Laboratorio di Programmazione: Linguaggio C
Lezione 23 del 9 giugno 2014

Damiano Macedonio

Esercizio 1 [Tratto dalla prova parziale del 30 maggio 2010]

Si considerino le liste di interi come definite a lezione:

```
struct list {
  int head;
  struct list *tail;
};
```

Si scriva una **funzione ricorsiva** `fusion` che riceve come parametri due liste **ordinate** non decrescenti (cioè due puntatori ad esse) e restituisce una lista ordinata non decrescente che contiene tutti gli elementi delle due liste (cioè un puntatore ad essa). Le due liste parametro non devono venire modificate.

Se tutto è corretto, l'esecuzione del seguente programma:

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

int main() {
  struct list *l1, *l2, *l3;
  char *s1, *s2, *s3;

  printf("Prima lista non descrescente di numeri positivi (<=0 per
      terminare):\n");
  l1 = read();
  printf("Seconda lista non descrescente di numeri positivi (<=0 per
      terminare):\n");
  l2 = read();

  printf("La fusione di %s e %s corrisponde a %s\n",
     s1 = toString(l1),
     s2 = toString(l2),
     s3 = toString(fusion(l1, l2)));

  free(s1); free(s2); free(s3);
  return 0;
}
```

si comporterà ad esempio come segue:

```
Prima lista non descrescente di numeri positivi (<=0 per terminare):
? 3
? 6
? 6
? 8
? 11
? 0
Seconda lista non descrescente di numeri positivi (<=0 per terminare):
? 6
? 11
? 34
? 56
? 134
? 135
? 0
La fusione di [3,6,6,8,11] e [6,11,34,56,134,135] corrisponde a
[3,6,6,6,8,11,11,34,56,134,135]
```

La struttura `list` e la funzione `fusion` devono essere dichiarate nel file `list.h`. Inoltre, in `list.h` devono essere dichiarate anche le seguenti funzioni:

`struct list *construct(int head, struct list *tail)` costruisce una nuova lista che ha come testa il parametro `head` e come coda il parametro `tail` (usare l'allocazione dinamica della memoria).

`struct list *read()` finchè l'utente inserisce un valore maggiore di zero la funzione richiede all'utente un nuovo valore da inserire nella lista (stampando il carattere `?`). La funzione utilizza il metodo `construct` per creare un nuovo elemento della lista da aggiungere alla lista corrente.

`char *toString(struct list *this)` funzione ricorsiva che ritorna una stringa rappresentante la lista passata come parametro. La funzione ha bisogno di due funzioni ausiliarie: `length` che determina la lunghezza della stringa, e `toString_aux` che viene richiamata ricorsivamente sui vari elementi della lista. La stringa risultato viene costruita utilizzando la funzione `int sprintf(char *str, const char *format, ...))` che riceve in ingresso: una stringa `str` in cui salvare la stringa prodotta, una stringa di formato `format` e una serie di parametri che servono a comporre la stringa (es. `sprintf(s, (%d),x)` salva nella stringa `s` il valore della variabile `x` tra parentesi tonde), e ritorna un intero rappresentante il numero di caratteri scritti.

`int length(struct list *this)` funzione ricorsiva che ritorna la lunghezza (numero di elementi) della lista passata come parametro.

Esercizio 2 [Tratto dalla prova parziale del 7 giugno 2011]

Si considerino le liste di interi come definite a lezione:

```
struct list {
  int head;
  struct list *tail;
};
```

Si scriva una **funzione ricorsiva** `remove_all` che riceve come parametro una lista `this` e un numero intero `n` e restituisce una lista uguale a `this` ma privata di tutti gli elementi uguali ad `n`. La lista `this` non deve essere modificata.

Se tutto è corretto, l'esecuzione del seguente programma:

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

int main() {
  struct list *l;
  char *s1, *s2;
  int n;

  printf("Inserisci una lista di numeri positivi (<=0 per terminare)\n");
  l = read();
  printf("Inserisci un numero da eliminare dalla lista: ");
  scanf("%d", &n);
  printf("Eliminando %d da %s ottengo %s\n", n, s1 = toString(l),
        s2 = toString(remove_all(l, n)));
  free(s1); free(s2);
  return 0;
}
```

si comporterà ad esempio come segue:

```
Inserisci una lista di numeri positivi (<=0 per terminare)
? 7
? 11
? 3
? 5
? 8
? 3
? 91
? 9
? 5
? 80
? 0
Inserisci un numero da eliminare dalla lista: 3
Eliminando 3 da [7,11,3,5,8,3,91,9,5,80] ottengo [7,11,5,8,91,9,5,80]
```

Potete usare le funzioni ausiliare definite nell'esercizio precedente.

Esercizio 3 [Tratto dall'esame del 11 luglio 2012]

Si considerino le liste viste a lezione.

```
struct list {
  int head;
  struct list *tail;
};
```

Si implementi una **funzione ricorsiva**:

```
int prefix(struct list *this, struct list *that);
```

che restituisce *vero* se e solo se `that` è un prefisso di `this` (cioè se e solo se `this` comincia con `that`). Si noti che la lista vuota è un prefisso di qualsiasi altra lista.

Se tutto è corretto, l'esecuzione del programma:

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

int main(void) {
  struct list *l =
    construct
    (11, construct
     (13, construct
      (-4, construct
       (13, construct
        (-6, construct
         (0, construct
          (1, NULL)))))));

  struct list *p1 = construct(11, construct(13, NULL));
  struct list *p2 = construct(13, construct(-4, construct(13, NULL)));

  printf("%s ", toString(p1));
  if (!prefix(l, p1)) printf("non ");
  printf("rappresenta un prefisso di %s\n", toString(l) );

  printf( "%s ", toString(p2) );
  if (!prefix(l, p2)) printf("non ");
  printf("rappresenta un prefisso di %s\n", toString(l) );

  return 0;
}
```

dovrà stampare:

```
[11, 13] rappresenta un prefisso di [11, 13, -4, 13, -6, 0, 1]
[13, -4, 13] non rappresenta un prefisso di [11, 13, -4, 13, -6, 0, 1]
```

Potete usare le funzioni ausiliare definite nell'esempio precedente.