Fondamenti di Informatica T-1 (A.A. 2018/2019) - Ingegneria Informatica

Prof.ssa Mello

Prova Scritta – 14 Giugno 2019 – durata 1h

Totale 12 punti, sufficienza con 7

ESERCIZIO 1 (6 punti)

Si scriva una funzione RICORSIVA

```
list listSum(list l, int p)
```

che, data una lista di numeri interi `l`, dia in uscita una lista composta dai numeri dati in ingresso a cui però sia sommato ad ogni elemento il valore di `p`. Se la lista di partenza è vuota la funzione restituirà la lista vuota.

Ad esempio se `l = [2, -3, 5, 1, -6]` e `p = 1`, la funzione `listSum` deve restituire la lista `[3, -2, 6, 2, -5]`.

Si realizzi, poi, una funzione ITERATIVA `totalSum`

```
int totalSum(list l)
```

che data in ingresso la lista prodotta da `listSum` restituisca la somma di tutti i valori contenuti in `l`, supponendo che la lista di partenza contenga almeno un elemento.

Si realizzi inoltre una funzione `main()` che crei una lista `l1`, legga un valore intero `cond` da input e invochi correttamente la funzione `listSum()` in modo che venga creata una nuova lista `l2` composta dagli elementi di `l1` a cui è sommato `cond` se `cond` è pari (0 compreso), sottratto `cond` se `cond` è dispari. Ad esempio se `l1 = [2, -3, 5, 1, -6]`, `cond` è pari =2, deve restituire la lista `l2 = [4, -1, 7, 3, -4]`; se `l1 = [2, -3, 5, 1, -6]`, `cond` è dispari =1, deve restituire `l = [1, -4, 4, 0, -7]`.

Invochi poi la funzione `totalSum()` sulla nuova lista `l2` e ne stampi il risultato.

Le funzioni dovranno essere implementate utilizzando le sole primitive dell'ADT lista, includendo `"list.h"`; ogni altra funzione dovrà essere opportunamente specificata dal candidato

ESERCIZIO 2 (2 punti)

Un elaboratore rappresenta i numeri interi su 8 bit tramite la notazione in complemento a 2. Indicare come viene svolta la seguente operazione aritmetica calcolandone il risultato secondo la rappresentazione binaria in complemento a 2 (si trasli anche il risultato in decimale per verificare la correttezza dell'operazione):
85 - 110

ESERCIZIO 3 (3 punti)

Il seguente programma C compila correttamente? In caso affermativo, quali sono i valori stampati a tempo di esecuzione? (Riportare il risultato stampato, **ben evidenziato**, e motivare opportunamente la risposta data).

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

void crazy(char* c, int k){
    switch (*(c+k)) {
        case 111:
            c[k] -= 32;
            break;
    }
    return;
}

void pet(char* a, int* b, int n, char cond){
    int i;
    for(i=0; i<n; ++i){
            *(a+n-i-1) += b[i];
            if(cond){
                crazy(a, n-i-1);
            }
    }
    return;
}

int main(){
    int i;
    char x[] = {'c', 'a', 't'};
    int y[] = {-16, 14, 4};

    pet(x, y, 3, 'a');

    for(i = 3; i >0; --i){
        printf("%c", x[i-1]);
    }
    return 0;
}
```

ESERCIZIO 4 (1 punto)

Si introduca il concetto di record d'attivazione di una funzione e se ne spieghi la struttura, anche con l'aiuto di esempi.

Soluzioni

ESERCIZIO 1

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

list listSum(list l, int p){
      if(empty(l))
            return emptyList();
      else
            return cons(head(l)+p, listSum(tail(l), p));
}

int totalSum(list l){
      int  sum = 0;

      while(!empty(l)){
            sum += head(l);
            l = tail(l);
      }
      return sum;
}

int main(){
      list l1, l2;
      int sum, cond;

      l1 = cons(2, cons(-3, cons(5, cons(1, cons(-6, emptyList())))));
      scanf("%d", &cond);

      if((cond%2)==0)
            l2=listSum(l1, cond);
      else
            l2=listSum(l1, -cond);

      sum = totalSum(l2);

    printf("SUM: %d\n", sum);

      return 0;
}
```

ESERCIZIO 2

85 - 110 = -25
85 → **01010101**
110 → 01101110
-110 → **10010010**
85 - 110 → **11100111**
-11100111 = 00011001 → 25

ESERCIZIO 3

Il programma è corretto e l'output prodotto è:

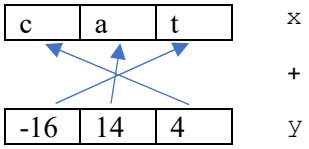
```
d
O
g
```

La funzione `main` inizializza l'array di caratteri `x` con i valori `{'c', 'a', 't'}` e l'array di interi `y` con i valori `{-16, 14, 4}`. Poi invoca la funzione `pet` passando come parametri `x, y`, l'intero `3` ed il carattere `'a'`.

La funzione `pet` riceve in input un puntatore a carattere `a`, un puntatore a intero `b`, un intero `n` ed un carattere `cond`. `a` diventa quindi puntatore all'array `x`, `b` diventa puntatore all'array `y`, `n` assume il valore `3`, `cond` contiene il carattere `'a'`.
All'interno della funzione `pet` viene creata una variabile intera `i`. La funzione `pet` esegue poi un ciclo `for`. All'inizio `i` assume il valore `0`; il ciclo si ripete finché `i` è minore di `n`. Ad ogni iterazione, il valore contenuto nella posizione `n-i-1` del vettore puntato da `a` viene incrementato del valore contenuto nella posizione `i` del vettore puntato da `b` (`a` viene analizzato quindi dall'ultimo elemento al primo, `b` dal primo all'ultimo); inoltre, poiché `cond` è maggiore di `0` (`'a'` equivale a `97`), la condizione logica è sempre `true` e, ad ogni iterazione, viene anche invocata la funzione `crazy` passando come parametri il vettore `a` e l'intero `n-i-1`.

La funzione `crazy` riceve come input un puntatore a carattere `c` ed un intero `k`. `c` diventa quindi puntatore all'array `x` (in quanto `a` a sua volta è un puntatore all'array `x`) e `k` assume il valore `3`.
All'interno della funzione `crazy`, se il valore contenuto in `c` in posizione `k` è uguale ad `'o'` (111), allora si decrementa tale valore di `32` (si cambia da minuscolo a maiuscolo).

Prima parte di `pet`:



Quindi, prima `crazy`, `x` è: `{103, 111, 100}`

g	o	d

Dopo `crazy`, `x` è: `{103, 79, 100}`

g	O	d

Infine, la funzione `main` stampa il contenuto dell'array `x`.

Dopodiché il programma termina.