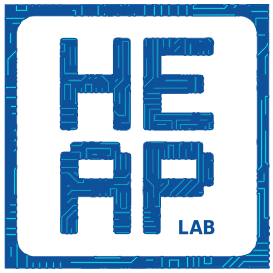
Esercizi su liste in C

Stefano Cherubin*



21/12/2017

[**Informatica A**] Esercitazione #14

corso per Ing. Gestionale a.a. 2017/18

*<nome>.<cognome>@polimi.it

Indice

1 Esercizi di riscaldamento

Sia data la seguente definizione di tipo

```
typedef struct list_elem_s {
  int value;
  struct list_elem_s* next;
} list_elem_t;
```

Si definiscano le seguenti procedure in linguaggio C.

1. `void displayOdd(list_elem_t* list)`
   - visualizza su standard output i soli elementi della lista che occupano posto di posizione dispari (si consideri la testa della lista come posizione 0, quindi pari)

2. `void displayEven(list_elem_t* list)`
   - visualizza su standard output i soli elementi della lista che occupano posto di posizione pari (si consideri la testa della lista come posizione 0, quindi pari). Si fornisca una versione ricorsiva di questa procedura.

3. `void dump(list_elem_t* list, const char* filename)`
   - esegue il dump (copia tutto il contenuto) della lista in un file di testo il cui nome è passato come parametro.

1.1 Soluzione C - displayOdd

Di seguito si presentano due equivalenti versioni. La prima analizza due elementi della lista per ogni iterazione, l'altra conserva un contatore con l'indice della posizione e controlla la parità dell'indice ad ogni iterazione.

Listato 1: displayOdd

```
1  void displayOdd(list_elem_t* list) {
2    /* while (list != NULL) */
3    while (list) {
4      /* salta il primo perché pari */
5      list = list->next;
6      if (list) {
7        printf("%d->", list->value);
8        list = list->next;
9      }
10   }
11   printf("||\n");
12   return;
13 }
```

Listato 2: displayOdd - variante con contatore

```
1  void displayOddCount(list_elem_t* list) {
2    int i = 0;
3    /* while (list != NULL) */
4    while (list) {
5      /* if (i % 2 != 0) */
6      if (i % 2) {
7        printf("%d->", list->value);
8      }
9      list = list->next;
10     ++i;
11   }
12   printf("||\n");
13   return;
14 }
```

1.2 Soluzione C - displayEven

Di seguito si presenta una versione che fa uso di due procedure chiamate ricorsivamente in catena. La prima si occupa di numeri dispari, la seconda di numeri. Solo la seconda contiene l'output su standard output.

Considerato che le due funzioni si chiamano ricorsivamente l'un l'altra, è fondamentale che i prototipi delle stesse vengano anteposti.

Listato 3: displayEven

```
1  void displayOddRec(list_elem_t* list);
2  void displayEvenRec(list_elem_t* list);
3
4  void displayOddRec(list_elem_t* list) {
5    if (list) {
6      list = list->next;
7      displayEvenRec(list);
8    }
9    return;
10 }
11
12 void displayEvenRec(list_elem_t* list) {
13   if (list) {
14     printf("%d->", list->value);
15     list = list->next;
16     displayOddRec(list);
17   }
18   return;
19 }
```

1.3 Soluzione C - dump

La soluzione consta in uno scorrimento della lista, accoppiato con la scrittura del file.

Listato 4: dump

```
1  void dump(list_elem_t* list, const char* filename) {
2    FILE* f;
3    f = fopen(filename, "w");
4    if (f) {
5      while (list) {
6        fprintf(f, "%d\n", list->value);
7        list = list->next;
8      }
9      fclose(f);
10   }
11   return;
12 }
```

2 ADT

Si implementino facendo uso di liste semplici concatenate i seguenti tipi di dato astratto

1. Stack (pila)
   - La cancellazione (pop) può avvenire solo sull'ultimo elemento
   - L'inserimento (push) può avvenire solo dopo l'ultimo elemento
   - L'accesso (top) può avvenire solo sull'ultimo elemento

2. Queue (coda)
   - L'inserimento (push) può avvenire solo dopo l'ultimo elemento
   - La cancellazione (pop) può avvenire solo sul primo elemento
   - L'accesso (top) può avvenire solo sul primo elemento

Si faccia uso del tipo `list_elem_t*` utilizzato nell'esercizio 1.

2.1 Soluzione C - stack

Listato 5: stack

```
1  list_elem_t* stack_pop(list_elem_t* list) {
2    list_elem_t* tmp;
3    tmp = list;
4    if (list) {
5      list = list->next;
6      free(tmp);
7    }
8    return list;
9  }
10
11 list_elem_t* stack_push(list_elem_t* list, list_elem_t
       * elem) {
12   if (elem) {
13     elem->next = list;
14   }
15   return elem;
16 }
17
18 list_elem_t* stack_top(list_elem_t* list) {
19   return list;
20 }
```

2.2 Soluzione C - queue

Listato 6: queue

```
1  list_elem_t* queue_pop(list_elem_t* list) {
2    list_elem_t* tmp;
3    tmp = list;
4    if (list) {
5      list = list->next;
6      free(tmp);
7    }
8    return list;
9  }
10
11 list_elem_t* queue_push(list_elem_t* list, list_elem_t
       * elem) {
12   elem->next = NULL;
13   if (!list) {
14     return elem;
15   }
16   list_elem_t* tmp = list;
17   while (tmp->next) {
18     tmp = tmp->next;
19   }
20   tmp->next = elem;
21   return list;
22 }
23
24 list_elem_t* queue_top(list_elem_t* list) {
25   return list;
26 }
```

3 HTML, quando voi nasceste

Si considerino le strutture dati definiti come segue

```
#define TITLE_LEN 15
#define BODY_LEN 15

typedef struct date_s {
  int dd;
  int mm;
  int yyyy;
} date_t;

typedef struct node_s {
  char title[TITLE_LEN];
  char body[BODY_LEN];
  date_t date;
} node_t;
```

1. Si codifichi in linguaggio C una struttura dati senza modificare le strutture sopra presentate per rappresentare una lista di nodi.
2. Si codifichi in linguaggio C una procedura `dumpToHTML(list_t* l, date_t birthday, char* htmlFile)` che prende in input una lista di nodi, il giorno di nascita di una persona e il nome di un file e scriva su file in linguaggio HTML i soli nodi contenuti nella lista associati ad una data antecedente alla data di riferimento.

   Si ricorda che un documento HTML è un file di testo che inizia con il tag `<html>` e termina con il tag `</html>`. Si compone di due sezioni (head e body). Nella sezione head è possibile specificare dettagli come il titolo della pagina `<head><title>TITOLO PAGINA</title></head>`. Nella sezione body è possibile specificare il contenuto del documento; si suggerisce l'utilizzo dei tag `<h1>TITOLO</h1>` e `<p>CORPO DEL PARAGRAFO</p>` per specificare rispettivamente i titoli e i contenuti testuali da inserire nel documento.

3.1 Soluzione C

Listato 7: dumpToHTML

```
1  int compareDate(date_t lhs, date_t rhs) {
2    if (lhs.yyyy < rhs.yyyy) return -1;
3    if (lhs.yyyy > rhs.yyyy) return +1;
4    if (lhs.mm < rhs.mm) return -1;
5    if (lhs.mm > rhs.mm) return +1;
6    if (lhs.dd < rhs.dd) return -1;
7    if (lhs.dd > rhs.dd) return +1;
8    return 0;
9  }
10
11 void dumpToHTML(my_list_elem_t* list, date_t birthday,
       char* htmlfile) {
12   FILE* f;
13   f = fopen(htmlfile, "w");
14   if (f) {
15     fprintf(f, "<html>\n");
16     fprintf(f, "<head><title>Result page</title></head
         >\n");
17     fprintf(f, "<body>\n");
18     while (list) {
19       if (compareDate(list->val.date, birthday) < 0) {
20         fprintf(f, "<h1>%s\n</h1>", list->val.title);
21         fprintf(f, "<p>%s\n</p>", list->val.body);
22         fprintf(f, "<p>%d %d %d\n</p>", list->val.date
               .dd, list->val.date.mm, list->val.date.yyyy
               );
23       }
24     list = list->next;
25     }
26   fprintf(f, "</body>\n");
27   fprintf(f, "</html>\n");
28   fclose(f);
29   }
30 }
```

4 Statistiche

Si definisca una struttura di lista concatenata semplice dove ciascun nodo contiene un numero in virgola mobile.

Si scriva un programma che acquisisce da tastiera un indeterminato numero di elementi e li inserisca nella lista. Al primo valore nullo inserito dall'utente, l'inserimento termina e si calcoli il valore minimo, massimo e mediano nella lista.

4.1 Soluzione C

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

typedef struct node_s {
  float val;
  struct node_s *next;
} node_t;

int countNodes(node_t *head);
node_t * getElementByPosition(node_t *head, int
    position);
node_t * loadList();
node_t * sortList(node_t *head);

int main() {
  node_t *head;
  node_t *min, *max, *med;
  int numberOfNodes;

  head = loadList();
  if (head == NULL) {
    printf("Lista vuota\n");
    return 0;
  }
  head = sortList(head);
  numberOfNodes = countNodes(head);

  /* min = head */
  min = getElementByPosition(head, 0);
  max = getElementByPosition(head, numberOfNodes - 1);
  med = getElementByPosition(head, numberOfNodes / 2);
  printf("Minimo: %f\tMediano: %f\tMassimo:%f\n", min
      ->val, med->val, max->val);
  return 0;
}

int countNodes(node_t *head) {
  int i;
  for(i = 0; head; head = head->next, i++);
  return i;
}

node_t * getElementByPosition(node_t *head, int
    position) {
```

```
42    int i;
43    if (position < 0) {
44      return NULL;
45    }
46    i = 0;
47    while (i < position && head) {
48      i ++;
49      head = head->next;
50    }
51    return head;
52  }
53
54  node_t * loadList() {
55    node_t *head, *tmp, *aux;
56    float tmp_val;
57    head = NULL;
58    tmp = head;
59    printf("Inserisci un valore nella lista (0 per
          terminare): ");
60    scanf("%f", &tmp_val);
61    while (tmp_val) {
62      tmp = (node_t*) malloc(sizeof(node_t));
63      if (tmp == NULL) {
64        printf("Errore di allocazione memoria.\n");
65        return head;
66      }
67      tmp->val = tmp_val;
68      tmp->next = NULL;
69      if (head) {
70        for (aux = head; aux->next; aux = aux->next);
71        aux->next = tmp;
72      } else {
73        head = tmp;
74      }
75      printf("Inserisci un valore nella lista (0 per
            terminare): ");
76      scanf("%f", &tmp_val);
77    }
78    return head;
79  }
80
81  /* Bubblesort: take head of the list as parameter and
       returns the updated head of the list */
82  node_t * sortList(node_t *head) {
83    int changed;
84    node_t *prev, *prev2, *tmp, *other;
```

```
85    if (head == NULL || head->next == NULL) {
86      return head;
87    }
88    do {
89      changed = 0;       /* termination flag */
90      prev2 = NULL;      /* element before   */
91      prev = head;       /*  first node to compare */
92      tmp = prev->next; /* second node to compare */
93      while (tmp != NULL) {
94        other = tmp->next;
95        if (prev->val > tmp->val) {
96          /* swap nodes */
97          tmp->next = prev;
98          prev->next = other;
99          if (prev == head) {
100           head = tmp;
101         }
102         if (prev2) {
103           prev2->next = tmp;
104         }
105         /* update for next iteration */
106         prev2 = tmp;
107         tmp = other;
108         changed = 1;
109       } else {
110         /* update for next iteration */
111         prev2 = prev;
112         prev = tmp;
113         tmp = other;
114       }
115     }
116   } while (changed);
117   return head;
118 }
```

Licenza e crediti

Licenza beerware[1]

Quest'opera è stata redatta da Stefano Cherubin. Mantenendo questa nota, puoi fare quello che vuoi con quest'opera. Se ci dovessimo incontrare e tu ritenessi che quest'opera lo valga, in cambio puoi offrirmi una birra.

[1] http://people.freebsd.org/~phk/