numeri.
- È necessario individuare un nuovo procedimento risolutivo che tenga conto delle limitate capacità dell'esecutore

Algoritmo 2

Passo 1.	Acquisisci il primo valore, sia esso A
Passo 2.	Acquisisci il secondo valore, sia esso B
Passo 3.	Associa 0 ad un terzo valore, sia esso C
Passo 4.	**Finché** B>0 **ripeti**
Passo 4.1.	Somma a C il valore A
Passo 4.2.	Sottrai a B il valore 1
Passo 5.	Il risultato è il valore C

Problema 1: (cont.)

Un esecutore che rispetta le ipotesi precedenti è un bimbo in grado di effettuare le operazioni richieste (somma, sottrazione e confronto) e di riportare i risultati di semplici calcoli su un quaderno.

Codifica

Passo 1.	Scrivi il primo numero nel riquadro A
Passo 2.	Scrivi il secondo numero nel riquadro B
Passo 3.	Scrivi il valore 0 nel riquadro C
Passo 4.	Ripeti i seguenti passi finché il valore nel riquadro B è maggiore di 0: - calcola la somma tra il valore in A e il valore in C - scrivi il risultato ottenuto in C - calcola la differenza tra il valore in B ed il numero 1 - scrivi il risultato ottenuto in B
Passo 5.	Il risultato è quanto contenuto nel riquadro C.

Il concetto di stato

Gli esempi visti finora consentono di fare le seguenti considerazioni di carattere generale:

- La specifica (astratta) di un problema consiste nella descrizione di uno stato iniziale (che descrive i dati del problema) e di uno stato finale (che descrive i risultati attesi).
- È necessario individuare una rappresentazione degli oggetti coinvolti (e dunque dello stato) che sia direttamente manipolabile dall'esecutore prescelto.
- Un algoritmo è una sequenza di passi elementari che, se intrapresi da un esecutore, comportano ripetute modifiche dello stato fino al raggiungimento dello stato finale desiderato.
- Le azioni di base che l'esecutore è in grado di effettuare devono dunque prevedere, tra le altre, azioni che hanno come effetto cambiamenti dello stato.

Un'astrazione dello stato

Introduciamo una possibile astrazione del concetto di stato, di cui faremo spesso uso in seguito.

Stato

Uno stato è un insieme di associazioni tra nomi simbolici e valori.

In uno stato, ad ogni nome simbolico è associato al più un valore.

Rappresentiamo una associazione tra il nome simbolico x ed il valore val con la notazione

$$x \rightsquigarrow val$$

Lo stato: esempi

Stati corretti

- {nome $\rightsquigarrow$ Antonio, cognome $\rightsquigarrow$ Rossi, eta' $\rightsquigarrow$ 25}
- {importo $\rightsquigarrow$ \$1650, tasso $\rightsquigarrow$ 10%, interesse $\rightsquigarrow$ \$165 }
- {a $\rightsquigarrow$ 25, b $\rightsquigarrow$ 3, c $\rightsquigarrow$ 50}

Stati non corretti

- {nome $\rightsquigarrow$ Antonio, nome $\rightsquigarrow$ Paolo, eta' $\rightsquigarrow$ 25}
- {b $\rightsquigarrow$ 45, a $\rightsquigarrow$ 150, b $\rightsquigarrow$ 10}

Pseudo-linguaggio

- Utilizziamo inizialmente uno pseudo-linguaggio con un insieme di costrutti linguistici che costituiscono il nucleo di un qualunque linguaggio di programmazione reale.
- Senza entrare in eccessivi dettagli formali, nel presentare le notazioni utilizzate (sintassi) diamo anche una descrizione informale (semantica) di ciò che accade al momento dell'esecuzione in corrispondenza dei vari costrutti.

Il linguaggio contiene costrutti per:

- rappresentare semplici calcoli attraverso le comuni operazioni logico/aritmetiche (espressioni)
- modificare le associazioni nello stato (assegnamento e ingresso)
- controllare l'ordine di esecuzione delle azioni (controllo)
- fornire i risultati (produzione in uscita dello stato finale)

Espressioni

Espressioni Numeriche

Il linguaggio consente di rappresentare semplici calcoli algebrici, attraverso le usuali espressioni costruite a partire dai valori numerici e dalle operazioni di

somma $+$
sottrazione -
prodotto $*$
divisione intera $/$
resto della divisione intera $\%$.

Il significato di un'espressione è il suo valore ottenuto secondo le usuali regole di calcolo.

Esempio:

il valore di $3 * 5 + 6$ è 21
il valore di $3 * (5 + 6)$ è 33

Espressioni (cont.)

Oltre ai valori numerici, le espressioni possono contenere nomi simbolici: il calcolo di un'espressione che contiene un nome simbolico x dipende dallo stato, e precisamente dal valore associato al nome x nello stato.

Esempio:

il valore di $3 * (5 + x)$ è

- 33 in uno stato che contiene l'associazione $x \rightsquigarrow 6$
- 18 in uno stato che contiene l'associazione $x \rightsquigarrow 1$

Espressioni (cont.)

Espressioni Booleane

Il linguaggio consente poi di rappresentare condizioni ovvero espressioni il cui valore è un valore di verità (true o false).

Le condizioni sono costruite attraverso le usuali operazioni di confronto ($==$, $!=$, $<$, $>$, $<=$, $>=$) e, come nel caso delle espressioni aritmetiche, il loro valore può dipendere dallo stato.

Esempio: il valore di $y==x+1$ è

- true in uno stato che contiene le associazioni $x \rightsquigarrow 5$ e $y \rightsquigarrow 6$
- false in uno stato che contiene le associazioni $x \rightsquigarrow 5$ e $y \rightsquigarrow 9$

Condizioni più complesse possono essere costruite attraverso operatori logici quali negazione (simbolo !), congiunzione (simbolo &&) e disgiunzione (simbolo ||).

Espressioni (cont.)

- Significato di ! - il valore di verità di ! P è
  - true se il valore di verità di P è false
  - false se il valore di verità di P è true
- Significato di && - il valore di verità di P && Q è
  - true se i valori di verità di P e Q sono entrambi true
  - false altrimenti
- Significato di || - il valore di verità di P || Q è
  - false se i valori di verità di P e Q sono entrambi false
  - true altrimenti

Esempio:

- Il valore di `(y >= x) &&(x > 5)` è
  - true in uno stato che contiene le associazioni x $\rightsquigarrow$ 15 e y $\rightsquigarrow$ 30
  - false in uno stato che contiene le associazioni x $\rightsquigarrow$ 3 e y $\rightsquigarrow$ 30
- Il valore di `(y >= x) || (x > 5)` è
  - true in uno stato che contiene le associazioni x $\rightsquigarrow$ 2 e y $\rightsquigarrow$ 30
  - false in uno stato che contiene le associazioni x $\rightsquigarrow$ 3 e y $\rightsquigarrow$ 1

Modifica dello stato: ASSEGNAMENTO

- L'istruzione che consente di rappresentare modifiche di stato è usualmente nota come assegnamento.
- L'assegnamento consente di cambiare un'associazione nello stato, ovvero il valore associato nello stato ad un nome simbolico.
- Useremo per l'assegnamento la seguente notazione

  x = exp;

  dove x è un nome simbolico e exp una espressione.
- L'esecuzione dell'assegnamento x = exp; consiste nel
  (i) calcolare il valore, sia esso val, dell'espressione exp
  (ii) introdurre nello stato l'associazione x $\rightsquigarrow$ val
- Si noti che (ii) comporta la rimozione dallo stato della eventuale associazione già presente per il nome simbolico x (si parla a questo proposito di assegnamento distruttivo).

ASSEGNAMENTO: esempi

Vediamo alcuni esempi, indicando lo stato prima e dopo l'esecuzione degli assegnamenti proposti. Le associazioni modificate nello stato finale sono evidenziate in verde.

Stato iniziale	Assegnamento	Stato Finale
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	x = 5;	$\{ x \rightsquigarrow 5, y \rightsquigarrow 20\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	x = y*2;	$\{ x \rightsquigarrow 40, y \rightsquigarrow 20\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	x = x+1;	$\{ x \rightsquigarrow 11, y \rightsquigarrow 20\}$

Si noti come, nel terzo esempio, lo stesso nome simbolico x giochi un duplice ruolo:

- a destra del simbolo = indica un valore (il valore associato ad x nello stato iniziale)
- a sinistra del simbolo = indica l'associazione da modificare nello stato a seguito dell'assegnamento.

Modifica dello stato: INPUT

- La seconda istruzione di modifica dello stato consente di acquisire valori dal mondo esterno al momento dell'esecuzione. La notazione utilizzata è

  Input(x);

  dove x indica un nome simbolico.
- L'esecuzione consiste nel:
  (i) Acquisire un nuovo valore, sia esso val
  (ii) Introdurre nello stato l'associazione $x \rightsquigarrow val$
- Come nel caso dell'assegnamento, il punto (ii) comporta la rimozione dallo stato dell'eventuale associazione già presente per il nome simbolico x
- La presenza di tale istruzione permette di descrivere algoritmi generali in cui non tutti i dati sono noti a priori, ma lo saranno solo al momento dell'esecuzione.

Istruzioni di controllo: SEQUENZA

- Negli esempi visti in precedenza gli algoritmi sono stati descritti come sequenze di passi elementari del tipo

  Passo 1. azione 1
  Passo 2. azione 2
  ...

- Abbiamo utilizzato una sorta di numerazione per indicare l'ordine di esecuzione delle varie azioni: prima azione 1 poi azione 2 poi ....
- Nel linguaggio che stiamo introducendo, secondo le convenzioni sintattiche del C, una sequenza di azioni viene rappresentata mediante un blocco

```
{
istruzione 1
istruzione 2
...
istruzione n
}
```


SEQUENZA (cont.)

- Scriveremo dunque

  ```
  {
  istruzione 1
  istruzione 2
  ...
  }
  ```

  ad indicare che l'ordine di esecuzione dei singoli passi è quello testuale del programma.
- Si noti che ogni istruzione viene eseguita a partire dallo stato risultante dall'esecuzione dell'istruzione che la precede nella sequenza.
- Assegnamento e ingresso sono istruzioni semplici il blocco è un'istruzione composta.

SEQUENZA: esempi

Stato iniziale	Sequenza	Stato Finale
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = 5; x = x+y; \}$	$\{ x \rightsquigarrow 25, y \rightsquigarrow 20\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = x+1; y = x+1; \}$	$\{ x \rightsquigarrow 11, y \rightsquigarrow 12\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$\{ x = x+y; y = x+y; \}$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 50\}$

- In tutti gli esempi, lo stato finale si ottiene dall'esecuzione del secondo assegnamento nello stato intermedio risultante dall'esecuzione del primo assegnamento.
- Ad esempio, nel terzo caso:

Stato iniziale	Prima istruzione	Stato Intermedio
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$x = x+y;$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 20\}$
Stato intermedio	**Seconda istruzione**	**Stato Finale**
$\{ x \rightsquigarrow 30, y \rightsquigarrow 20\}$	$y = x+y;$	$\{ x \rightsquigarrow 30, y \rightsquigarrow 50\}$

Istruzioni di controllo: CONDIZIONALE

- Permette di determinare l'azione da intraprendere a seconda del verificarsi o meno di una condizione. La notazione utilizzata è la seguente

  if (condizione) istruzione1 **else** istruzione2

  dove condizione indica una espressione booleana, e istruzione1 istruzione2 sono istruzioni (semplici o composte).
- L'esecuzione del condizionale **if** (C) S1 **else** S2 consiste nel
  (i) Calcolare il valore, sia esso val, dell'espressione booleana C
  (ii) Eseguire S1 se val è true, eseguire S2 se val è false.
- Si noti che la presenza dell'istruzione condizionale non è in conflitto con il requisito di non ambiguità degli algoritmi: l'azione da intraprendere è univocamente determinata dal valore di verità della condizione e dunque non vi è facoltà di scelta da parte dell'esecutore.

CONDIZIONALE: esempi

Stato iniziale	Condizionale	Stato Finale
	if (x > y) z = x ; **else** z = y ;	
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$		$\{ x \rightsquigarrow 10, y \rightsquigarrow 20, z \rightsquigarrow 20\}$
$\{ x \rightsquigarrow 10, y \rightsquigarrow 5\}$		$\{ x \rightsquigarrow 10, y \rightsquigarrow 5, z \rightsquigarrow 10\}$

Istruzioni di controllo: RIPETIZIONE

- Consente di ripetere l'esecuzione di una istruzione (o sequenza di istruzioni) fino al verificarsi di una certa condizione (cfr. esempio del calcolo del prodotto tra due numeri).

  while (condizione) Istruzione

  dove condizione indica una espressione booleana.
- Terminologia: in **while** (C) S , la condizione C è detta guardia e l'struzione S è detta corpo (del ciclo).
- L'esecuzione di **while** (C) S consiste nel
  - (i) Calcolare il valore, sia esso val, dell'espressione booleana C
  - (ii) Se val è false, terminare l'esecuzione.
  - (iii) Se val è true, eseguire S e ripetere dal punto (i).

RIPETIZIONE

Esempio: Riformulazione del passo 4 dell'algoritmo per il prodotto (versione 2):

```
while (b>0)
{
  c = c+a;
  b = b-1;
}
```

- Intuitivamente, l'esecuzione di

  while (guardia) corpo

  corrisponde all'esecuzione di una sequenza del tipo

  { corpo corpo corpo ... corpo ...}

- In tale sequenza, ogni ripetizione dell'istruzione corpo viene detta iterazione del ciclo.
  In generale, non è possibile determinare a priori il numero di iterazioni (che può anche essere 0 nel caso in cui la guardia sia falsa nello stato iniziale).

RIPETIZIONE

- L'aspetto più critico è il fatto che l'esecuzione di un ciclo può essere fonte di non terminazione dell'esecuzione dell'intero algoritmo.

  while (3>0) x = 0;

- Un ciclo siffatto provoca la non terminazione dell'esecuzione, dal momento che il valore di verità della guardia è sempre true e dunque l'esecuzione corrisponde ad una sequenza infinita

  x = 0; x = 0; . . . . . ; x = 0; . . . . .

- È compito di chi definisce l'algoritmo assicurare che i cicli presenti non diano luogo a non terminazione.
- La pratica programmativa, ed il buon senso, suggeriscono ad esempio di utilizzare cicli in cui:
  - il valore di verità della guardia dipende dallo stato;
  - l'esecuzione del corpo comporta modifiche di associazioni nello stato dalle quali dipende il valore di verità della guardia.

RIPETIZIONE

- Le indicazioni appena viste non sono tuttavia sufficienti a garantire la terminazione dell'esecuzione. Si consideri ad esempio il ciclo:

  while (x>0) x = x+1;

  che soddisfa entrambi i requisiti richiesti, ma che può non terminare nel caso in cui venga eseguito a partire da uno stato dove il valore associato al nome x è un valore positivo.
- A partire dagli anni '70 sono stati sviluppati dei metodi formali per la verifica di correttezza di programmi, che comprendono tecniche per la dimostrazione formale di proprietà di terminazione dei cicli.

Istruzioni di controllo: OUTPUT

- Permette di rendere visibile all'esterno la parte desiderata dello stato. La notazione utilizzata è

  Output(x);

  dove x indica un nome simbolico.
- L'esecuzione di questa istruzione consiste nel:
  (i) Recuperare il valore, sia esso val, associato nello stato al nome simbolico x.
  (ii) Produrre in uscita tale valore.
- Questa operazione non comporta la modifica dello stato. Nei linguaggi di programmazione reali, l'esecuzione delle operazioni di uscita produce di solito la visualizzazione di valori su supporti fisici quali video, stampanti etc.

Problema: Calcolo del valore assoluto di un numero

Vediamo alcuni esempi di specifica di algoritmi che utilizzano lo pseudo-linguaggio.

Specifica:

Stato iniziale: $\{$ numero $\rightsquigarrow$ A $\}$
Stato finale: $\{$ risultato $\rightsquigarrow |A| \}$
dove A indica un (generico) valore intero.

Algoritmo:

```
if (numero > 0)
          risultato =  numero;
     else risultato =  - numero;
```


Problema: Ordinare due interi positivi

Specifica:

Stato iniziale: $\{$ num1 $\rightsquigarrow$ A, num2 $\rightsquigarrow$ B $\}$
Stato finale: $\{$ num1 $\rightsquigarrow$ *max*(A,B), num2 $\rightsquigarrow$ *min(A,B)* $\}$

Algoritmo:

```
if (num1 < num2)
    {
        temp =  num1;
        num1 =  num2;
        num2 =  temp;
    }
else ;
```

Problema: Ordinare due interi positivi (cont.)

- Si noti l'utilizzo di un nome simbolico aggiuntivo (temp): è essenziale per poter effettuare correttamente lo scambio tra i valori associati ai due interi (a causa del carattere distruttivo dell'assegnamento e della sequenzialità dell'esecuzione).
- È facile verificare che una sequenza del tipo

```
x = y;
y = x;
```

non consente, in generale, di scambiare i valori associati a x e y.

Esempio:

Stato iniziale		Stato Intermedio
$\{ x \rightsquigarrow 10, y \rightsquigarrow 20\}$	$x = y;$	$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$
Stato intermedio		**Stato Finale**
$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$	$y = x$	$\{ x \rightsquigarrow 20, y \rightsquigarrow 20\}$

Problema: Elevamento a potenza

Specifica:

Stato iniziale: $\{$ base $\rightsquigarrow$ A, esponente $\rightsquigarrow$ B $\}$ con A$>$0 e B $\geq$0
Stato finale: $\{$ risultato $\rightsquigarrow A^B\}$

- L'algoritmo si basa sulla seguente, ben nota, definizione di elevamento a potenza.

$$A^0 = 1$$
$$A^B = \underbrace{A \times A \times \ldots \times A}_{B \text{ volte}} \qquad (\text{se } B > 0)$$

Algoritmo:

```
risultato = 1;
while (esponente > 0)
{
    risultato = risultato * base;
    esponente = esponente - 1;
}
```

- Nell'algoritmo, si assume che i valori iniziali di base ed esponente soddisfino i requisiti della specifica.

Input/Output

- Nei problemi visti fino ad ora non ci siamo preoccupati di acquisire i dati iniziali né di produrre in uscita i risultati finali.

  Esempio: Ordinare due valori interi acquisiti in input e stampare i valori ordinati.

  Algoritmo:

```
Input(num1);
Input(num2);,
if num1 < num2
{
     temp = num1;
      num1 = num2;
     num2 = temp;
}
else ;
Output(num1);
Output(num2);
```

Problema: Calcolare quoziente e resto della divisione tra due numeri naturali non nulli.

Specifica:

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B\}$ con $A,B > 0$

Stato finale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B,\ q \rightsquigarrow Q,\ r \rightsquigarrow R\}$
con $A = Q \cdot B + R$ e $0 \leq R < B$

Supponiamo un esecutore in grado di eseguire solo le operazioni elementari di somma e sottrazione.

Algoritmo:

```
q = 0;
r = x;
while (r ≥ y)
{
    q = q + 1;
    r = r - y;
}
```


Supponiamo di avere lo stesso problema ma un esecutore piu' potente.

Specifica:

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B\}$ con $A,B > 0$

Stato finale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B,\ q \rightsquigarrow Q,\ r \rightsquigarrow R\}$
con $A = Q \cdot B + R$ e $0 \leq R < B$

Supponiamo adesso un esecutore in grado di eseguire anche l'operazione di moltiplicazione ma non di divisione.

Algoritmo:

```
c = 1;
r = 0;
while ((c*y) ≤ x)
{
    c = c + 1;
}
r=x-((c-1)*y);
q=c-1;
```

Calcolare il MCD con l'algoritmo di Euclide

- Dati due naturali non nulli si calcola il MCD utilizzando le seguenti proprietà:

$$\begin{aligned} MCD(x,x) &= x \\ MCD(x,y) &= MCD(x-y,y) \qquad \text{se } x > y \\ MCD(x,y) &= MCD(x,y-x) \qquad \text{se } y > x \end{aligned}$$

Specifica:

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B\}$ con $A,B > 0$

Stato finale: $\{x \rightsquigarrow MCD(A,B)\}$

Calcolare il MCD con l'algoritmo di Euclide (cont.)

Algoritmo:

```
while (x ≠ y)
{
    if (x > y)
        x = x - y;
    else
        y = y - x;
}
```

Lo stato

- Insieme di associazioni tra nomi e valori.

  $x \rightsquigarrow 5$

- Nelle specifiche, si vogliono spesso indicare valori costanti ma generici. Utilizziamo le seguenti convenzioni:
  - usiamo lettere minuscole per i nomi simbolici utilizzati nello stato e negli algoritmi;
  - usiamo lettere maiuscole per indicare costanti generiche
- Laddove necessario, possiamo specificare condizioni al contorno sul dominio di appartenenza di tali costanti.

  Esempio:

  $\{\text{naturale} \rightsquigarrow A, \text{base} \rightsquigarrow 2\}$ con $A \geq 0$

  Al nome simbolico naturale è associato un generico valore naturale ($A \geq 0$), mentre al nome simbolico base è associata la costante 2.

ATTENZIONE: all'interno degli algoritmi non è possibile utilizzare esplicitamente le costanti generiche.

Esempio: Elevamento a potenza

Specifica:

Stato iniziale: $\{ \text{base} \rightsquigarrow A, \text{esponente} \rightsquigarrow B \}$ con $A>0$ e $B \geq 0$

Stato finale: $\{ \text{risultato} \rightsquigarrow A^B \}$

Algoritmo:

```
risultato = 1;
while (esponente > 0) {
    risultato = risultato * A;
    esponente = esponente - 1;
}
```

NO!

Le espressioni non possono contenere costanti generiche (proprio perché tali!).

Esempi

Esempio: Ordinare tre valori interi distinti tra loro

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B,\ z \rightsquigarrow C\}$ con A,B,C distinti tra loro

Stato finale: $\{ x \rightsquigarrow \max(\{A,B,C\}),\ y \rightsquigarrow \max(\{A,B,C\} \setminus \{\max(\{A,B,C\})\}),\ z \rightsquigarrow \min(\{A,B,C\})\}$

Algoritmo

Passo 1. "Ordiniamo" x e y, portandoci nello stato intermedio

Stato 1: $\{ x \rightsquigarrow \max(A,B),\ y \rightsquigarrow \min(A,B),\ z \rightsquigarrow C \}$

Passo 2. "Ordiniamo" x e z, portandoci nello stato intermedio

Stato 2: $\{ x \rightsquigarrow \max(\{A,B,C\}),\ y \rightsquigarrow \min(A,B),\ z \rightsquigarrow \min(\max(A,B),\ C) \}$

Passo 3. "Ordiniamo" y e z, portandoci nello stato finale desiderato

Soluzione nello pseudo-linguaggio

```
if (x < y)
{
    temp = x;
    x = y;
    y = temp;
}
else ;
  { x ⇝ max(A,B), y ⇝ min(A,B), z ⇝ C }
if (x < z)
{
    temp = x;
    x = z;
    z = temp;
}
else ;
  { x ⇝ max({A,B,C}), y ⇝ min(A,B), z ⇝ min(max(A,B), C) }
if (y < z)
{
    temp = y;
    y = z;
    z = temp;
}
else ;
  Stato finale
```

Esempio: Calcolo del fattoriale di un numero naturale. Ricordiamo che:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n & \text{se } n > 0 \end{cases}$$

Specifica:

Stato iniziale: $\{\mathtt{n} \rightsquigarrow \mathtt{N}\}$ con $\mathtt{N} \geq 0$

Stato finale: $\{\mathtt{n} \rightsquigarrow \mathtt{N}, \mathtt{fatt} \rightsquigarrow \mathtt{N!}\}$

- Attenzione: la specifica impone che il valore di n non deve cambiare!

Soluzione:

```
fatt = 1;
i = 1;
while (i <= n)
{
    fatt = fatt * i;
    i = i + 1;
}
```

- i viene spesso chiamata variabile di controllo del ciclo, dal momento che l'esecuzione di ogni iterazione dipende dal valore di i.

  Viene incrementata di 1 alla fine di ogni iterazione

  $\Longrightarrow$ per ogni valore di i $\in$ [1,N] .....

- Ad ogni iterazione del ciclo, vale la seguente proprietà:

  $$\mathtt{fatt} = \prod_{\mathtt{j}=1}^{\mathtt{i}-1} \mathtt{j} \quad = (\mathtt{i}\text{-}1)! \qquad (\bullet)$$

- Al termine del ciclo, quando cioè i=N+1, la (•) diventa

  fatt = N!, ovvero lo stato finale desiderato.

Sequenze: nomi simbolici con indice

Consentiamo anche l'uso di nomi simbolici con indice, per rappresentare sequenze.

$$c[i] \rightsquigarrow 5$$

dove i è un valore naturale. Ad esempio:

$$\{i \rightsquigarrow K,\ c[0] \rightsquigarrow 0,\ \ldots,\ c[K-1] \rightsquigarrow 0\} \qquad K \geq 0$$

Nello stato sono presenti:

- un'associazione per il nome simbolico i, al quale è associato un valore naturale K.
- K associazioni per i nomi simbolici c[0], c[1], ..., c[K-1], a ciascuno dei quali è associato il valore 0.

All'interno degli algoritmi, consentiamo di riferire nomi simbolici con indice nella forma c[*exp*] dove *exp* deve essere una espressione a valori naturali. Anche in questo caso *exp* non deve contenere costanti generiche (nell'esempio, non è lecito un assegnamento del tipo c[K-1] = 5, mentre è lecito c[i-1] = 5).

Esempio: Calcolare il numero di elementi pari in una sequenza data.

Stato iniziale: $\{dim \rightsquigarrow K,\ c[0] \rightsquigarrow V_1,\ \ldots,\ c[K\text{-}1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{count \rightsquigarrow \#\{\ j \mid j \in [0,K\text{-}1] \wedge\ V_j \text{ pari}\}\ \}$

- Dobbiamo calcolare

$$\sum_{\substack{0 \leq j \leq dim\text{-}1 \\ c[j] \text{ pari}}} 1$$

- Di nuovo, utilizziamo un ciclo controllato da una variabile di controllo i che assume tutti i valori nell'intervallo [0,dim-1].
- Facciamo in modo che, ad ogni iterazione, valga la seguente proprietà

$$count = \sum_{\substack{0 \leq j \leq i\text{-}1 \\ c[j] \text{ pari}}} 1$$

- Se, alla fine del ciclo, i=dim, abbiamo quanto desiderato e cioè

$$count = \sum_{\substack{0 \leq j \leq dim\text{-}1 \\ c[j] \text{ pari}}} 1$$

Soluzione:

```
count = 0;
i = 0;                          punto 1
while (i <dim)
{
    if (c[i] % 2 == 0)
        count = count + 1;
    else ;
    i = i + 1;
}
```

- Nello stato iniziale del while, al punto (1), count $\rightsquigarrow$ 0, i $\rightsquigarrow$ 0 e dunque

$$\sum_{\substack{1\le j\le i\text{-}1\\ c_j \text{ pari}}} 1 = \sum_{\substack{0\le j\le -1\\ c_j \text{ pari}}} 1 = 0 = \text{count}$$

- il corpo del while mantiene vera la proprietà

$$\text{count} = \sum_{\substack{0\le j\le i\text{-}1\\ c_j \text{ pari}}} 1$$

Considerazioni generali

- In molti problemi è necessario operare allo stesso modo su tutti gli elementi di un intervallo dato

  per ogni $j \in [a,b]$ OP_j

  esempio: $\sum_{j=a}^{b} exp_j$

- in tutti questi casi si ricorre a cicli in cui una variabile di controllo permette di scandire tutto l'intervallo di interesse

```
i = a;
while (i <= b)
{
    ⟨operazione in funzione di i⟩
    i = i+1;
}
```

Esercizi proposti

Problema 1: Calcolare il numero di occorrenze di un valore dato in una sequenza data

Stato iniziale: $\{dim \rightsquigarrow K, val \rightsquigarrow V, c[0] \rightsquigarrow V_0, \ldots, c[K\text{-}1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{occ \rightsquigarrow \#\{\ j \ \mid\ j \in [0,K\text{-}1] \wedge\ V_j = V\}\ \}$

Problema 2: Calcolare il massimo e il minimo di una sequenza data di interi

Stato iniziale: $\{dim \rightsquigarrow K, c[0] \rightsquigarrow V_0, \ldots, c[K\text{-}1] \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{massimo \rightsquigarrow max(\{V_0,\ldots,V_{K-1}\}),$
$minimo \rightsquigarrow min(\{V_0,\ldots,V_{K-1}\})\}$

Problema 3: Calcolare il numero di occorrenze del valore massimo in una sequenza di interi

Stato iniziale: ? Stato finale: ?

Problema 4: Calcolare il numero di occorrenze di una cifra nella rappresentazione decimale di un numero naturale

Soluzione problema 1:

Stato iniziale: $\{dim \rightsquigarrow K, val \rightsquigarrow V, c_0 \rightsquigarrow V_0, \ldots, c_{K-1} \rightsquigarrow V_{K-1}\}$ con $K > 0$

Stato finale: $\{occ \rightsquigarrow \#\{\ j \ \mid\ j \in [0,K\text{-}1] \wedge\ V_j = V\}\ \}$

```
occ = 0;
i = 0;
while (i < dim)
{
    if (c[i] == val)
        occ = occ + 1;
    else ;
    i = i + 1;
}
```

Ad ogni iterazione del ciclo vale la proprietà:

$occ = \#\{\ j \ \mid\ j \in [0,i) \wedge\ c[j] = val\}$

Soluzione problema 2

Stato iniziale: $\{dim \rightsquigarrow K, c_0 \rightsquigarrow V_0, \ldots, c_{K-1} \rightsquigarrow V_{K-1}\}$ con $K > 0$
Stato finale: $\{massimo \rightsquigarrow max(\{V_0,\ldots,V_{K-1}\}),$
$minimo \rightsquigarrow min(\{V_0,\ldots,V_{K-1}\})\}$

- Preoccupiamoci prima di calcolare il massimo
- Scorriamo l'intera sequenza, attraverso una variabile di controllo i, facendo in modo che, ad ogni scansione, valga la proprietà

$$massimo = max\{c[j] \mid j \in [0, i)\}$$

```
massimo = c[0];
i = 1;
while (i < dim)
{        qui massimo = max{c[0], ..., c[i-1]}
    if (c[i] > massimo)
        massimo = c[i];
    else ;
    i = i + 1;  anche qui massimo = max{c[0], ..., c[i-1]}
}
```


- Il calcolo del minimo è analogo. Otteniamo:

```
massimo = c[0];
minimo = c[0];
i = 1;
while (i <dim)
{
    if (c[i] > massimo)        calcolo del massimo
        massimo = c[i];
    else ;
    if (c[i] < minimo)       calcolo del minimo
        minimo = c[i];
    else ;
    i = i + 1;
}
```

- Usiamo un algoritmo piu' efficiente: diminuiamo il numero di confronti.

```
massimo = c[0];
minimo = c[0];
i = 1;
while (i < dim)
{
    if (c[i] > massimo)        calcolo del massimo
        massimo = c[i];
    else
        if (c[i] < minimo)       calcolo del minimo
        minimo = c[i];
        else ;
    i = i + 1;
}
```


Soluzione problema 3

Stato iniziale: $\{\text{dim} \rightsquigarrow K,\ c[0] \rightsquigarrow V_0, \ldots, c_{K-1} \rightsquigarrow V_{K-1}\}$

Stato finale: $\{\text{occ} \rightsquigarrow \#\{j \mid j \in [0,K\text{-}1] \wedge V_j = max\{V_0, \ldots, V_{K-1}\}\}\}$

- Un algoritmo ingenuo:
  - Calcoliamo il valore massimo come nel problema 2
  - Scandiamo nuovamente la sequenza per contare il numero di occorrenze del massimo
- Comporta una duplice scansione della sequenza
- Un algoritmo che comporta una singola scansione si ottiene adattando quello visto per il calcolo del massimo

```
massimo = c[0];
i = 1;
occ = 1;
while (i < dim)
{
    if (c[i] > massimo)
    {
        massimo = c[i];
        occ = 1;
    }
    else
        if (c[i] == massimo)
            occ = occ + 1;
        else ;
    i = i + 1;
}
```


Soluzione problema 4

Stato iniziale: $\{\text{numero} \rightsquigarrow N, \text{cif} \rightsquigarrow C\}$ con $N > 0$

Stato finale: $\{\text{occ} \rightsquigarrow \#\{\, j \mid j \in [0,K] \wedge C_j = C\} \,\}$

dove $N = (C_K \ldots C_0)_{10}$

- Analogo al problema 1, ma questa volta dobbiamo anche calcolare gli elementi della sequenza
- La lunghezza della sequenza non è nota a priori

```
n = numero; occ = 0;
while (n != 0)
{
    nuova_cifra = n % 10;
    if (nuova_cifra == cif)
        occ = occ + 1;
    else ;
    n = n / 10;
}
```

- questa volta la variabile di controllo è n

- Variante: acquisire un numero naturale ed una cifra decimale, contare il numero di occorrenze di quest'ultima nella rappresentazione decimale del numero letto, e produrlo in uscita.

```
Input(numero);
Input(cif);
n = numero; occ = 0;
while (n != 0)
{
...
if (nuova_cifra == cif) ...
}
Output(occ);
```

- da uno stato iniziale che non contiene associazioni, o meglio su cui non facciamo alcuna ipotesi, ci portiamo in uno stato che corrisponde allo stato iniziale del problema precedente
- nello stato finale, produciamo in uscita il risultato calcolato