UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dispense del Corso di Laboratorio di Fondamenti di Informatica II e Lab

Esercitazione 07: Liste

Ultimo aggiornamento: 17/04/2019

Introduzione

- Date le seguenti definizioni di tipi:

```
typedef int element;
typedef struct list_element {
      element value;
      struct list_element *next;
} item;
typedef item* list;
```

- N.B. normalmente non è una buona idea (anzi è proprio pessima!) mascherare dei puntatori dietro ad alias, come viene fatto in questo caso con la definizione del tipo `typedef item* list`. Durante il corso utilizzeremo comunque questa sintassi a scopo puramente didattico.

Introduzione

- E date le implementazioni delle seguenti primitive:
  - `list EmptyList();`
  - `list Cons(const element e, list l);`
  - `bool IsEmpty(list l);`
  - `element Head(list l);`
  - `list Tail(list l);`
  - `element Copy(const element e);`

- Trovate le dichiarazioni e le rispettive implementazioni delle primitive sopra elencate nei file `list_int.h` e `list_int.c` su dolly. Al link https://vivappz.appspot.com/liste_int_8h.html trovate anche la loro documentazione provvisoria, che vi aiuterà nella risoluzione degli esercizi.

Liste: Carica Lista da File 1/2

- Esercizio 1 (Load):

Nel file `liste.c` si implementi la definizione della funzione `LoadList`:

```
list LoadList(const char *filename);
```

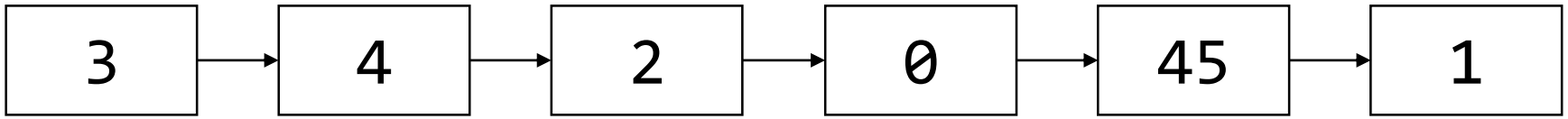
La funzione prende in input un nome di file, `filename`. Il file (di testo) contiene numeri interi separati da spazi. La funzione deve aprire il file in modalità lettura, leggere i numeri e aggiungerli ad una lista. La lista deve essere costruita utilizzando la primitiva `Cons`. La funzione deve quindi restituire la lista così costruita. Se non è possibile aprire il file o se il file è vuoto la funzione deve ritornare una lista vuota.

Liste: Carica Lista da File 2/2

- Si utilizzi opportunamente il debugger per verificare il funzionamento della funzione `LoadList`.
- Dato ad esempio il file `data_00.txt` (disponibile su dolly) contenente i valori:

```
1 45 0 2 4 3
```

la lista ritornata dalla funzione `LoadList` dovrà essere:



Liste: Dealloca Lista

- Esercizio 2 (FreeList):

Nel file `liste.c` si implementi la definizione della funzione `FreeList`:

```
void FreeList(list l);
```

La funzione prende in input una lista e deve deallocare la memoria occupata dai suoi elementi. L'implementazione può fare uso delle primitive.

Si utilizzi opportunamente il debugger per verificare se la funzione `FreeList` libera correttamente la memoria.

Liste: Intersezione fra Liste 1/2

- Esercizio 3 (Intersect):

Nel file `liste.c` si implementi la definizione della funzione `Intersect`:

```
list Intersect(list l1, list l2)
```

La funzione prende in input due liste e restituisce una terza lista contenente tutti e soli i valori presenti sia in l1 che in l2. La funzione può fare uso delle primitive.

Nel `main()` si testi la funzione `Intersect` caricado da file due liste (dovreste già aver implementato la funzione `LoadList`). Si utilizzi il debugger per verificare che la lista ritornata sia corretta.

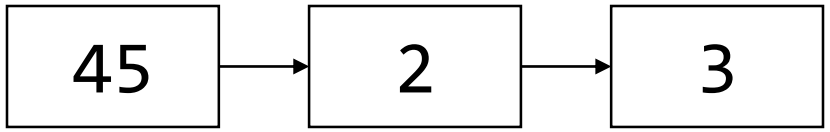
Liste: Intersezione fra Liste 2/2

- Se ad esempio la funzione viene testata con le liste costruite a partire dai seguenti file:

  data_00.txt: `1 45 0 2 4 3`

  data_01.txt: `7 45 3 2 5 8`

  la funzione `Intersect()` dovrà ritorna una lista con i seguenti elementi:



N.B. l'ordine degli elementi potrebbe variare.

Liste: Differenza fra Liste 1/2

- Esercizio 4 (Diff):

Nel file `liste.c` si implementi la definizione della funzione `Diff` :

```
list Diff(list l1, list l2)
```

La funzione prende in input due liste e restituisce una terza lista contenente tutti i valori presenti in l1 che non compaiono in l2. La lista risultante è quindi `l1 – l2`. La funzione `Diff` **non** può fare uso delle primitive.

Nel `main()` si testi la funzione `Diff` caricando da file due liste (dovreste già aver implementato la funzione `LoadList`). Si utilizzi il debugger per verificare che la lista ritornata sia corretta.

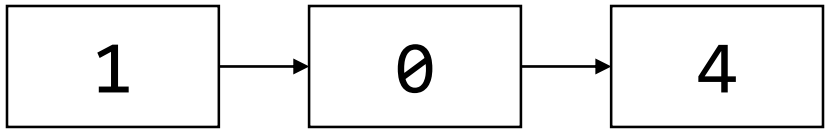
Liste: Differenza fra Liste 2/2

- Se ad esempio la funzione viene testata con le liste costruite a partire dai seguenti file:

  data_00.txt: `1 45 0 2 4 3`

  data_01.txt: `7 45 3 2 5 8`

  la funzione `Diff` dovrà ritorna la lista:



Liste: Varianti di Intersect e Diff

- Esercizio 5 :

Nel file `liste.c` si implementino le definizioni delle funzioni `IntersectNoRep` e `DiffNoRep`:

```
list IntersectNoRep(list l1, list l2);
list DiffNoRep(list l1, list l2);
```

Le funzioni devono avere lo stesso comportamento di `Intersect` e `Diff` descritte negli esercizi precedenti con la differenza che le liste ritornate ***non*** devono contenere valori ripetuti.

Liste: Elemento Massimo

- Esercizio 6 (Diff):

Nel file `liste.c` si implementi la definizione della funzione `MaxElement`:

```
element MaxElement(list l1)
```

La funzione prende in input una lista e restituisce l'elemento di valore massimo.

Nel `main()` si testi la funzione `MaxElement` caricado da file una lista (dovreste già aver implementato la funzione `LoadList`). Si utilizzi il debugger per verificare che l'elemento ritornato sia corretto.