Corso di Fondamenti di Programmazione - canali A-D e E-O
aa. 2008/2009

Homework 5, consegnare entro le 24:00 del 23/01/2009

Esercizio 1 [**Separa Liste**].
Definire un tipo STRUCT `Nodo` esattamente come segue:

```
typedef struct nodo{
  int val;
  struct nodo * succ;
} Nodo;
```

Scrivere una funzione in linguaggio C con prototipo:

```
void separa_liste( Nodo ** origine, int k, Nodo ** multipli_e_divisori,
                                                          Nodo ** resto);
```

che presi in input una lista `* origine` con elementi di tipo `Nodo` (per lista si intende un puntatore al primo elemento di tipo `Nodo` che sarà poi collegato al successivo, e così via fino all'ultimo elemento che contiene un puntatore a `NULL` come terminatore), un intero `k`, e altre due liste con elementi di tipo `Nodo`, divide la lista `* origine` in due liste. Notate che tutte le liste in input vengono passate per RIFERIMENTO.

- La prima lista, che deve essere puntata da `* multipli_e_divisori`, deve contenere gli elementi che hanno nel campo `val` un valore che sia multiplo o divisore di `k`.
- La seconda lista, che deve essere puntata da `* resto`, deve contenere i valori rimanenti (che non sono pertanto né multipli né divisori del valore `k`).

Il valore puntato da `* origine` al termine della chiamata di funzione deve essere `NULL`, per indicare che la lista `* origine` non è più utilizzabile.

Attenzione: la funzione `separa_liste` (o eventuali funzioni ausiliarie da essa invocate) NON deve allocare nuova memoria ma modificare i campi `succ` degli elementi della lista `* origine` per costruire le due nuove liste.

Attenzione: tutti gli elementi devono essere inseriti nelle liste rispettando l'ordinamento non decrescente per il campo `val`.

Scrivere quindi un programma completo che, presa in input una sequenza di interi non negativi, terminata dal valore -1, e il valore `k`, popoli una lista di elementi di tipo `Nodo` (ricordate che la lista deve essere ordinata in maniera non decrescente) e richiami la funzione

`separa_liste`. Al termine della chiamata di funzione, il programma deve stampare i valori dei campi `val` degli elementi della lista `* multipli_e_divisori` e successivamente i valori dei campi `val` degli elementi della lista `* resto`. I valori devono essere separati da UNO ed UN SOLO spazio o da UN carattere di nuova linea.

ESEMPIO: se l'input è

```
5 15 8 40 24 0 19 30 -1 10
```

`k` vale 10, la lista `origine` contiene nell'ordine i valori

```
0 5 8 15 19 24 30 40
```

Dopo la chiamata alla funzione `separa_liste`, la lista `* origine` sarà vuota (puntatore a `NULL`, mentre le liste `* multipli_e_divisori` e `* resto` conterranno rispettivamente i nodi con i valori

```
5 30 40

0 8 15 19 24
```

L'output del programma deve quindi essere

```
5 30 40
0 8 15 19 24
```

Esercizio 2 [Pseudo tabella hash].

Definire un tipo STRUCT `Nodo` esattamente come segue:

```
typedef struct nodo{
  int chiave;
  double val;
  struct nodo * succ;
} Nodo;
```

Scrivere le funzioni in linguaggio C con i seguenti prototipi:

```
void inserisci( Nodo * tabella [10], int c, double v);

void elimina( Nodo * tabella [10], int c);

Nodo * trova( Nodo * tabella [10], int c);

int max_sbilanciamento( Nodo * tabella[10]);
```

- La funzione `inserisci` prende in input un vettore di liste di elementi di tipo `Nodo`, e una coppia chiave-valore (rispettivamente gli argomenti `c` e `v`) da inserire in una delle liste della tabella. La lista viene selezionata prendendo come indice del vettore `tabella` il valore dato da `c` modulo 10 (ricordiamo che il modulo è il resto della divisione intera `i=c%10`).

  Se nella lista selezionata non è presente alcun elemento con campo `chiave` uguale al parametro `c`, la funzione `inserisci` crea un nuovo elemento di tipo `Nodo` e lo inserisce nella lista. Altrimenti aggiorna il campo `val` dell'elemento con campo `chiave` uguale al valore di `c` con il nuovo valore `v`.

- La funzione `elimina` prende in input un vettore di liste di elementi di tipo `Nodo`, e una chiave `c` da eliminare da una delle liste della tabella. La lista viene selezionata come nella precedente funzione.

  Se nella lista selezionata non è presente alcun elemento con campo `chiave` uguale al parametro `c` , la funzione `elimina` non fa nulla, altrimenti tale elemento viene eliminato dalla lista (ricordarsi di liberare la memoria allocata!).

- La funzione `trova` prende in input un vettore di liste di elementi di tipo `Nodo`, ed una chiave `c` e restituisce un puntatore all'elemento con campo `chiave` uguale a `c`, se esiste, `NULL` altrimenti.

- La funzione `max_sbilanciamento` prende in input un vettore di liste di elementi di tipo `Nodo` e restituisce la differenza fra il numero di elementi nella lista più lunga e il numero di elementi nella lista più corta.

Scrivere quindi un programma completo, che prende in input un intero non negativo `n` e quindi `n` fra coppie di valori `chiave val` (se la chiave è positiva) o semplicemente il valore della chiave se negativa, ed un ulteriore valore intero `x`. Se il valore della chiave è > 0, la coppia chiave-valore deve essere inserita nella tabella tramite la funzione `inserisci`. Se il valore della chiave è < 0, l'elemento con chiave `abs(chiave)` deve essere elimitato (se presente) dalla tabella tramite chiamata alla funzione `elimina` (valori delle chiavi pari a zero non sono ammessi e non verranno mai presentati come input). Il programma deve quindi stampare il massimo sbilanciamento (ottenuto tramite chiamata alla funzione `max_sbilanciamento`), quindi utilizzare il valore `x` per stampare il valore del campo `val` dell'elemento con chiave uguale a `x`, se presente nella tabella, nulla altrimenti.

ESEMPIO: se l'input è

```
8 10 0.43 -1 1 6 2 3.45 -4 -2 14 2.46 4 12.03 14
```

vanno lette 8 chiavi e 5 valori (i valori seguono infatti le chiavi positive che corrispondono agli inserimenti, mentre le chiavi negative non sono seguite da valori e corrispondono a cancellazioni):
$(10, 0.43)(-1)(1, 6)(2, 3.45)(-4)(-2)(14, 2.46)\ (4, 12.03)$.

Al termine degli inserimenti e cancellazioni otteniamo che:
nella lista puntata dall'elemento con indice 0 della tabella, deve essere presente un elemento di tipo `Nodo` con campo `chiave=10` e campo `val=0.43`;
nella lista puntata dall'elemento con indice 1 della tabella, deve essere presente un elemento di tipo `Nodo` con campo `chiave=1` e campo `val=6.00` (visto che l'eliminazione con chiave -1, precedente l'inserimento della coppia (1,6.00) non ha alcun effetto);
nella lista puntata dall'elemento con indice 2 non è presente alcun elemento, visto che l'inserimento della coppia $(2, 3.45)$ viene annullato dalla cancellazione con chiave -2;
nella lista puntata dall'elemento con indice 3 della tabella, non c'è alcun elemento;
nella lista puntata dall'elemento con indice 4 della tabella, devono essere presenti due elementi di tipo `Nodo`, uno con campo `chiave=14` e campo `val=2.46` e l'altro con campo `chiave=4` e campo `val=12.03` (l'ordine fra questi due elementi è arbitrario);
tutte le altre liste sono vuote.

Il programma deve quindi stampare

```
2 2.46
```

infatti il massimo sbilanciamento è pari a 2 (la lista con indice 4 ha 2 elementi e ci sono diverse liste con 0 elementi). Nella tabella è presente un elemento con chiave pari a 14, e il suo valore è 2.46.

Attenzione: i valori in output devono essere separati da UNO ed UN SOLO spazio o UN carattere di nuova linea, il campo `val` deve essere stampato utilizzando 2 cifre decimali.