Homework 2

Chiara Petrioli

Soluzioni dell'Homework 2

Sia data la struttura dati seguente da usare nei 4 esercizi che seguono:

```
struct nodo {
int elem;
struct nodo * next;
};
typedef struct nodo L_ELEM;
typedef L_ELEM * L_PTR;
```

Esercizio 1

Implementare ***ricorsivamente*** la funzione C che ha il prototipo:

L_PTR sum_binaria(L_PTR LX, L_PTR LY,int resto);

e che: riceve in ingresso 2 liste di nodi **LX** e **LY** (di lunghezze anche diverse) **LX** e **LY** contengono i valori dei bit di due numeri binari (un bit per ogni nodo) i bit sono nell'ordine DAL MENO SIGNIFICATIVO al più significativo

e che costruisce e dà come risultato: una lista di nodi contenente i bit del numero binario ottenuto sommando **LX** ed **LY** (sempre in ordine dal bit meno significativo al più significativo).

NOTA le due liste **LX** ed **LY** non debbono essere modificate.

Esempio

resto = 0 LX = 1 -> 1 -> 0 -> 0 -> 1 LY = 1 -> 0 -> 1 -> 1

risultato = 0 -> 0 -> 0 -> 0 -> 0 -> 1

Infatti X = 19 (ovvero 10011 in binario) Y = 13 (ovvero 1101 in binario) somma = 32 (ovvero 100000 in binario)

Esercizio 3

Si implementi ***ricorsivamente*** la funzione C che ha il prototipo:

void inserisciMenoUno (in Nt,int i ,L_PTR *L);

che riceve come argomenti: un puntatore ad un puntatore ad una lista **L** contenente valori interi non negativi un intero **N** positivo (ogni quanti nodi inserire il valore -1) un intero **i** non negativo (che conta a quale nodo siamo arrivati) e che modifica la lista **L** inserendo un nodo contenente il valore **-1** prima di ciascun nodo la cui posizione (iniziando da 0 per il primo nodo) è divisibile per **N**

NOTA la lista **L** deve essere modificata.

Esempio:

*L = 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 N = 3

Risultato: *L = -1 -> 0 -> 1 -> 2 -> -1 -> 3 -> 4 -> 5 -> -1 -> 6 -> 7 -> 8 -> -1 -> 9 -> 10

Soluzioni dell'Homework 2

Esercizio 2

Si implementi ***ricorsivamente*** la funzione C che ha prototipo:

L_PTR selezionaValori(int min,int max,L_PTR L);

che riceve come argomenti una lista **L** di nodi contenenti valori interi e una coppia di interi **min** e **max**

seleziona dalla lista **L** tutti gli elementi il cui valore è compreso tra **min** e **max** (inclusi) torna come risultato la lista degli elementi selezionati

gli elementi devono rimanere nell'ordine che avevano originariamente

NOTA la lista **L** può essere modificata.

Esempio:

L = 1 -> 2 -> 3 -> 7 -> 2 -> -2 -> 9 min = 2 max = 5

Risultato = 2 -> 3 -> 2

Esercizio 4

Si modifichi la funzione dell'esercizio 2 per usare puntatori di puntatori:

L_PTR selezionaValori2(int min, int max, L_PTR * L);

la funzione, oltre a tornare la lista di elementi selezionati deve modificare la lista L in modo che contenga tutti gli altri elementi (nell'ordine originale)

NOTA la lista **L** deve essere modificata.

Esempio:

L = 1 -> 2 -> 3 -> 7 -> 2 -> -2 -> 9 min = 2 max = 5

Risultato = 2 -> 3 -> 2

L = 1 -> 7 -> -2 -> 9

Esercizio 1

```
L_PTR sum_binaria(L_PTR LX, L_PTR LY, int resto)
{
L_PTR temp;
int sum=0;
if ((LX==NULL)&&(LY==NULL)&&(resto==0))
return NULL;
else
{
if (LX!=NULL)
   sum+=LX->elem;
if (LY!=NULL)
   sum+=LY->elem;
temp=malloc(sizeof(L_ELEM));
temp->elem=(sum+resto)%2;
if ((LX!=NULL)&&(LY!=NULL))
   temp->next=sum_binaria(LX->next,LY->next,(sum+resto)/2);
else if (LX!=NULL)
   temp->next=sum_binaria(LX->next,LY,(sum+resto)/2);
else if (LY!=NULL)
   temp->next=sum_binaria(LX,LY->next,(sum+resto)/2);
else
   temp->next=sum_binaria(LX,LY,(sum+resto)/2);
return temp;
}
}
```

Si scriva
Una funzione che
Dati due numeri
binari rappresentati
con liste ne calcoli
e restituisca la
somma

Esercizio 2

```
L_PTR selezionaValori(int min,int max, L_PTR L)
{
L_PTR temp=L;
if (temp==NULL) return NULL;
else if ((temp->elem >=min)&&(temp->elem <=max))
{
                                                    ;
}
else
{
}
}
```

**Si scriva
Una funzione che
Data una lista
La modifichi
Cancellando tutti
Gli elementi il cui
Valore non e' in
Un intervallo dato**

Esercizio 2

```
L_PTR selezionaValori(int min,int max, L_PTR L)
{
L_PTR temp=L;
if (temp==NULL) return NULL;
else if ((temp->elem >=min)&&(temp->elem <=max))
{
temp->next=selezionaValori(min,max,L->next);
return temp;
}
else
{
temp=selezionaValori(min,max,L->next);
free(L);
return temp;
}
}
```

**Si scriva
Una funzione che
Data una lista
La modifichi
Cancellando tutti
Gli elementi il cui
Valore non e' in
Un intervallo dato**

Esercizio 4

```
L_PTR selezionaValori2(int min,int max,L_PTR *L)
{
L_PTR temp=*L;
if (temp==NULL) return NULL;
else if ((temp->elem >=min)&&(temp->elem <=max))
{
*L=(*L)->next;
temp->next=selezionaValori2(min,max,L);
return temp;
}
else
{
temp=selezionaValori2(min,max,&((*L)->next));
return temp;
}
}
```

Si scriva
Una funzione che
Data una lista
Crei due liste
Una contenente gli
Elementi della lista
Con valori in
Un intervallo dato
E l'altra contenente
Tutti gli altri
elementi

Esercizio 3

```
void inserisciMenoUno(int N, int i, L_PTR *L)
{
L_PTR temp;
if (*L!=NULL)
{
if (i==0)
{
[redacted]
}
else
{
[redacted]
}
}

}
```

**Si scriva
Una procedura che
Data una lista
La modifichi
Inserendo un
Elemento con valore
-1 ogni N**

Esercizio 3

```
void inserisciMenoUno(int N, int i, L_PTR *L)
{
L_PTR temp;
if (*L!=NULL)
{
if (i==0)
{
   temp=malloc(sizeof(L_ELEM));
   temp->elem=-1;
   temp->next=*L;
   *L=temp;
   inserisciMenoUno(N,(i+1)%N,&((*L)->next->next));
}
else
{
   inserisciMenoUno(N,(i+1)%N,&((*L)->next));
}
}

}
```

Si scriva
Una procedura che
Data una lista
La modifichi
Inserendo un
Elemento con valore
-1 ogni N