Programmazione I - Laboratorio

Esercitazione 6 - Liste

Gianluca Mezzetti[1] Paolo Milazzo[2]

1. Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼mezzetti
mezzetti@di.unipi.it

2. Dipartimento di Informatica, Università di Pisa
http://www.di.unipi.it/∼milazzo
milazzo@di.unipi.it

Corso di Laurea in Informatica
A.A. 2012/2013

Liste

Usare la seguente definizione negli esercizi seguenti:

```
struct el
{
  int info;
  struct el *next;
}
typedef struct el ElementoDiLista;
typedef ElementoDiLista* ListaDiElementi;
```

Esercizio 1

Scrivere una funzione stampaLista che ricevuta una `ListaDiElementi`, la stampi a video in questo modo: `1 -> 2 -> 3 -> 4 -> //`

Esercizio 1 - esercizio1.c

```
/* esercizio1.c */
#include <stdio.h>
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio1.h"

void stampaLista(ListaDiElementi l){
        if (l==NULL) printf("//\n");
        else
        {
                printf("%d -> ",l->info);
                stampaLista(l->next);
        }
}
```

Esercizio 2

Definire una funzione (fornirne due versioni, una iterativa e una ricorsiva) `lunghezzaLista` che data una `ListaDiElementi`, restituisca la sua lunghezza

Esercizio 2 - esercizio2.c

```
/* esercizio2.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio2.h"

int lunghezzaLista_iterativa(ListaDiElementi l)
{
        int len=0;
        while (l!=NULL)
        {
                l=l->next;
                len++;
        }
        return len;
}

int lunghezzaLista_ricorsiva(ListaDiElementi l)
{
        if (l==NULL) return 0;
        else return lunghezzaLista_ricorsiva(l->next)+1;
}
```

Esercizio 3

Definire una funzione deallocaLista che riceve una `ListaDiElementi` e ne dealloca tutti gli elementi.

Esercizio 3 - esercizio3.c

```c
/* esercizio3.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio3.h"

void deallocaLista(ListaDiElementi l)
{
        if (l!=NULL)
        {
                deallocaLista(l->next);
                free(l);
        }
}
```

Esercizio 4

Definire una funzione (fornirne due versioni, una iterativa e una ricorsiva) `primoPari` che data una `ListaDiElementi`, restituisca il puntatore al primo elemento pari nella lista (restituisce `NULL` se la lista e' vuota o non contiene elementi pari)

Esercizio 4 - esercizio4.c

```c
/* esercizio4.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio4.h"

ListaDiElementi primoPari_iterativo(ListaDiElementi l)
{
  while ((l!=NULL)&&(l->info%2!=0)) /*notare ordine condizioni in and
        l=l->next;
  return l;
}

ListaDiElementi primoPari_ricorsivo(ListaDiElementi l)
{
  if ((l==NULL)||(l->info%2==0)) return l;
  else primoPari_ricorsivo(l->next);
}
```

Esercizio 5

Definire una funzione (fornirne due versioni, una iterativa e una ricorsiva) `minimoPari` che data una `ListaDiInteri`, restituisca il puntatore al minimo elemento pari nella lista (restituisce `NULL` se la lista e' vuota o non contiene elementi pari)

Esercizio 5 - esercizio5.c

```c
/* esercizio5.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio5.h"

ListaDiElementi minimoPari_iterativo(ListaDiElementi l)
{
  ListaDiElementi min=NULL;
  while (l!=NULL)
  {
    /* come primoPari (esercizio4) */
    while ((l!=NULL)&&(l->info%2!=0)) /*notare ordine condizioni in a
        l=l->next;
    if (min==NULL) min=l;
    else if ((l!=NULL)&&(l->info<min->info)) min=l;
    if (l!=NULL) l=l->next;
  }

  return min;
}

ListaDiElementi minimoPari_ricorsivo(ListaDiElementi l)
{
  if (l==NULL) return NULL;
  /* un solo elemento pari */
  else if ((l->next==NULL)&&(l->info%2==0)) return l;
  /* un solo elemento dispari */
  else if (l->next==NULL) return NULL;
  else {
```

Esercizio 6

Definire una procedura (sia iterativa che ricorsiva) 'elimina' che ricevuta una `ListaDiElementi` e un intero `X`, elimini (senza deallocare) i primi `X` elementi e ritorni il puntatore alla testa della lista modificata

Esercizio 6 - esercizio6.c

```c
/* esercizio6.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio6.h"

ListaDiElementi elimina_iterativo(ListaDiElementi l, int X)
{
  while ((l!=NULL)&&(X>0))
  {
    l=l->next;
    X--;
  }
  return l;
}

ListaDiElementi elimina_ricorsivo(ListaDiElementi l, int X)
{
  if (l==NULL||X==0) return l;
  else return elimina_ricorsivo(l->next,X-1);
}
```

Esercizio 7

Definire una funzione ordinaLista che modifica una `ListaDiElementi` data ordinandola in modo crescente. La funzione non deve usare allocazione dinamica della memoria (`malloc` e `free`), ne modificare il campo `info` degli elementi.
La funzione restituisce il puntatore al primo elemento ottenuto dopo l'ordinamento

Esercizio 7 - esercizio7.c I

```
/* esercizio7.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio7.h"

ListaDiElementi ordinaLista(ListaDiElementi l)
{
  ListaDiElementi l1,tmp,prec_tmp;

  if ((l==NULL)||(l->next==NULL)) return l;
  else l1 = ordinaLista(l->next);
  /* l1 e' di sicuro non vuota */

  tmp = l1;
  prec_tmp = l;
  /* ATTENZIONE: l'ordine delle condizioni in && e' importante! */
  while ((tmp!=NULL)&&(tmp->info<l->info))
  {
    prec_tmp=tmp;
    tmp=tmp->next;
  }

  /* devo piazzare il primo elemento di l dentro a l1 */
  if (tmp==NULL) /* inserimento in coda */
  {
    l->next=NULL;
    prec_tmp->next=l;
```

Esercizio 7 - esercizio7.c II

```
    return l1;
  }
  else if (tmp==l1) /* inserimento in testa */
  {
    l->next=l1;
    return l;
  }
  else /* inserimento in mezzo */
  {
    l->next = tmp;
    prec_tmp->next = l;
    return l1;
  }
}
```

Esercizio 8

Definire una funzione `merge` che date due `ListaDiElementi` ordinate, restituisca una nuova `ListaDiElementi` ordinata contenente tutti gli elementi delle due liste. Le liste originali devono restare immutate

Esercizio 8 - esercizio8.c I

```
/* esercizio8.c */
#include <stdlib.h>
#include "definizione_lista.h"
#include "esercizio8.h"

ListaDiElementi merge(ListaDiElementi l1, ListaDiElementi l2)
{
  ListaDiElementi risultato=NULL;
  ListaDiElementi new=NULL;
  int primavolta = 1;

  /* scorre le due liste simultaneamente */
  while ((l1!=NULL)&&(l2!=NULL)) {
    if (primavolta)
    {
      new = (ListaDiElementi) malloc(sizeof(ElementoDiLista));
      risultato = new;
      primavolta=0;
    }
    else
    {
      new->next = (ListaDiElementi) malloc(sizeof(ElementoDiLista));
      new=new->next;
    }
    if (l1->info<l2->info)
      { new->info=l1->info; l1=l1->next; }
    else
```

Esercizio 8 - esercizio8.c II

```
        { new->info=l2->info; l2=l2->next; }
  }

  /* uno (e solo uno) dei seguenti due cicli viene eseguito */
  while (l1!=NULL)
  {
    new->next = (ListaDiElementi) malloc(sizeof(ElementoDiLista));
    new=new->next;
    new->info = l1->info;
    l1 = l1->next;
  }
  while (l2!=NULL)
  {
    new->next = (ListaDiElementi) malloc(sizeof(ElementoDiLista));
    new=new->next;
    new->info = l2->info;
    l2 = l2->next;
  }

  /* mette NULL in fondo alla lista */
  if (new!=NULL) new->next=NULL;

  return risultato;
}
```

Il main di questa esercitazione - main.c I

L'insieme degli esercizi di questa esercitazione e il `main` seguente sono un esempio di programmazione modulare a cui si può applicare la compilazione separata dei moduli

```
/*
  programma che esegue tutti gli esercizi sulle liste di questa lezio
 */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "definizione_lista.h"
#include "esercizio1.h"
#include "esercizio2.h"
#include "esercizio3.h"
#include "esercizio4.h"
#include "esercizio5.h"
#include "esercizio6.h"
#include "esercizio7.h"
#include "esercizio8.h"

ListaDiElementi generaLista();

int main()
{
```

Il main di questa esercitazione - main.c II

```
ListaDiElementi l1,l2,lord1,lord2,ltmp;

srand(time(0));
l1 = generaLista(10);
l2 = generaLista(14);

printf("Esercizio 1:\n");
printf("Lista l1\n");
stampaLista(l1);
printf("Lista l2\n");
stampaLista(l2);
printf("\n");


printf("Esercizio 2:\n");
printf("Lunghezze l1: %d\n",lunghezzaLista_iterativa(l1));
printf("Lunghezze l2: %d\n",lunghezzaLista_ricorsiva(l2));
printf("\n");


printf("Esercizio 3:\n");
ltmp=generaLista(3);
deallocaLista(ltmp);
printf("Niente da visualizzare per questo esercizio\n");
printf("\n");
```

Il main di questa esercitazione - main.c III

```
printf("Esercizio 4:\n");
printf("Primo pari l1: %d\n",primoPari_iterativo(l1)->info);
printf("Primo pari l2: %d\n",primoPari_ricorsivo(l2)->info);
printf("\n");


printf("Esercizio 5:\n");
printf("Minimo pari l1: %d\n",minimoPari_iterativo(l1)->info);
printf("Minimo pari l2: %d\n",minimoPari_ricorsivo(l2)->info);
printf("\n");


printf("Esercizio 6:\n");
printf("Elimina i primi 3 da l1\n");
ltmp=elimina_iterativo(l1,3);
stampaLista(ltmp);
printf("Elimina i primi 3 da l2\n");
ltmp=elimina_ricorsivo(l2,3);
stampaLista(ltmp);
printf("\n");


printf("Esercizio 7:\n");
printf("Ordina l1\n");
lord1=ordinaLista(l1);
stampaLista(lord1);
printf("Ordina l2\n");
```

Il main di questa esercitazione - main.c IV

```
  lord2=ordinaLista(l2);
  stampaLista(lord2);
  printf("\n");


  printf("Esercizio 8:\n");
  printf("Fusione di l1 e l2\n");
  ltmp=merge(lord1,lord2);
  stampaLista(ltmp);
  printf("\n");

  return 0;
}

ListaDiElementi generaLista(int numelem)
{
  if (numelem==0) return NULL;
  else {
    ListaDiElementi new = (ListaDiElementi) malloc(sizeof(ElementoDiL
    new->info=rand()%100;
    new->next=generaLista(numelem-1);
    return new;
  }
}
```