Esame di Laboratorio del 08/07/2021

Note importanti:

- È considerato errore qualsiasi output non richiesto dagli esercizi.
- È consentito utilizzare funzioni ausiliarie per risolvere gli esercizi.
- Quando caricate il codice sul sistema assicuratevi che siano presenti tutte le direttive di include necessarie, comprese quelle per l'utilizzo delle primitive. Non dovete caricare l'implementazione delle primitive.
- È importante sviluppare il codice in Visual Studio (o altri IDE) prima del caricamento sul sistema, così da poter effettuare il debug delle funzioni realizzate!
- Su OLJ non sarà possibile eseguire più di una compilazione/test ogni 3 minuti.

Esercizio 1

Scrivere un programma a linea di comando con la seguente sintassi:

```
isprime <n>
```

Il programma prende in input un numero intero positivo `n` e deve stampare a video (`stdout`) la sequenza di caratteri `true` se il numero è primo, la sequenza di caratteri `false` altrimenti. Il programma deve usare una funzione **ricorsiva** per determinare se `n` è primo oppure no.

Esempi:

```
n = 0  -> false
n = 1  -> false
n = 4  -> false
n = 7  -> true
n = 9  -> false
n = 11 -> true
```

Se il numero dei parametri passati al programma non è corretto o se `n < 0`, questo termina con codice `1`, senza stampare nulla, altrimenti termina con codice di uscita `0` dopo aver completato la stampa.

Non saranno considerate valide soluzione che non fanno uso della ricorsione per stabilire se `n` è primo.

Esercizio 2

Una pulce salterina si trova in posizione 0 del piano cartesiano e il suo nido è situato lungo l'asse x, a distanza h dall'origine.

La pulce deve raggiungere il nido spostandosi lungo l'asse x secondo le seguenti regole:

- La pulce può saltare in avanti (verso destra lungo l'asse x) di esattamente `a` posizioni;
- La pulce può saltare all'indietro (verso sinistra lungo l'asse x) di esattamente `b` posizioni;

- La pulce **non** può saltare all'indietro due volte consecutivamente;
- La pulce **non** può saltare in posizioni proibite o in posizioni negative dell'asse x;
- La pulce può saltare in posizioni successive al nido ($> h$), ovvero può saltare oltre il nido e tornare indietro successivamente;
- La pulce può fare al massimo `n` salti, saltare in avanti di `a` posizioni o indietro di `b` posizioni contano come uno spostamento;

Nel file `pulce.c` implementare la funzione corrispondente alla seguente definizione:

```
extern char *GuidaLaPulce(const int *f, size_t f_size, int a, int b,
                          int n, int h, size_t *ret_size);
```

Dato `f`, un vettore di interi che rappresenta le posizioni proibite, ovvero le posizioni lungo l'asse x in cui la pulce non può mai saltare, la sua dimensione,`f_size`, `a`, `b`,`n` e `h`, la funzione implementa un algoritmo di backtracking che individua la sequenza minima di mosse per permettere alla pulce di raggiungere il nido.

La funzione deve ritornare un vettore di `char` allocato dinamicamente e contenente la soluzione, ovvero la sequenza di spostamenti eseguiti (il carattere `a` corrisponde ad un salto in avanti, `b` ad un salto all'indietro). La funzione deve scrivere la dimensione del vettore ritornato nel parametro di output `ret_size`.

Se non esistono soluzioni la funzione ritorna NULL e scrive 0 in `ret_size`.

Di seguito vengono riportati alcuni esempi:

Esempio 1:

```
input:
f = [12, 4, 7, 1, 15]
a = 3
b = 15
h = 9
n = 5

output:
ret = [a, a, a]
ret_size = 3
```

Esempio 2:

```
f = [8, 3, 16, 6, 12, 20]
a = 15
b = 13
h = 11
n = 10

output:
ret = NULL
ret_size = 0
```

Esempio 3:

```
f = [1, 6, 2, 14, 5, 17, 4]
a = 16
b = 9
h = 7
n = 5

output:
ret = [a, b]
ret_size = 2
```

Esercizio 3

Nel file `rotate.c` definire la funzione corrispondente alla seguente dichiarazione:

```
extern Item *Rotate(Item *i, int n);
```

La funzione prende in input una lista di interi, `i`, ed un numero intero n. La funzione deve **modificare** la lista passata come parametro, ruotandola verso sinistra di n posizioni, e ritornare il puntatore alla nuova testa.

Ad esempio, dati i parametri:

```
i -> 1 -> 3 -> 4 -> 7 -> 8
n = 2
```

la funzione deve ritornare il puntatore alla testa della lista modificata:

```
ret -> 4 -> 7 -> 8 -> 1 -> 3
```

Se in input avessimo invece:

```
i -> 1 -> 3 -> 4 -> 7 -> 8
n = 1
```

la funzione dovrebbe ritornare la lista così modificata:

```
ret -> 3 -> 4 -> 7 -> 8 -> 1
```

La funzione Rotate() **non** deve modificare i *valori* degli item, ma soltanto i puntatori.

Sia len la lunghezza della lista, si può supporre che $0 \leq n < len$.

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
ElemType ElemCopy(const ElemType *e);
void ElemSwap (ElemType *e1, ElemType *e2)
void ElemDelete(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.

Esercizio 4

Dato un array di `ElemType` senza duplicati, `v`, un *albero binario massimo* può essere costruito in maniera ricorsiva mediante il seguente algoritmo:

1. Si crea la radice dell'albero il cui valore, `m`, è il massimo in `v`;
2. Si costruisce ricorsivamente il sottoalbero sinistro utilizzando il sottovettore `vl` contenente gli elementi di `v` a sinistra di `m`;
3. Si costruisce ricorsivamente il sottoalbero destro utilizzando il sottovettore `vr` contenente gli elementi di `v` a destra di `m`;

Nel file `maxbin.c` definire la funzione corrispondente alla seguente dichiarazione:

```c
extern Node *CreateMaxBinTree(const ElemType *v, size_t v_size);
```

La funzione prende in input un vettore di `ElemType`, costruisce e ritorna l'*albero binario massimo* secondo l'algoritmo sopra descritto.

Esempio 1:

```
input:
v = [3,2,1,6,0,5]

output:

      6
     / \
    /   \
   3     5
    \   /
     2 0
      \
       1
```

Esempio 2:

```
input:
v = [3,2,1]

Output:
    3
     \
      2
       \
        1
```

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

```
typedef int ElemType;

struct Node {
    ElemType value;
    struct Node *left;
    struct Node *right;
};
typedef struct Node Node;
```

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);

Node *TreeCreateEmpty(void);
Node *TreeCreateRoot(const ElemType *e, Node *l, Node *r);
bool TreeIsEmpty(const Node *n);
```

```
const ElemType *TreeGetRootValue(const Node *n);
Node *TreeLeft(const Node *n);
Node *TreeRight(const Node *n);
bool TreeIsLeaf(const Node *n);
void TreeDelete(Node *n);

void TreeWritePreOrder(const Node *n, FILE *f);
void TreeWriteStdoutPreOrder(const Node *n);
void TreeWriteInOrder(const Node *n, FILE *f);
void TreeWriteStdoutInOrder(const Node *n);
void TreeWritePostOrder(const Node *n, FILE *f);
void TreeWriteStdoutPostOrder(const Node *n);
```

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

Esercizio 5

Nel file `pancakesort.c` definire la procedura corrispondente alla seguente dichiarazione:

```
extern void PancakeSort(int *v, size_t v_size);
```

Dato un vettore di `int`, `v`, e la sua dimensione, `v_size`, la funzione deve ordinare gli elementi di `v` in senso crescente utilizzando l'algoritmo di *ordinamento delle frittelle* (o dall'inglese *pancake sort*).

L'unica operazione a disposizione per eseguire l'ordinamento è `flip(v, i)`, che inverte gli elementi del vettore `v` dalla posizione 0 alla posizione `i`, estremi inclusi.

As esempio, dato il vettore seguente:

```
     0  1  2  3  4  5  6
v = [1, 2, 3, 4, 5, 6, 7]
```

- la funzione `flip(v, 2)` produrrebbe `v_ = [3, 2, 1, 4, 5, 6, 7]`;
- la funzione `flip(v, 5)` produrrebbe `v_ = [6, 5, 4, 3, 2, 1, 7]`;
- ...

Dato il vettore `v` e la sua dimenzione `v_size`, l'algoritmo di ordinamento utilizza una serie di *pancake flip* per produrre il vettore finale. Inizialmente si imposta `curr_size = v_size`, quindi si procede come segue:

- Si trova l'indice dell'elemento di valore massimo, `m`, in `v`, considerando solo gli elementi da quello di indice `0` a quello di indice `curr_size - 1`;
- Si invoca la funzione `flip(v, m)`;
- Si invoca la funzione `flip(v, curr_size - 1)`;
- Si decrementa `curr_size` di 1.

L'algoritmo termina quando `curr_size` vale 1.