

Laboratorio 8 — Lista di Task con Operazioni Predefinite

Assignment — todo_list.c

Università di Catania — Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

Programmazione 1 e Laboratorio (Canale O-Z)

Descrizione del Programma

Si scriva un programma in linguaggio C, chiamato `todo_list.c`, che gestisca una **lista di task** utilizzando una **lista semplicemente concatenata** allocata dinamicamente nello heap.

Ogni task rappresenta un'attività da svolgere ed è caratterizzata da una descrizione testuale e da un flag che indica se l'attività è stata completata o meno.

Il programma legge l'input da **standard input** in tre fasi:

1. Il numero di task iniziali N .
2. Le N descrizioni dei task iniziali (una per riga).
3. Il numero di operazioni M da applicare alla lista, seguito da M righe che rappresentano operazioni di modifica.

Al termine dell'elaborazione, il programma stampa su standard output lo stato finale della lista di task.

Rappresentazione dei Dati

La lista deve essere implementata tramite nodi collegati, secondo la seguente definizione:

```
typedef struct Task {  
    char descrizione[128];  
    int  completato;  
    struct Task *prossimo;  
} Task;
```

```
typedef Task* ListaTask;
```

La lista vuota è rappresentata dal puntatore `NULL`.

Formato dell'Input

Il programma deve leggere da `stdin` il numero di task iniziali `N` con $1 \leq N \leq 100$. Se il valore è fuori intervallo, stampare:

`Errore: numero di task iniziali non valido.`

e terminare con exit status 1.

Le descrizioni iniziali devono essere lette con `fgets` e non possono essere vuote. Se una descrizione è vuota, stampare:

`Errore: descrizione task non valida.`

e terminare con exit status 1.

Dopo aver costruito la lista iniziale, il programma deve leggere il numero di operazioni `M` con $0 \leq M \leq 100$. Se il valore non è valido, stampare:

`Errore: numero di operazioni non valido.`

e terminare con exit status 1.

Le operazioni possibili sono `DONE k`, `DEL k` e `ADD testo del task`. Le operazioni non riconosciute devono essere ignorate senza messaggi di errore.

Dettaglio delle Operazioni sulla Lista

Il programma deve implementare le funzioni di utilità:

- `inserisci_coda`
- `trova_task_per_indice`
- `elimina_task_per_indice`
- `libera_lista`
- `stampa_lista`

Se `malloc` restituisce `NULL`, il programma deve stampare:

`Errore: memoria insufficiente.`

e terminare con exit status 1.

La stampa finale deve usare il formato:

```
[ ] descrizione task non completato
[X] descrizione task completato
```

Al termine dell'elaborazione il programma deve liberare tutta la memoria utilizzata dalla lista e terminare con exit status 0.

Output di Controllo (Esempi di Esecuzione)

Caso 1 — Operazioni di completamento e cancellazione

Input da standard input:

```
3
Compilare il programma
Scrivere la relazione
Inviare email al docente
4
DONE 1
ADD Stampare le slide
DEL 2
DONE 3
```

Output atteso su standard output:

```
[X] Compilare il programma
[ ] Inviare email al docente
[X] Stampare le slide
```

(Exit code atteso: 0)

Caso 2 — Indici fuori range e nessuna operazione

Input da standard input:

```
2
Task A
Task B
3
DONE 3
DEL 0
DONE -1
```

Output atteso:

```
[ ] Task A
[ ] Task B
```

(Exit code atteso: 0)

Caso 3 — Descrizione iniziale vuota

Input:

```
1
0
```

Il programma deve stampare:

Errore: descrizione task non valida.

(Exit code atteso: 1)