Fondamenti di Informatica T-1 (A.A. 2017/2018) - Ingegneria Informatica

Prof.ssa Mello

Prova Scritta - 13 Settembre 2018 – durata 1h

Totale 12 punti, sufficienza con 7

Compito A

ESERCIZIO 1 (6 punti)

Data una lista `list1` di valori interi, si realizzi una funzione RICORSIVA

```
list sommaLista(list list1);
```

che restituisca una nuova lista di interi in cui l'elemento in posizione k sia dato dalla somma di tutti gli elementi di `list1` in posizione `i >= k` .

Ad esempio se `list1 = [3, 2, 5]` la funzione `sommaLista` applicata a tale lista deve restituire la nuova lista `[10, 7, 5]`, dove `10` è dato dalla somma di `3` e `2` e `5`, `7` è dato dalla somma tra `2` e `5`, 5 è dato dalla somma di `5`.

Se `list1` risulta vuota il risultato sarà lista vuota.

La funzione dovrà essere implementata utilizzando le sole primitive dell'ADT lista; ogni altra funzione utilizzata per risolvere l'esercizio dovrà essere opportunamente specificata dal candidato. Si realizzi inoltre una semplice funzione `main()` di prova che invochi correttamente la funzione `sommaLista` sulla seguente lista: `[1, 3, 7, 1]`, stampando anche il risultato a terminale. Si scrivano le direttive di inclusione necessarie.

ESERCIZIO 2 (2 punti)

Si consideri la seguente grammatica `G` con scopo `S`, simboli non terminali `{S, Y, F, U}` e simboli terminali `{a, t, c, g, -}`.

```
S := gc-U | F
Y := cg | c-F
F := at-g| ta-Y
U := F-U-at | at
```

La stringa **gc-ta-cg-at-g-at-at-at** appartiene al linguaggio di tale grammatica?

In caso affermativo se ne mostri la derivazione left-most.

ESERCIZIO 3 (3 punti)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione? (si motivi opportunamente la risposta data).

```
#include <stdio.h>
#include <stdlib.h>
int check(int* n0, int n1, char c) {
    switch (c) {
    case 'M':
        return (*n0) > n1 ? 0 : 1;
    case 'm':
        return (*n0) < n1 ? 0 : 1;
    }
}

int checkEach(int* p, int l, char c) {
    int* ind = p;
    int i;
    int count = 0;
    for (i = 0; i < l-1; i++) {
        count += check(ind, ind[1], c);
        printf("%d\n", count);
        ind++;
    }
    return;
}

int main(){
    int a[] = {3,5,-2,4};
    checkEach(a,4,'M');
    return 0;
}
```

ESERCIZIO 4 (1 punto)

Si spieghi cosa sono le variabili globali, a cosa servono e come si fanno a dichiarare/definire mostrando un semplice esempio.

.

Soluzioni

ESERCIZIO 1

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

list sommaLista(list list1){

    if (empty(list1)) {
        return emptyList();
    } else {
        return cons(sum(list1), sommaLista(tail(list1)));
    }
}

int sum(list list1){

    if (empty(list1)) {
        return 0;
    } else {
        return head(list1) + sum(tail(list1));
    }


int main() {
    list list1 = emptyList();

    list1 = cons('1',list1);
    list1 = cons('7',list1);
    list1 = cons('3',list1);
    list1 = cons('1',list1);


    list2 = sommaLista(list1);
    while( ! empty(list2) ){
        printf("%d\n", head(list2));
        list2 = tail(list2);
    }
    return 0;
}
```

ESERCIZIO 2

La frase appartiene al linguaggio. In particolare, la si può ottenere tramite la seguente derivazione left-most:
S → gc-U → gc-F-U-at → gc-ta-Y-U-at → gc-ta-cg-U-at →
gc-ta-cg-F-U-at-at→ gc-ta-cg-at-g-U-at-at → gc-ta-cg-at-g-at-at-at

ESERCIZIO 3

Il programma è corretto e l'output prodotto è:

```
1
1
2
```

La funzione `main` inizializza un array `a` e invoca la funzione `checkEach` passando come parametri `a`, l'intero `4` e il carattere '`M`'.

La funzione `checkEach` riceve in input un puntatore a intero `p`, un intero `l` e un carattere `c`. `p` diventa quindi puntatore all'array `a`, mentre `l` assume il valore `4` e `c` il valore '`M`'.
Viene creato un nuovo puntatore a intero `ind` che viene inizializzato come puntatore ad `a`, viene dichiarato un intero `i` e un altro intero `count` viene inizializzato a `0`. All'inizio del ciclo `for`, `i` assume il valore `0`; il ciclo si ripeterà finché `i` sarà minore di `l-1`, ovvero `4-1`, ovvero `3`.
Poiché `0 < 3`, si entra nel ciclo: all'intero `count` è sommato il risultato della funzione `check`. La funzione `check` è invocata passando come parametri il puntatore `ind`, il valore contenuto nella cella di memoria successiva a quella riferita dal puntatore `ind` e il carattere `c`. Quindi in questo caso viene invocata passando come parametri il puntatore alla prima cella dell'array `a`, il valore `5` e '`M`'.

All'interno della funzione `check`, poiché il carattere `c` ha valore '`M`', si entra nel primo caso. Il programma testa se il valore contenuto all'indirizzo `n0` è maggiore del valore `n1`, ovvero se `3` è maggiore di `5`. Poiché ciò non è vero, viene restituito il valore `1`.

All'interno della funzione `checkEach`, la variabile `count` assume quindi il valore `1`, che viene stampato. Successivamente il valore del puntatore `ind` è incrementato, ciò significa che punterà alla seconda cella dell'array `a`. L'intero `i` è incrementato di `1`, assumendo il valore `1`. Poiché `1 < 3`, il ciclo prosegue.
La funzione `check` è invocata passando l'indirizzo alla seconda cella di `a` e il valore `-2`. Poiché il contenuto della cella, `5`, è maggiore di `-2`, viene restituito dalla funzione il valore `0`. Il valore di `count` rimane quindi `1`, che viene stampato. Il valore di `ind` viene incrementato, ora punta alla terza cella di `a`. Il valore di `i` viene incrementato a `2`. Poiché `2 < 3`, il ciclo prosegue.
La funzione `check` è invocata passando l'indirizzo alla terza cella di `a` e il valore `4`. Poiché il contenuto della cella, `-2`, non è maggiore di `4`, viene restituito dalla funzione il valore `1`. Il valore di `count` diventa quindi `2`, che viene stampato. Il valore di `ind` viene incrementato, ora punta alla quarta cella di `a`. Il valore di `i` viene incrementato a `3`; poiché `3 < 3` non è vero, il ciclo termina.
Dopodiché il programma termina.