Esercitazione di Laboratorio del 07/05/2021: Liste

Esercizio 1

Nel file `load.c` implementare la definizione della funzione:

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

La funzione prende in input il nome di un file di testo, `filename`, contenente numeri interi separati da `<whitespace>`. La funzione deve aprire il file in modalità lettura, leggere i numeri e aggiungerli in testa ad una lista opportunamente allocata. Gli elementi devono essere aggiunti nell'ordine in cui vengono letti da file. La funzione ritorna la lista (puntatore alla testa) così costruita.

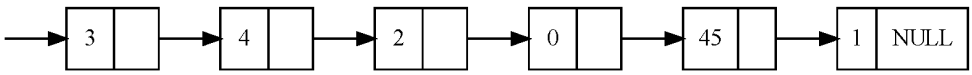
Se non è possibile aprire il file o se il file è vuoto la funzione deve ritornare una lista vuota.

Si utilizzi opportunamente il *debugger* per verificare il funzionamento della funzione `ListLoad()`.

Dato, ad esempio, il file `data_00.txt` contenente i valori:

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

La funzione deve costruire e ritornare la lista:



o, utilizzando la notazione basata su `[]`, `l = [3, 4, 2, 0, 45, 1]`

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

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

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);
```

```
Item *ListCreateEmpty(void);
Item *ListInsertHead(const ElemType *e, Item* i);
bool ListIsEmpty(const Item *i);
const ElemType *ListGetHeadValue(const Item *i);
Item *ListGetTail(const Item *i);
Item *ListInsertBack(Item *i, const ElemType *e);
void ListDelete(Item *item);
void ListWrite(const Item *i, FILE *f);
void ListWriteStdout(const Item *i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `list.h` e `list.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `load.c`

Esercizio 2

Nel file `intersect.c` implementare la definizione della seguente funzione:

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

La funzione prende in input due liste (puntatori alla testa), eventualmente vuote, e ritorna una terza lista (puntatore alla testa) contenente gli elementi di `i1` (nello stesso ordine) i cui valori sono presenti anche in `i2`. La lista di ritorno deve essere creata da zero.

Si utilizzi il *debugger* per verificare che la lista ritornata sia corretta.

Date, ad esempio, le liste:

```
i1 = [3, 4, 2, 0, 45, 1, 3]
i2 = [8, 5, 2, 3, 45, 7]
```

la funzione `Intersect()` dovrà ritorna la lista:

```
r = [3, 2, 45, 3]
```

Suggerimento. Per testare la funzione `Intersect()` è possibile caricare due liste da file utilizzando la funzione definita per l'Esercizio 1.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

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

```
};
typedef struct Item Item;
```

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);

Item *ListCreateEmpty(void);
Item *ListInsertHead(const ElemType *e, Item* i);
bool ListIsEmpty(const Item *i);
const ElemType *ListGetHeadValue(const Item *i);
Item *ListGetTail(const Item *i);
Item *ListInsertBack(Item *i, const ElemType *e);
void ListDelete(Item *item);
void ListWrite(const Item *i, FILE *f);
void ListWriteStdout(const Item *i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `list.h` e `list.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `intersect.c`

Esercizio 3

Nel file `diff.c` implementare la definizione della seguente funzione:

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

La funzione prende in input due liste (puntatori alla testa), eventualmente vuote, e restituisce una terza lista (puntatore alla testa) costruita da zero e contenente tutti i valori presenti in `i1` (nello stesso ordine) che non compaiono in `i2` (`i1 - i2`).

Date, ad esempio, le liste:

```
i1 = [1, 45, 0, 2, 4, 1, 3]
i2 = [7, 45, 3, 2, 5]
```

la funzione `Diff()` dovrà ritornare la lista:

```
r = [1, 0, 4, 1]
```

Suggerimento. Per testare la funzione `Diff()` è possibile caricare due liste da file utilizzando la funzione definita per l'Esercizio 1.

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```
typedef int ElemType;

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

e le seguenti funzioni primitive e non:

```
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);

Item *ListCreateEmpty(void);
Item *ListInsertHead(const ElemType *e, Item* i);
bool ListIsEmpty(const Item *i);
const ElemType *ListGetHeadValue(const Item *i);
Item *ListGetTail(const Item *i);
Item *ListInsertBack(Item *i, const ElemType *e);
void ListDelete(Item *item);
void ListWrite(const Item *i, FILE *f);
void ListWriteStdout(const Item *i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `list.h` e `list.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `diff.c`

Esercizio 4

Nel file `no_rep.c` implementare 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 rispettivamente nell'Esercizio 2 e 3 con la differenza che le liste ritornate non devono contenere valori ripetuti.

Ad esempio, date le liste:

```
i1 = [3, 4, 2, 0, 45, 1, 3]
i2 = [8, 5, 2, 3, 45, 7]
```

la funzione `IntersectNoRep()` dovrà ritornare la lista:

```
r = [3, 2, 45]
```

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```c
typedef int ElemType;

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

e le seguenti funzioni primitive e non:

```c
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);

Item *ListCreateEmpty(void);
Item *ListInsertHead(const ElemType *e, Item* i);
bool ListIsEmpty(const Item *i);
const ElemType *ListGetHeadValue(const Item *i);
Item *ListGetTail(const Item *i);
Item *ListInsertBack(Item *i, const ElemType *e);
void ListDelete(Item *item);
void ListWrite(const Item *i, FILE *f);
void ListWriteStdout(const Item *i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `list.h` e `list.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `no_rep.c`

Esercizio 5

Nel file `max.c` implementare la definizione della seguente funzione:

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

La funzione prende in input una lista (puntatore alla testa) e restituisce l'indirizzo dell'elemento di valore massimo (prima occorrenza). Se la lista è vuota la funzione ritorna `NULL`.

Qual è la complessità dell'algoritmo? Se la lista fosse ordinata in senso crescente potrei fare meglio? E se fosse ordinata in senso decrescente?

Per la risoluzione di questo esercizio avete a disposizione le seguenti definizioni:

```c
typedef int ElemType;

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

e le seguenti funzioni primitive e non:

```c
int ElemCompare(const ElemType *e1, const ElemType *e2);
ElemType ElemCopy(const ElemType *e);
void ElemDelete(ElemType *e);
int ElemRead(FILE *f, ElemType *e);
int ElemReadStdin(ElemType *e);
void ElemWrite(const ElemType *e, FILE *f);
void ElemWriteStdout(const ElemType *e);

Item *ListCreateEmpty(void);
Item *ListInsertHead(const ElemType *e, Item* i);
bool ListIsEmpty(const Item *i);
const ElemType *ListGetHeadValue(const Item *i);
Item *ListGetTail(const Item *i);
Item *ListInsertBack(Item *i, const ElemType *e);
void ListDelete(Item *item);
void ListWrite(const Item *i, FILE *f);
void ListWriteStdout(const Item *i);
```

Trovate le definizioni, le dichiarazioni e le rispettive implementazioni nei file `elemtype.h`, `elemtype.c`, `list.h` e `list.c` scaricabili da OLJ, così come la loro documentazione.

Su OLJ dovete sottomettere solamente il file `max.c`