Compito di Programmazione - Bioinformatica

13 settembre 2023 (tempo disponibile: 2 ore)

Esercizio 1 (15 punti, **si consegni** `parentesi.c`)

Una sequenza di caratteri è una *struttura di parentesi* se e solo se: (1) è vuota (cioè ha lunghezza zero), oppure (2) è composta da una parentesi aperta (tonda, quadra o graffa), seguita da una struttura di parentesi, seguita da una parentesi chiusa (corrispondente come tipo a quella aperta). Si noti che una struttura di parentesi ha sempre lunghezza pari. Le seguenti sono tutte strutture di parentesi:

```
((({{{({()})}}})))
[(({}))]

({[{({{[]}})}]})
```

(la terza è una sequenza vuota). Mentre le seguenti sequenze di caratteri non lo sono:

```
{]([[({{}})]])} (la seconda parentesi non e' chiusa dalla penultima)
{$([[({{}})]])} (il secondo carattere non e' una parentesi)
{([[({{}})]])}  (la terza parentesi e' di tipo diverso dalla terzultima)
[               (la prima parentesi non e' mai chiusa)
```

Si completi il seguente file `parentesi.c` (**dove `struttura_di_parentesi` deve essere ricorsiva**):

```c
// riempie arr, lungo length, con una struttura di parentesi casuale lunga
// length; si dia per scontato che length >= 0 e che length sia pari
void init_random(char arr[], int length) {} // COMPLETATE

// determina se arr, lungo length, e' una struttura di parentesi;
// si dia per scontato che length >= 0 e che length sia pari;
int struttura_di_parentesi(char arr[], int length) {} // COMPLETATE RICORS.
```

I file `parentesi.h` e `main_parentesi.c` sono già scritti e completi, non vanno modificati e non vanno consegnati. Si possono aggiungere funzioni ausiliarie dentro `parentesi.c`.

Se tutto è corretto, un esempio di esecuzione di `main_parentesi.c` potrebbe essere:

```
((({{{({()})}}))) si'
[(({}))] si'
[((([{[{}]}])))] si'
({[[(({{}}))]]}) si'
[{((([{}])))}] si'
{[{{{{{()}}}}}]} si'
{(([[({{}})]]))} si'
 si'
[{(({{{([])}})})}] si'
({[{{{{[]}}}}]}) si'
({{{(][([()])]})}}) no
```

Esercizio 2 (15 punti, **si consegni MaxLoc.c**)

Data una lista di interi si dice che un elemento della lista con valore intero n è un massimo locale se non è direttamente preceduto e non è direttamente seguito da un elemento $m \geq n$.

Si modifichi il seguente file `MaxLoc.c` completando opportunamente le funzioni add e isLocMax.

```
#include <stdio.h>
#include <stdlib.h>

struct Lista {
    int val;
    struct Lista * next;
};
typedef struct Lista LL;
LL * add(int, LL* );
int isLocMax(LL* , int );

//funzione che aggiunge un elemento in testa ad una lista di interi.
LL * add(int num, LL* l) {
    // ... a cura delle/i studentesse/i
}

//funzione che verifica se esiste almeno un elemento nella lista h
//con campo valore uguale a n che e' un massimo locale.
//La funzione deve restituire 0 in caso di riposta negativa ed 1 in
//caso di risposta positiva.

int isLocMax(LL* h, int n) {
    // ... a cura delle/i studentesse/i
}

int main(void) {
    int n;

    LL* l0=add(1,NULL);
    LL* l1=add(1,add(1,NULL));
    LL* l2=add(1,add(1,add(3,NULL)));
    LL* l3= add(9,add(1,add(3,add(2,add(6,NULL)))));
    n=1;
    if (isLocMax(l0, n)) {
        printf("%d e' locmax di l0 \n", n);
    } else {
        printf("%d non e' locmax di l0\n", n);
    }
    if (isLocMax(l1, n)) {
        printf("%d e' locmax di l1\n", n);
    } else {
        printf("%d non e' locmax di l1\n", n);
    }
    n = 3;
    if (isLocMax(l2, n)) {
        printf("%d e' locmax di l2\n", n);
```

```
    } else {
        printf("%d non e' locmax di l2\n", n);
    }
    if (isLocMax(l3, n)) {
        printf("%d e' locmax di l3\n", n);
    } else {
        printf("%d non e' locmax di l3 \n", n);
    }
    return 0;
}
```

La funzione main non va modificata.

Se tutto è corretto, l'esecuzione di `MaxLoc.c` stamperà:

```
1 e' locmax di l0
1 non e' locmax di l1
3 e' locmax di l2
3 e' locmax di l3
```