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: $\{n \rightsquigarrow N\}$ con $N \geq 0$

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

- 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:

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

- 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