algoritmi e strutture di dati

implementazioni di liste con oggetti e riferimenti

m.patrignani

nota di copyright

strutture di dati con oggetti e riferimenti

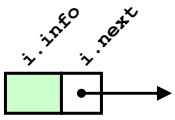
- alcuni linguaggi supportano oggetti e riferimenti
  - lo pseudocodice è uno di questi
- con oggetti e riferimenti si possono realizzare strutture di dati elementari in modo più naturale
- vedremo la realizzazione del tipo astratto di dato lista
  - pile e code possono essere rivisti come casi particolari di liste
  - due principali varianti implementative:
    - lista singolarmente concatenata
    - lista doppiamente concatenata

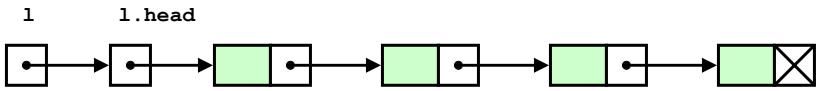
operazioni su una lista di interi

`CREATE-LIST()`	ritorna il riferimento ad una lista vuota
`HEAD(l)`	ritorna l'iteratore del primo elemento della lista
`LAST(l)`	ritorna l'iteratore dell'ultimo elemento della lista
`NEXT(l,i)`	ritorna l'iteratore dell'elemento che segue `i` nella lista ritorna un iteratore invalido se `i` è l'ultimo elemento
`PREV(l,i)`	ritorna l'interatore dell'elemento che precede `i` nella lista ritorna un iteratore invalido se `i` è il primo elemento
`INSERT(l,n)`	inserisce l'elemento `n` in testa alla lista `l`
`INSERT-BEFORE(l,n,i)`	inserisce l'elemento `n` prima della posizione i
`ADD(l,n)`	aggiunge `n` in coda alla lista `l`
`ADD-AFTER(l,n,i)`	aggiunge l'elemento `n` dopo la posizione i
`DELETE(l,i)`	rimuove l'elemento in posizione i dalla lista `l`
`DELETE(l,n)`	rimuove l'elemento `n` dalla lista `l`
`EMPTY(l)`	vuota la lista
`SEARCH(l,n)`	ritorna l'iteratore dell'elemento n nella lista `l`
`IS-EMPTY(l)`	ritorna `true` se la lista è vuota, altrimenti ritorna `false`

lista concatenata (singly linked list)

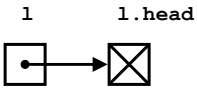
- l'iteratore `i` è un riferimento ad un nodo della lista, che è un oggetto composto dai seguenti attributi
  - `i.info`
    - elemento in lista del tipo opportuno
    - può essere un riferimento ad un oggetto esterno con dati satellite
  - `i.next`
    - riferimento al nodo seguente o `NULL`
- una lista `l` ha un solo attributo
  - `l.head`
    - riferimento al primo nodo





lista concatenata: lista vuota

- quando la lista `l` è vuota `l.head` è `NULL`



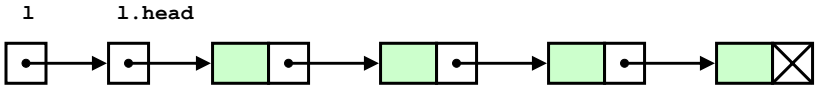
- pseudocodice delle procedure `IS-EMPTY` e `EMPTY`

```
IS-EMPTY(l)
1.  return l.head == NULL
```

```
EMPTY(l)
1.  l.head = NULL    ▷ lo pseudocodice non dealloca memoria
```

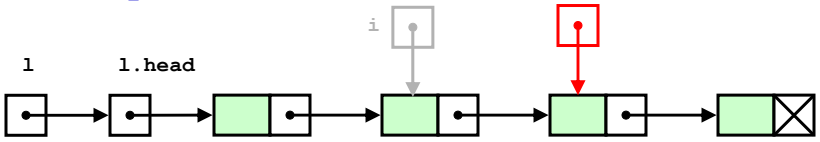

lista concatenata: first e next

- `FIRST`: iteratore dell'elemento affiorante




```
FIRST(l)
1.  return l.head     ▷ potrebbe essere NULL
```

- `NEXT`: prossimo elemento




```
NEXT(l,i)
1.  return i.next     ▷ il parametro l non è utilizzato
```


realizzazione di funzioni elementari

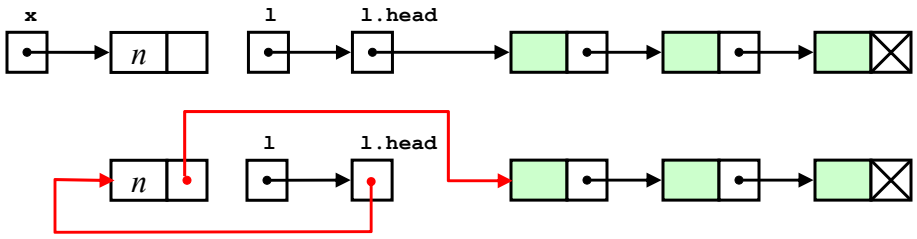
- la semplicità di alcune funzioni (come `IS-EMPTY`, `EMPTY`, `FIRST`, `NEXT`, ecc) induce a sostituirle con le istruzioni opportune direttamente nello pseudocodice
- esempio

```
...
   if !IS-EMPTY(l)
   then ...
...
   x = NEXT(l,x)
...
```

```
...
   if l.head != NULL
   then ...
...
   x = x.next
...
```


lista concatenata: inserimento in testa

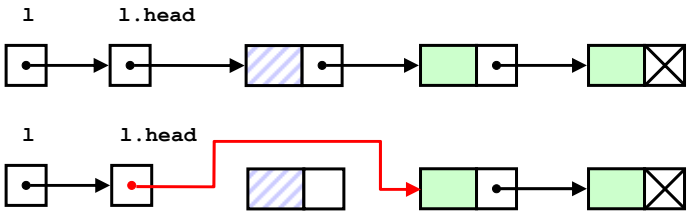
- `INSERT`: inserimento di *n* in testa alla lista




```
INSERT(l,n)      ▷ n è un intero
1. x.info = n    ▷ x è un nuovo oggetto con due campi:
2.               ▷     x.info (intero)
3.               ▷     x.next (rif. ad analogo oggetto)
4. x.next = l.head
5. l.head = x
```


lista concatenata: cancellazione

- `DELETE-FIRST`: rimozione del primo nodo




```
DELETE-FIRST(l)
1. ▷ NOTA: lo pseudocodice non dealloca l'elemento
2. if l.head == NULL
3.     error("lista vuota")
4. else
5.     l.head = l.head.next
```

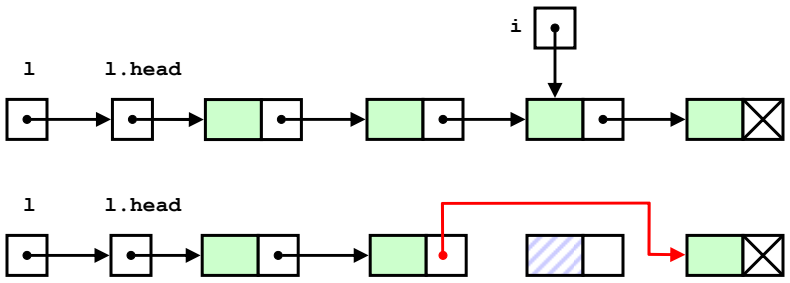

esercizi: liste singolarmente concatenate

esercizi sullo scorrimento delle liste

1. scrivi lo pseudocodice della procedura `MASSIMO(l)` che ritorna il valore del massimo elemento contenuto in una lista singolarmente concatenata di interi
2. scrivi lo pseudocodice della procedura `SOMMA(l)` che ritorna la somma degli elementi contenuti in una lista singolarmente concatenata di interi

lista concatenata: cancellazione

- `DELETE(l,i)`: cancellazione del nodo `i`
- la cancellazione di un nodo diverso dal primo è poco efficiente in una lista singolarmente concatenata



- occorre infatti modificare l'attributo `next` del nodo che lo precede

esercizi: liste singolarmente concatenate

3. scrivi lo pseudocodice della procedura `SEARCH(l,u)` che ritorna il riferimento all'elemento `i` che contiene il valore intero `u` in una lista singolarmente concatenata di interi (oppure `NULL` se `u` non è nella lista)
   - discuti la complessità dell'algoritmo in funzione del numero n degli elementi in lista
4. scrivi lo pseudocodice della procedura `PREV(l,i)` che ritorna il riferimento all'elemento che precede l'elemento `i` in una lista singolarmente concatenata di interi (oppure `NULL` se `i` è il primo elemento della lista)

esercizi: liste singolarmente concatenate

5. scrivi lo pseudocodice dell'operazione `DELETE(l,i)` che cancella il nodo `i` di in una lista singolarmente concatenata
   - discuti della complessità dell'algoritmo
6. scrivi lo pseudocodice dell'operazione `DELETE(l,u)` che cancella il nodo che contiene il valore intero `u` in una lista singolarmente concatenata di interi
   - discuti della complessità dell'algoritmo

esercizi: liste singolarmente concatenate

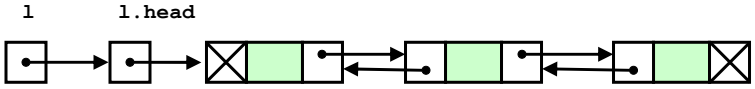
7. implementa una pila di interi utilizzando oggetti e riferimenti
   - devi realizzare le funzioni `CREATE-STACK()`, `IS-EMPTY(p)`, `PUSH(p,u)`, e `POP(p)` facendo uso di oggetti e riferimenti
8. implementa una coda di interi utilizzando oggetti e riferimenti
   - devi realizzare le funzioni `CREATE-QUEUE()`, `IS-EMPTY(c)`, `ENQUEUE(c,u)`, e `DEQUEUE(c)` facendo uso di oggetti e riferimenti

esercizi: liste singolarmente concatenate

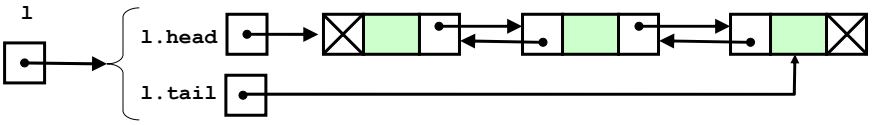
9. scrivi lo pseudocodice della procedura `COMUNI(`l_1`,`l_2`)` che ritorna il numero di elementi della lista l_1 che sono anche contenuti nella lista l_2
   - discuti la complessità dell'algoritmo proposto
10. scrivi lo pseudocodice della procedura non ricorsiva `INVERSA(l)` che ritorna una nuova lista singolarmente concatenata in cui gli elementi sono in ordine inverso
11. scrivi lo pseudocodice della precedura `ACCODA(`l_1`,`l_2`)` che accoda gli elementi della lista l_2 alla lista l_1 mantenendo l'ordine relativo che gli elementi avevano nelle liste originarie
    - puoi supporre di poter modificare le liste in input

lista doppiamente concatenata

- oltre all'attributo `next` i nodi dispongono anche dell'attributo `prev`

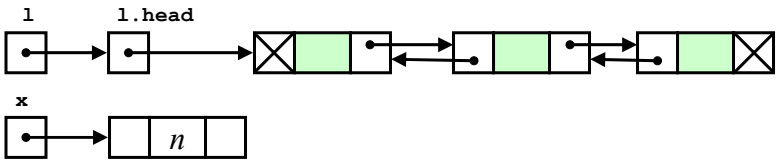


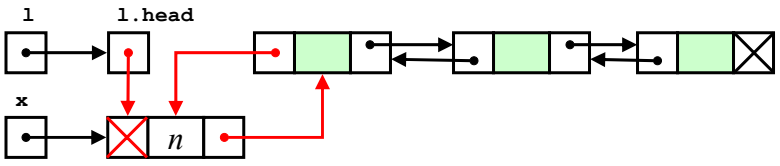
- talvolta la lista `l` dispone anche di un attributo `l.tail`



inserimento nella lista

- `INSERT(l,n)`: inserimento in testa alla lista



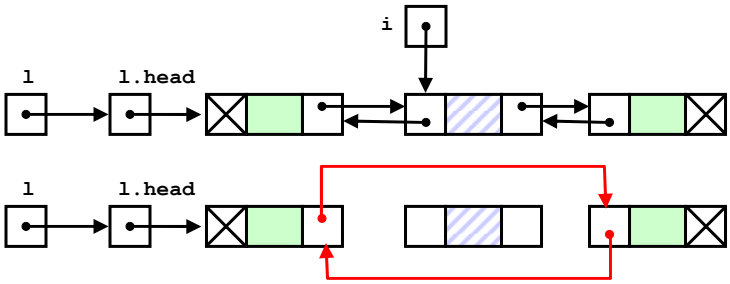


inserimento nella lista

- `INSERT(l,n)`: inserimento in testa alla lista

```
INSERT(l,n)
1.  ▷ x è un nuovo oggetto con tre campi:
2.  ▷      x.info (intero)
3.  ▷      x.prev, x.next (riferimenti ad oggetti analoghi)
4.  x.prev = NULL
5.  x.info = n
6.  x.next = l.head
7.  if l.head != NULL
8.     l.head.prev = x
9.   l.head = x
```


cancellazione di un elemento




```
DELETE(l,i)
1.  if i.prev != NULL
2.      then   i.prev.next = i.next
3.      else   l.head = i.next
4.  if i.next != NULL
5.      then   i.next.prev = i.prev
```


esercizi su liste doppiamente concatenate

12. scrivi lo pseudocodice dell'operazione `INSERT-BEFORE(l,n,i)` che riceva come parametri una lista doppiamente concatenata `l`, un intero `n` ed un iteratore `i`, e inserisca `n` nella lista prima dell'elemento riferito da `i`
    - discuti la complessità della procedura
13. scrivi lo pseudocodice dell'operazione `ADD-AFTER(l,n,i)` che riceva come parametri una lista doppiamente concatenata `l`, un intero `n` ed un iteratore `i`, e inserisca `n` nella lista dopo l'elemento riferito da `i`
    - discuti la complessità della procedura

esercizi su liste doppiamente concatenate

14. implementa una coda utilizzando una lista doppiamente concatenata
    - è possibile che le operazioni `ENQUEUE` e `DEQUEUE` abbiano entrambe complessità $\Theta(1)$?
    - come si potrebbe fare per ottenere questo risultato?
15. scrivi lo pseudocodice della procedura `DELETE(l,u)` che rimuova l'elemento che ha valore `u` da una lista doppiamente concatenata di interi
    - discuti la complessità dell'algoritmo

esercizi su liste doppiamente concatenate

16. scrivi lo pseudocodice della procedura `INSERT_ORDERED(l,u)` che inserisca nella lista `l` (che si suppone ordinata in senso crescente) un intero `u` mantenendo l'ordinamento crescente della lista
17. scrivi lo pseudocodice della procedura `MERGE(`l_1`,` l_2`)` che accetti come parametri due liste doppiamente concatenate di interi ordinate in senso crescente e restituisca una lista ordinata in senso crescente con gli elementi di entrambe
    - puoi supporre che tutti gli elementi delle liste siano diversi

esercizi sulle liste ordinate

18. scrivi lo pseudocodice della procedura `DOPPIONI(l)` che verifichi che una lista (non ordinata) doppiamente concatenata di interi non abbia doppioni
19. scrivi lo pseudocodice della procedura `DOPPIONI-SORTED(l)` che verifichi che una lista doppiamente concatenata di interi ordinata in senso non-decrescente non abbia doppioni

