UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

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

Federico Bolelli, Silvia Cascianelli

Esercitazione 07: Liste

Ultimo aggiornamento: 14/05/2020

Introduzione

- Siano date le seguenti definizioni:

```
struct ElemType{
  int *data;
  size_t size;
};
typedef struct ElemType ElemType;

struct Item
{
    ElemType value;
    struct Item *next;
};
typedef struct Item Item;
```

Introduzione

- Siano date le implementazioni delle seguenti funzioni specifiche per la creazione, eliminazione, confronto, acquisizione da file e scrittura su file di `ElemType` di tipo `int`:

```
- int ElemCompare(const ElemType *e1, const ElemType *e2);
- ElemType ElemCopy(const ElemType *e);
- void ElemDelete(ElemType *e);
- ElemType ReadElem(FILE *f);
- ElemType ReadStdinElem();
- void WriteElem(const ElemType *e, FILE *f);
- void WriteStdoutElem(const ElemType *e);
```

Introduzione

- Siano inoltre date le implementazioni delle seguenti funzioni primitive (e non):

```
- Item* CreateEmptyList(void);
- Item* InsertHeadList(const ElemType *e, Item* i);
- bool IsEmptyList(const Item *i);
- const ElemType* GetHeadValueList(const Item *i);
- Item* GetTailList(const Item* i);
- Item* InsertBackList(Item* i, const ElemType *e);
- void DeleteList(Item* item);
- void WriteList(const Item *i, FILE *f);
- void WriteStdoutList(const Item *i);
```

Introduzione

- Trovate le dichiarazione e le definizione dei tipi di dato e delle funzioni descritte nelle slide precedenti nel repository GitHub al link: https://github.com/prittt/Fondamenti-II
- Leggete attentamente il README, il quale vi spiega come scaricare lo zip contenente i file `list_int.h` e `list_int.c` con le implementazione che vi serviranno per l'esercitazione, e dove trovare la documentazione delle suddette funzioni.
- Consultate la documentazione prima di utilizzare una funzione di cui non conoscete con esattezza il comportamento.
- Attenzione: non dovete implementare voi le funzioni primitive, ma dovete utilizzare quelle che vi vengono fornite.

Liste: Carica Lista da File

- Esercizio 1 (Load):

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

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

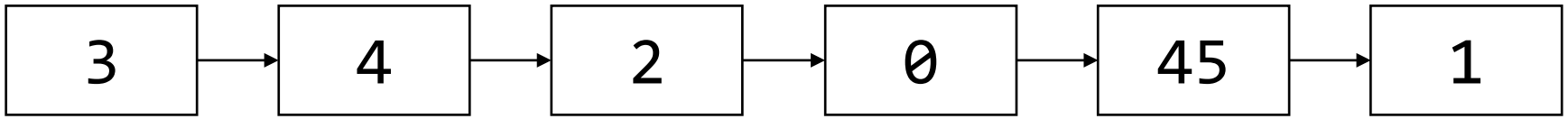
La funzione prende in input un nome di file, `filename`. Il file (di testo) contiene numeri interi separati da spazi. La funzione deve aprire il file in modalità lettura, leggere i numeri e aggiungerli ad una lista. La lista deve essere costruita utilizzando la primitiva di inserimento in testa. La funzione deve quindi restituire la lista (puntatore alla testa) 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

- 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: Intersezione fra Liste

- Esercizio 2 (Intersect):

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

```
Item* Intersect(const Item* i1, const Item* i2)
```

La funzione prende in input due liste (puntatori alla testa) e restituisce una terza lista (puntatore alla testa) contenente tutti e soli i valori presenti in entrambe le liste di input. La lista di ritorno deve essere creata da zero. La funzione può fare uso delle primitive. Le liste di input possono essere liste vuote.

Nel `main()` si testi la funzione `Intersect` caricando da file due liste. Si utilizzi il debugger per verificare che la lista ritornata sia corretta.

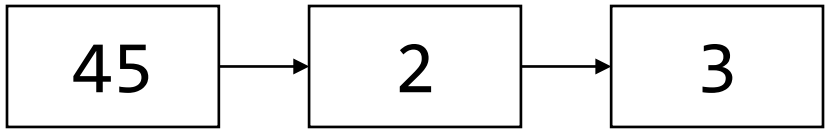
Liste: Intersezione fra Liste

- 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 a seconda della primitiva usata per costruire la lista.

Liste: Differenza fra Liste

- Esercizio 3 (Diff):

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

```
Item* Diff(const Item* i1, const Item* i2)
```

La funzione prende in input due liste (puntatori alla testa) e restituisce una terza lista (puntatore alla testa) costruita da zero contenente tutti i valori presenti in `i1` che non compaiono in `i2`. La lista risultante è quindi `i1 - i2`. La funzione `Diff` **non** deve fare uso delle primitive. Le liste di input possono essere liste vuote.

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.

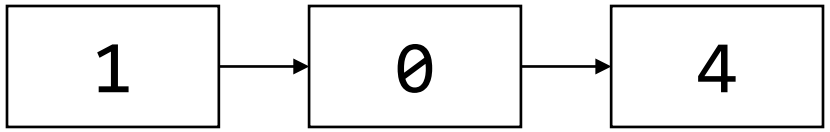
Liste: Differenza fra Liste

- 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_02.txt: 7 45 3 2 5`

  la funzione `Diff` dovrà ritornare la lista:



N.B. l'ordine degli elementi potrebbe variare a seconda della primitiva usata per costruire la lista.

Liste: Varianti di Intersect e Diff

- Esercizi 4 e 5:

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

```
Item* IntersectNoRep(const Item* i1, const Item* i2);
Item* DiffNoRep(const Item* i1, const Item* i2);
```

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 (MaxElement):

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

```
ElemType MaxElement(const Item* i)
```

La funzione prende in input una lista (puntatore alla testa) e restituisce l'elemento di valore massimo. Si assuma che la lista non sia mai vuota.

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.

Modalità di Consegna

- Per questa esercitazione dovrete consegnare gli esercizi 1, 2, 3 e 6 utilizzando il sistema di sottomissione online.
- **Collegatevi al sito https://aimagelab.ing.unimore.it/OLJ2/esami e fate il login utilizzando le vostre credenziali shibboleth di UNIMORE.**
- Selezionate *Esercitazione Liste I,* aprite il link dell'esercizio di cui volete fare la sottomissione e incollate il codice nei box relativi ai rispettivi file. **Non dovete caricare il `main()`.**
- Assicuratevi tuttavia di scrivere il `main()` in Visual Studio per verificare se quello che avete fatto funziona, prima di caricare la soluzione!
- Quando necessario, il codice che sottomettete dovrà opportunamente includere il file delle primitive (`#include "list_int.h"`).
- **Non caricate sul sistema l'implementazione delle primitive.**

Alcune Considerazioni

- Normalmente vi invitiamo a considerare errore qualsiasi *warning* vi venga fornito dal compilatore. In questo particolare caso però, se il sistema di sottomissione vi riporta *warning* del tipo:

```
list_int.h(84): warning #2135: Static 'GetHeadValueList' is not referenced.
list_int.h(63): warning #2135: Static 'WriteStdoutElem' is not referenced.
list_int.h(55): warning #2135: Static 'ReadStdinElem' is not referenced.
```

- Potete ignorarli, in quanto vi stanno semplicemente avvisando del fatto che una particolare funzione primitiva non è stata utilizza.
- Non è obbligatorio usare in ogni programma tutte le primitive.