

# Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Liste concatenate di interi\*

## 1 Inserimento e ricerca

Considerate il codice contenuto nel file `list.c`. La funzione `list_insert` inserisce un nuovo elemento all'inizio di una lista; la funzione `list_search` cerca un elemento all'interno della lista. La lista è individuata dall'indirizzo del suo primo elemento.

---

```
1 struct node {
2     int info;
3     struct node *next;
4 };
5
6 struct node *list_insert( int n, struct node *l ){
7     struct node *new = malloc( ... );
8     new -> info = n;
9     new -> next = l;
10    return new;
11 }
12
13 struct node *list_search( int n, struct node *l ){
14     while ( l != NULL && l -> info != n )
15         l = l -> next;
16     return l;
17 }
```

---

Analizzate il codice e rispondete per iscritto alle seguenti domande.

1. Completate la riga 7 dimensionando opportunamente lo spazio richiesto tramite la `malloc`.
2. Usate la `typedef` per definire un nuovo tipo `Node`, puntatore a `struct node`. Riscrivete le righe 6, 7 e 13 usando il nuovo tipo `Node`.
3. Cosa succede quando la funzione `list_insert` viene invocata con `NULL` come secondo argomento?
4. Senza eseguirlo al computer, tracciate l'esecuzione della funzione `list_search` quando riceve come argomenti il numero 7 e l'indirizzo del terzo nodo della lista nella figura sotto.



5. Riscrivete la funzione `list_search` usando un ciclo `for` invece che un ciclo `while`.
6. Cosa restituisce la funzione `list_search` se il valore passato come primo argomento è contenuto in più di un nodo?
7. Scrivete una funzione ricorsiva `list_search_rec` che cerchi un elemento all'interno di una lista.

---

\*Ultima modifica 1 novembre 2020

8. Notate che l'istruzione alla riga 10 è necessaria poiché la funzione modifica la testa della lista; pertanto quando si invoca `list_insert` è necessario aggiornare la testa della lista con un assegnamento del tipo
- ```
list = list_insert( 3, list );
```
- Riscrivete la funzione `list_insert` in modo che modifichi direttamente il puntatore alla testa della lista passata come argomento. Per farlo è opportuno passare tale puntatore per riferimento, quindi il secondo argomento passato alla funzione deve essere l'indirizzo del puntatore in cui è memorizzato l'indirizzo del primo elemento della lista.

## 2 Cancellazione

Considerate il codice contenuto nel file `delete.c`. La funzione `list_delete` dovrebbe cancellare dalla lista un elemento specificato, ma presenta dei problemi.

---

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void list_delete( int n, struct node *l){
5     struct node *curr, *prev;
6     for ( curr = l, prev = NULL; curr != NULL; prev = curr, curr = curr -> next ) {
7         if (curr -> info == n ) break;
8     }
9     if (curr == NULL)
10        return;
11    if (prev == NULL)
12        l = l -> next;
13    else
14        prev -> next = curr -> next;
15    free(curr);
16 }
17
18 int main() {
19     struct node *first = NULL;
20     int n = ...
21     ...
22     list_delete( n , first );
23     list_print( first );
24     return 0;
25 }
```

---

Analizzate il codice e rispondete per iscritto alle seguenti domande.

1. Quando viene eseguita la riga 10? Fornite un esempio specifico.
2. Quando viene eseguita la riga 12? Fornite un esempio specifico.
3. Quando viene eseguita la riga 14? Fornite un esempio specifico.
4. Commentate la funzione `list_delete` in modo da documentare quando vengono eseguiti i vari casi.
5. Considerate gli esempi che avete scritto prima: in quale caso l'invocazione della funzione `list_delete` e della successiva stampa nelle righe 22 e 23 non produce l'effetto desiderato?
6. Modificate la funzione `list_delete` e la sua invocazione nel `main` in modo da eliminare il problema rilevato sopra.

## 3 Altre funzioni

1. Scrivete una funzione `void list_print( Node l )` che stampa gli elementi di una lista (individuata dall'indirizzo del suo primo elemento).
2. Scrivete una funzione ricorsiva `void list_printInv( Node l )` che stampa gli elementi della lista al contrario. Nel passo iterativo, la funzione dovrà prima richiamare sé stessa sul prossimo nodo e poi concludersi stampando il valore del nodo corrente.

3. Scrivete una funzione che, data una lista, costruisce un array con gli elementi della lista e ne restituisce l'indirizzo; l'array va allocato dinamicamente.
4. Scrivete una funzione `int* listToArray( Node l )` che, data una lista `l` contenente `n` interi, restituisce l'indirizzo di un array di interi allocato dinamicamente contenente gli elementi della lista.
5. Scrivete una funzione `void list_destroy ( Node l )` che cancelli tutti gli elementi della lista e liberi con `free` lo spazio che era occupato dai nodi. Provate a scrivere la funzione sia in forma iterativa che in forma ricorsiva. Perché la funzione è di tipo `void`?

## 4 Lista ordinata

1. Scrivete una funzione `struct node *olist_insert( int n, struct node *l )` che inserisca un elemento di valore `n` nella lista ordinata `l`. Per trovare la posizione corretta in cui inserire `n` potete scorrere la lista fintanto che il nodo corrente contiene un valore minore di quello di `n`.
2. Scrivete una funzione `struct node *olist_search( int n, struct node *l )` che inserisca un elemento di valore `n` nella lista ordinata `l`. In quali casi si può interrompere la scansione della lista prima di arrivare all'ultimo elemento?

## 5 Insiemi di interi

In questo esercizio usiamo una lista concatenata per rappresentare un insieme di interi che varia nel tempo (dinamico). L'ordine in cui gli elementi di un insieme compaiono nella lista è irrilevante e gli elementi della lista saranno tutti distinti.

**NOTA BENE:** Questa implementazione non è particolarmente efficiente, ne vedremo di migliori più avanti nel corso.

Dovete scrivere un programma che legga da standard input una sequenza di istruzioni secondo il formato nella tabella, dove `n` è un intero. I vari elementi sulla riga sono separati da uno o più spazi. Quando una riga è letta, viene eseguita l'operazione associata; le operazioni di stampa sono effettuate sullo standard output, e ogni operazione deve iniziare su una nuova riga.

| Istruzione in input | Operazione                                                                                         |
|---------------------|----------------------------------------------------------------------------------------------------|
| <code>+ n</code>    | Se <code>n</code> non appartenga all'insieme lo inserisce, altrimenti non compie alcuna operazione |
| <code>- n</code>    | Se <code>n</code> appartiene all'insieme lo elimina, altrimenti non compie alcuna operazione       |
| <code>? n</code>    | Stampa un messaggio che dichiara se <code>n</code> appartiene all'insieme                          |
| <code>c</code>      | Stampa il numero di elementi dell'insieme                                                          |
| <code>p</code>      | Stampa gli elementi dell'insieme (non importa l'ordine)                                            |
| <code>d</code>      | Cancella tutti gli elementi dell'insieme                                                           |
| <code>f</code>      | Termina l'esecuzione                                                                               |

Si assume che l'input sia inserito correttamente. Conviene scrivere le istruzioni di input in un file `in.txt` ed eseguire il programma reindirigendo lo standard input.

Nel `main()` va definita una variabile `head` di tipo `Node` che indica l'inizio della lista che rappresenta l'insieme. Quindi:

- Se `head` vale `NULL`, `head` rappresenta la lista vuota.
- Se `head` è diverso da `NULL`, `head` è l'indirizzo al primo elemento (testa) della lista.

Strutturate il `main` con un ciclo in cui ad ogni iterata esamini una nuova riga ed esegua l'istruzione corrispondente. Per distinguere le varie istruzioni, usate il costrutto `switch`.

Per contare gli elementi, anziché scrivere una funzione che determina la lunghezza della lista conviene definire nel `main()` una variabile contatore `count` il cui valore è il numero di elementi nell'insieme. Tale contatore va aggiornato ogni volta che si inseriscono o cancellano elementi dall'insieme.

## Variante

Il contatore degli elementi e la testa della lista che rappresenta l'insieme possono essere raggruppati in una `struct` che può essere usata per definire un nuovo tipo `Set`:

---

```
typedef struct {  
    Node head;  
    int count  
} Set;
```

---

Invece di dichiarare `Node head` bisognerà quindi dichiarare una variabile `Set *s`.

Naturalmente, vanno modificate tutte le funzioni precedenti: al posto del parametro `Node l`, servirà un parametro `Set *s`; inoltre per accedere alla testa della lista o al numero dei suoi elementi, dovrete scrivere `s -> head` e `s -> count`, rispettivamente.