Algoritmi e Strutture di Dati

Pseudocodifica

m.patrignani

Nota di copyright

Che cos'è la pseudocodifica

- La pseudocodifica è un linguaggio
  - non è né compilato né interpretato, in quanto non deve essere eseguito su nessuna piattaforma
- La pseudocodifica serve ai programmatori per descrivere un algoritmo
  - la sintassi non è rigorosa, in quanto l'unico requisito è che la semantica sia comprensibile
  - si suppone che un algoritmo descritto tramite pseudocodice possa essere implementato in qualsiasi linguaggio di programmazione

Restrizioni della pseudocodifica

- Vista la sua universalità, la pseudocodifica presenta alcune restrizioni
  - il tipo delle variabili è fisso e non è modificabile a run-time
    - non è possibile eseguire casting implicito od esplicito
  - esistono i riferimenti, ma non è supportata l'algebra dei puntatori
    - non esistono gli operatori * e &

Variabili e assegnazioni

- Le variabili dello pseudocodice hanno associato un tipo (intero, reale, booleano, …) ma non necessitano di dichiarazione
  - la prima volta che vengono usate si inferisce il loro tipo
  - in caso di equivoco si ricorre a dei commenti
- Le assegnazioni sono denotate da un segno di ugualianza (=)

```
1. max = -1          // max è un intero
2. max = max + 1     // incremento max
```

```
1. /* definisco un reale che chiamo area */
2. area = base * altezza / 2
```


Semantica delle assegnazioni

- Un'assegnazione è un'operazione

```
<left-value> = <right-value>
```

individua una zona di memoria che ha un determinato tipo (e quindi una specifica dimensione)

è una generica espressione che produce un valore del tipo coerente con il left-value

- Queste due assegnazioni sono diverse

```
a = b  // nella cella di mem di a metto il valore di b
```

```
b = a  // nella cella di mem di b metto il valore di a
```


Esempi di assegnazioni

- <variabile> = <costante>

```
1.  verificato = false
```

- <variabile> = <variabile>

```
1.  a = b
```

- <variabile> = <espressione generica>

```
1.  verificato = verificato and FUNZIONE(a, b, c)
```

- Uso scorretto delle assegnazioni

```
1.  a = b + c
2.  b = 3
3.  c = 4
4.  return a
```


Istruzioni condizionali

- **`if-then-else`**, con ovvio significato
  - spesso per brevità si tace il “then”
- Un blocco di istruzioni è denotato tramite indentazione

```
1.  if i < 0 then
2.      // prendo il valore assoluto di i
3.      i = -i
```

```
1.  if i < 0 then
2.      abs = -i
3.      i_minore_zero = true
4.  else
5.      abs = i
6.      i_minore_zero = false
```


Espressioni booleane

- Sono espressioni booleane
  - le variabili booleane e le costanti **`true`** e **`false`**
  - il risultato di operatori booleani come: and, or, not
    - possono anche essere scritti &&, ||, !
  - il risultato di operatori relazionali come: ==, >, ≥, <, ≤

```
1.  if i < max and trovato == false then
2.       i = i+1
```

- Attenzione
  - talvolta viene usato il simbolo uguale (=) per denotare l'operatore relazionale e la freccia (←) per l'assegnazione
    - è però indispensabile utilizzare i simboli in modo coerente!

Leggi di de Morgan

- Quando un not si distribuisce su un and quest'ultimo viene trasformato in or
- Le seguenti espressioni sono dunque equivalenti
  - not (condizione-1 and condizione-2)
    - esempio: spero che non piova e io sia all'aperto
  - (not condizione-1) or (not condizione-2)
    - esempio: spero che non piova oppure che io non sia all'aperto
- Quando un not si distribuisce su un or quest'ultimo viene trasformato in and
- Le seguenti espressioni sono dunque equivalenti
  - not (condizione-1 or condizione-2)
    - esempio: spero che non piova o nevichi
  - (not condizione-1) and (not condizione-2)
    - esempio: spero che non piova e che non nevichi

Istruzione ripetitiva **for**

- Esempi

```
1.  for i = 0 to A.length-1
2.      /* pongo a zero tutte le celle di A */
3.      A[i] = 0
```

```
1.  for i = A.length-1 down to 1
2.      /* traslo in avanti tutti i valori di A */
3.      A[i] = A[i-1]
4.  A[0] = new    // inserisco new in testa
```

- Non è possibile porre condizioni aggiuntive ad un for
  - esempio errato

```
1.  for i = 0 to A.length-1 and trovato == false
```


Istruzione ripetitiva **while**

- Il blocco viene eseguito finché (=mentre) la condizione è verificata
  - la condizione è una "condizione di permanenza nel ciclo"
  - se la condizione è falsa il blocco non viene eseguito mai
  - spesso per brevita si tace il "do"

```
1.  // pongo a zero tutte le posizioni di A
2.  i = 0
3.  while i < A.length do
4.      A[i] = 0
5.      i = i+1
```

```
1.  while i < A.length and not trovato do
2.      if A[i] == quello_che_cerco then
3.          trovato = true
4.      else
5.          i = i+1
```


Istruzione ripetitiva repeat until

- Il blocco viene eseguito finché (=mentre) la condizione è falsa
  - la condizione è una "condizione di uscita dal ciclo"
  - il blocco viene eseguito almeno la prima volta
- Esempio

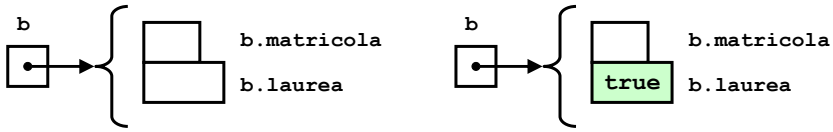
```
1.  // pongo a zero tutte le posizioni di A
2.  i = 0
3.  repeat
4.      A[i] = 0
5.      i = i+1
6.  until i == A.length
```


Pseudocodice: oggetti

- E' possibile definire dati compositi (oggetti) stutturati in attributi
- Si accede all'attributo **`campo`** dell'oggetto **`x`** tramite **`x.campo`**

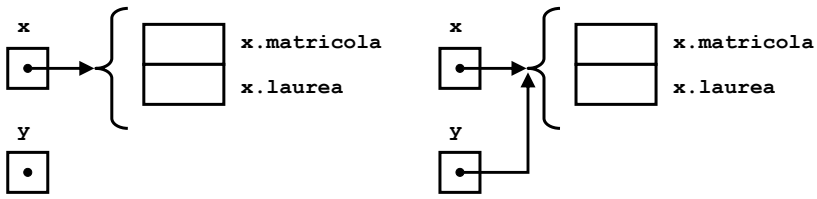
```
1.  // b è un oggetto con gli attributi
2.  // matricola (intero) e laurea (booleano)
3.  b.laurea = true      // con mille auguri!
```

- Una variabile che rappresenta un oggetto è trattata come un riferimento (puntatore) all'oggetto



Oggetti e operazioni sui riferimenti

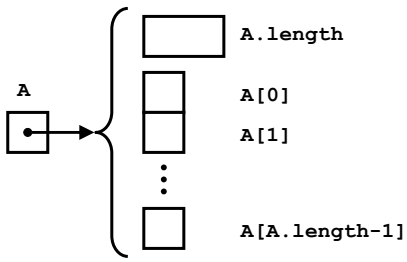
- Se **`x`** e **`y`** sono due (riferimenti ad) oggetti
  - dopo l'assegnazione **`y = x`** si ha **`y.campo = x.campo`**
  - se si modifica un attributo di **`x`** (**`x.campo = 3`**), si modifica anche il corrispondente attributo di **`y`** (**`y.campo = 3`**)



- Per porre a zero un riferimento si usa la costante **`NULL`**
- ATTENZIONE:
  - in linguaggio C si fa distinzione tra un oggetto ed il suo riferimento
  - in pseudocodice abbiamo solo il riferimento

Pseudocodice: array

- Si possono definire array di qualunque tipo
- A[i] identifica l'elemento dell'array A in posizione i
- La lunghezza dell'array A è data da A.length
  - A.length è un campo, non una funzione
  - ATTENZIONE: non c'è equivalente di A.length in linguaggio C
- Le posizioni dell'array A vanno da 0 ad A.length-1



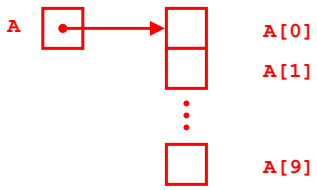
di fatto, un array è visto come un oggetto che ha dei campi indicizzati tramite un intero

Esempi di uso di array

```
1.  A[0] = 1              // A è un array 10 di interi
2.  A[A.length-1] = 0     // l'ultimo elemento è zero
```

```
1.  // A è un array contenente 3 array di 10 interi
2.  A[2][9] = 0       // l'ultimo elemento di A[2] è 0
3.  A[2,9] = 0        // equivalente alla precedente
```

- ATTENZIONE: in linguaggio C gli array non hanno il campo length!!



la lunghezza dell'array in linguaggio C deve essere memorizzata in un'altra variabile, come per esempio

lengthA | 10

Pseudocodice e tipi delle variabili

- Lo psedocodice è privo di definizioni di tipo
- Le variabili si suppongono allocate in memoria nel momento in cui vengono menzionate per la prima volta
- I commenti vengono utilizzati per esplicitare il loro tipo, se ciò non è evidente dal contesto (come nelle linee 2 e 5 qui sotto)
  - ciò è evidentemente necessario per gli oggetti

```
1.  i = p - 1          (in assenza di commento il tipo di i è lo stesso di p)
2.  x.info = 1         /* x è un oggetto con i campi: info
3.                        (intero) next e prev (riferimenti ad
4.                        oggetti dello stesso tipo) */
5.  TEMP[1] = 3        // TEMP è un array di 20 interi
6.  y = x              (il tipo di y è lo stesso di x: commento non necessario)
7.  TEMP[2] = 4        (TEMP già noto: commento non necessario)
```


Pseudocodice e procedure

- Le variabili utilizzate in una procedura sono sempre locali alla procedura stessa
  - non esistono variabili globali
- Alcune procedure non ritornano alcun valore
  - la sequenza delle loro istruzioni termina, oppure viene eseguita l'istruzione **`return`**
- Alcune procedure ritornano un valore
  - le loro istruzioni terminano sempre con
    `return` <*espressione*>
    - dove la valutazione di <*espressione*> genera il valore ritornato
  - non si possono ritornare due o più valori
    - eventualmente si usano degli oggetti costruiti opportunamente

Procedure e parametri

- Il passaggio dei parametri ad una procedura è solo per valore
  - la procedura riceve il valore dei *parametri formali* tramite i *parametri attuali* presenti nella chiamata
    - un parametro attuale è in generale un'espressione, che può anche ridursi ad una singola variabile
- Un parametro è a tutti gli effetti una variabile locale inizializzata con il valore ricevuto al momento della chiamata
- se il valore di un parametro formale viene modificato, la modifica non ha effetto all'esterno sul parametro attuale
  - non c'è "side effect"
- Se però il parametro è un oggetto o un'array, l'oggetto non viene riprodotto, ma viene passato il valore del riferimento (indirizzo, puntatore) all'oggetto/array stesso
  - le assegnazioni `oggetto.campo=valore` e `array[indice]=valore` hanno quindi un side effect sull'oggetto e sull'array passati come parametro

Esercizi sullo pseudocodice

1. Scrivi la procedura `MASSIMO`(*a*,*b*) che ritorni il massimo tra due interi *a* e *b*
2. Scrivi la procedura `MASSIMO`(*A*) che riceva come parametro un array di interi *A* e ritorni il massimo dei valori contenuti
3. Scrivi la procedura `SOMMA`(*M*) che riceva come parametro una matrice (un array di array) di interi *M* e ritorni la somma dei valori contenuti
   - puoi usare una funzione `SOMMA`(*A*) che somma gli elementi di un array per realizzare `SOMMA`(*M*)

Esercizi sullo pseudocodice

4. Scrivi la procedura `POSITIVO`(*A*) che riceva come parametro un array di interi *A* e ritorni true se *A* contiene solo valori maggiori di zero, false altrimenti
5. Scrivi la procedura `POSIZIONE-MASSIMO`(*A*) che riceva come parametro un array di interi *A* e ritorni il valore massimo contenuto e la sua posizione nell'array
   - attenzione: non si possono ritornare due valori!
   - occorre ritornare un oggetto con due campi