

ADT Lista

Liste sequenziali

Marco Alberti



Dipartimento
di Matematica
e Informatica



Università
degli Studi
di Ferrara

Programmazione e Laboratorio, A.A. 2020-2021

Ultima modifica: 19 dicembre 2020

Attenzione! Questo materiale didattico è per uso personale dello studente ed è coperto da copyright.
Ne sono vietati la riproduzione e il riutilizzo anche parziale, ai sensi e per gli effetti della legge sul diritto d'autore.

Sommario

1 ADT Lista

2 Implementazione sequenziale

Lista

Rappresenta una sequenza di entità dello stesso tipo (ad esempio interi, caratteri, strutture, altre liste...)

Notazione *astratta* (non sintassi C)

$L = [el1, el2, \dots, elN]$

Esempio

- $['a', 'b', 'c']$ è una lista di caratteri
- $[2, 5, 3]$ è una lista di numeri interi
- $[[2,4], [1], [], [2,5,4]]$ è una lista di liste di interi

$$\{1, 2\} \equiv \{2, 1, 2\}$$

Per sequenza si intende un **multi-insieme finito e ordinato**.

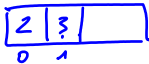
- **multi-insieme**: è possibile che lo stesso elemento compaia più volte ([1, 2] e [2, 1, 2] sono entrambe liste valide (e diverse)).
- **finito**: ad ogni momento, contiene un numero finito di elementi (La successione $a_n = n : n \in \mathcal{N}$ non è una lista). $[1, 2, \quad]$
- **ordinato**: due liste con gli stessi elementi che compaiono in ordine diverso sono diverse (le liste ['a', 'b'] e ['b', 'a'] sono diverse).

$$[2, 3] \neq [3, 2]$$

Dato un tipo di dato T che rappresenta un elemento della lista:

- Il dominio di una **lista di T** è l'insieme dei multi-insiemi finiti e ordinati di elementi di tipo T
- Tipiche operazioni sono
 - **inizializzazione**: creazione di una lista vuota $\rightarrow []$
 - **inserimento** di un elemento di tipo T a una lista di T (in testa, in coda o seguendo altri criteri, come l'ordinamento) $[2, 5] \xrightarrow{ins 1} [1, 2, 5]$
 - **eliminazione** di un elemento di tipo T a una lista di T $[2, 5] \xrightarrow{elim 2} [5]$
 - **svuotamento**: eliminazione di tutti gli elementi
 - **stampa** di tutti gli elementi di una lista
 - **ricerca** di un elemento di tipo T in una lista di T
- Tipici predicati verificano
 - se la lista è **vuota**, cioè non contiene elementi
 - se la lista è **piena** (la definizione astratta non lo consente, ma l'implementazione concreta può limitare la dimensione)

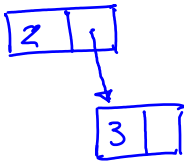
Implementazioni



Le implementazioni delle liste si dividono in due categorie principali:



- Sequenziali (in cui il successore di un elemento della lista è posto nell'area di memoria immediatamente seguente), tipicamente tramite array
- Collegate (in cui il successore di un elemento della lista non è necessariamente fisicamente contiguo, ma si ottiene seguendo un collegamento)
 - tramite array (che non vedremo)
 - tramite puntatori e allocazione/deallocazione dinamica



Sommario

1 ADT Lista

2 Implementazione sequenziale

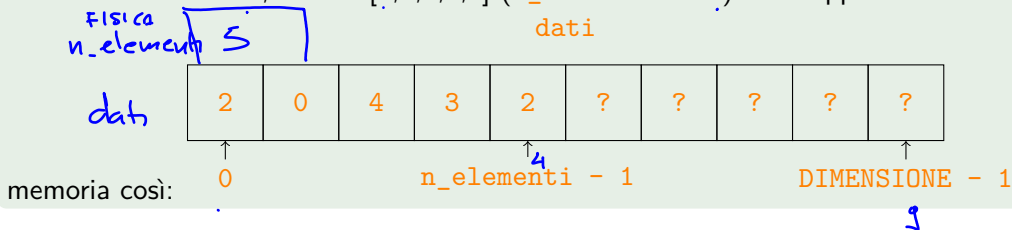
Implementazione sequenziale

Gli elementi sono memorizzati in un vettore di dimensione massima **DIMENSIONE** prefissata.

Lo stato della lista è rappresentato da una struttura con due campi: il vettore **dati** con gli elementi e la dimensione logica **n_elementi** della lista.

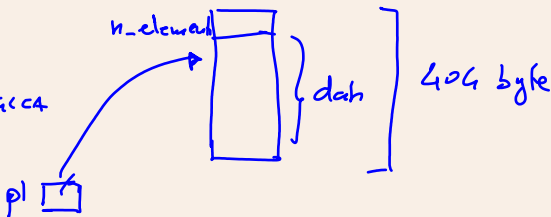
Esempio

Se **DIMENSIONE** è 10, la lista [2,0,4,3,2] (**n_elementi** = 5) sarà rappresentata in



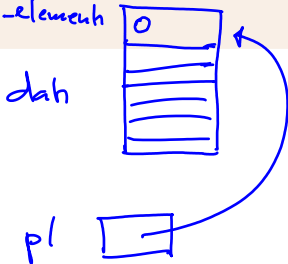
210_liste_sequenziali/lista-sequenziale.h

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #define DIMENSIONE 100
4
5 typedef struct {
6     int n_elementi; DIMENSIONE LOGICA
7     int dati[DIMENSIONE];
8 } Lista;
9
10 void nuova_lista(Lista* pl);
11 int vuota(Lista l);
12 int piena(Lista l);
13 void insTesta(Lista* pl, int numero);
14 void insOrd(Lista* pl, int numero);
15 int ricerca(Lista l, int numero);
16 void elim1(Lista* pl, int numero);
17 void elimTutti(Lista* pl, int numero);
18 int lunghezza(Lista l);
19 void stampa(Lista l);
```



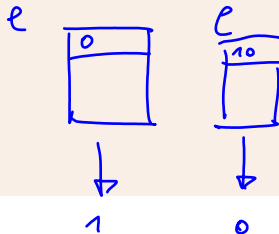
210_liste_sequenziali/nuova-lista.c

```
1 void nuova_lista(Lista* pl) {  
2     // imposto a 0 la dimensione logica della lista  
3     pl->n_elementi = 0;  
4     (*pl).n_elementi = 0;    n-elementi
```



210_liste_sequenziali/vuota.c

```
1 int vuota(Lista l)
2 {
3     // e' vuota se la dimensione logica e' 0
4     return l.n_elementi == 0;
5 }
```

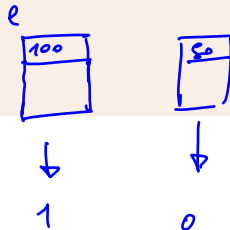


210_liste_sequenziali/piena.c

```

1 int piena(Lista l)
2 {
3     // e' piena se la dimensione logica e' pari alla dimensione
      massima FISCA
4     return l.n_elementi == DIMENSIONE;
5 }

```



210_liste_sequenziali/insTesta.c

```

1 void insTesta(Lista* pl, int numero) {
2     int i;
3     if (piena(*pl)) {
4         // se la lista e' piena non posso inserire elementi
5         printf("Errore: lista piena\n");
6         exit(-1);
7     }
8     // faccio spazio per il nuovo numero spostando gli altri a destra
9     for (i = pl->n_elementi; i > 0; i--)
10         pl->dati[i] = pl->dati[i - 1];
11     // inserisco il nuovo numero
12     pl->dati[0] = numero;
13     // incremento la dimensione logica
14     pl->n_elementi++;
15 }

```



Inserimento ordinato

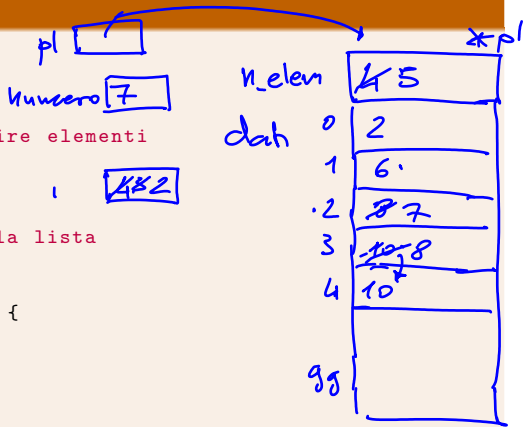
$[2, 6, 8, 10] \xrightarrow{\text{ins ord } 7} [2, 6, 7, 8, 10]$

210_liste_sequenziali/insOrd.c

```

1 void insOrd(Lista* pl, int numero) {
2     int i;
3     if (piena(*pl)) {
4         // se la lista e' piena non posso inserire elementi
5         printf("Errore: lista piena\n");
6         exit(-1);
7     }
8     // sposto a destra tutti gli elementi della lista
9     // maggiori del numero da inserire
10    i = pl->n_elementi;
11    while (i > 0 && pl->dati[i - 1] > numero) {
12        pl->dati[i] = pl->dati[i - 1];
13        i--;
14    }
15    // inserisco il numero
16    pl->dati[i] = numero;
17    // incremento la dimensione logica
18    pl->n_elementi++;
19 }

```



$(2, 5, 3)$
 $\xrightarrow{\text{ricerca } 5} 1$
 $\xrightarrow{\text{ricerca } 4} 0$

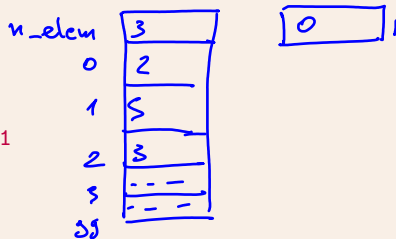
210_liste_sequenziali/ricerca.c

```

1  int ricerca(Lista l, int numero) {
2      int i;
3      // scorro tutti gli elementi della lista
4      for (i = 0; i < l.n_elementi; i++) {
5          // se trovo il numero cercato, ritorno 1
6          if (l.dat[i] == numero)
7              return 1;
8      }
9      // la lista e' finita: ritorno 0
10     return 0;
11 }

```

5
 elemento
 -1



210_liste_sequenziali/elim1.c

```

1 void elim1(Lista* pl, int numero) {
2     int i = 0;
3     while (i < pl->n_elementi) // scorro tutti gli elementi
4     {
5         // se ne trovo uno uguale al numero da eliminare...
6         if (pl->dati[i] == numero) {
7             int j;
8             // sposto a sinistra tutti gli elementi che lo seguono
9             for (j = i; j < pl->n_elementi - 1; j++)
10                 pl->dati[j] = pl->dati[j + 1];
11             pl->n_elementi--; // decremento la dimensione logica
12             return;
13         } else
14             i++;
15     }
16 }

```

Diagram illustrating the state of the list after the first element (2) is found to be equal to the number to be eliminated (5). The list is shown as an array of 5 elements, with the first element (2) highlighted. The number of elements (n-elem) is 5, and the data (dat) is [2, 5, 6, 5, 3].

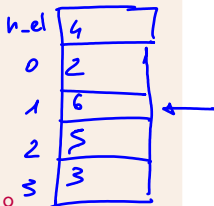
n-elem	dat
5	2
4	5
3	6
2	5
1	3

210_liste_sequenziali/elimTutti.c

```
1 void elimTutti(Lista* pl, int numero) {  
2     int i = 0;  
3     while (i < pl->n_elementi) // scorro tutti gli elementi  
4     {  
5         // se ne trovo uno uguale al numero da eliminare...  
6         if (pl->dati[i] == numero) {  
7             int j;  
8             // spostato a sinistra tutti gli elementi che lo seguono  
9             for (j = i; j < pl->n_elementi - 1; j++)  
10                 pl->dati[j] = pl->dati[j + 1];  
11             pl->n_elementi--; // decremento la dimensione logica  
12         } else  
13             i++;  
14     }  
15 }
```

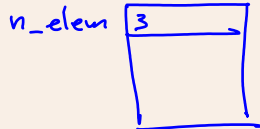
i 1

h-el		4
0		2
1		6
2		5
3		3



210_liste_sequenziali/lunghezza.c

```
1 int lunghezza(Lista l) {  
2     // Corrisponde alla dimensione logica della lista  
3     return l.n_elementi;  
4 }  
    lunghezza
```



210_liste_sequenziali/stampa.c

```

1 void stampa(Lista l) {
2     int i;
3     // stampo tutti gli elementi, separati da spazi
4     for (i = 0; i < l.n_elementi; i++) {
5         printf("%d ", l.dat[i]);
6     }
7     // vado a capo
8     printf("\n");
9 }

```

i 2 5

n_elem

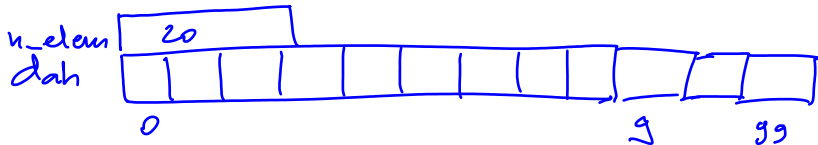
dat

2 5 3 ↴

l	
	3
0	2
1	5
2	3
	...

Limiti della rappresentazione ad array *SEQUENZIALE*

- La dimensione massima è fissata (la lista non può crescere oltre il limite prefissato)
- Anche una lista con pochi elementi occupa memoria quanto una lista della dimensione massima
- Inserimento ed eliminazione onerosi (gli elementi successivi devono "scalare" a destra o a sinistra, rispettivamente)



Lista ordinata

Scrivere un programma che richieda all'utente un massimo di 50 numeri positivi, fermandosi quando l'utente scrive un numero minore o uguale a 0, e stampi i numeri positivi inseriti in ordine crescente.

Per memorizzare i numeri inseriti, definire un ADT lista sequenziale in un file `lista.c`. Il programma principale (nel file `main.c`) dovrà riferirsi alla lista solo attraverso le operazioni disponibili nell'header `lista.h`, contenente la definizione dei tipi di dato e la dichiarazione delle funzioni condivise.

INPUT

5 3 6 2 -1

OUTPUT

2 3 5 6

.!1

[5]

[3,5]

[3,5,6]

[2,3,5,6]