Университет ИТМО
Кафедра ВТ

Языки системного программирования

Лабораторная работа №4

Группа P3210
Нгу Фыонг Ань
ПРОВЕРИЛ:
Лукьянов Николай Михайлович

2018 год

10.6 Assignment: Linked List

10.6.1 Assignment

The program accepts an arbitrary number of integers through stdin. What you have to do is

1. Save them all in a linked list in reverse order.

2. Write a function to compute the sum of elements in a linked list.

3. Use this function to compute the sum of elements in the saved list.

4. Write a function to output the n-th element of the list. If the list is too short, signal about it.

5. Free the memory allocated for the linked list.

You need to learn to use

- Structural types to encode the linked list itself.
- The EOF constant. Read the section “Return value” of the man scanf.

You can be sure that

- The input does not contain anything but integers separated by whitespaces.
- All input numbers can be contained into int variables.

Following is the recommended list of functions to implement:

- list_create – accepts a number, returns a pointer to the new linked list node.
- list_add_front – accepts a number and a pointer to a pointer to the linked list.

Prepends the new node with a number to the list.

For example: a list (1,2,3), a number 5, and the new list is (5,1,2,3).

- list_add_back, adds an element to the end of the list. The signature is the same as

list_add_front.

- list_get gets an element by index, or returns 0 if the index is outside the list bounds.
- list_free frees the memory allocated to all elements of list.
- list_length accepts a list and computes its length.
- list_node_at accepts a list and an index, returns a pointer to struct list, corresponding to the node at this index. If the index is too big, returns NULL.
- list_sum accepts a list, returns the sum of elements.

11.7.2 Assignment

The input contains an arbitrary number of integers.

1. Save these integers in a linked list.

2. Transfer all functions written in previous assignment into separate .h and c files.

Do not forget to put an include guard!

3. Implement foreach; using it, output the initial list to stdout twice: the first time, separate elements with spaces, the second time output each element on the new line.

4. Implement map; using it, output the squares and the cubes of the numbers from list.

5. Implement foldl; using it, output the sum and the minimal and maximal element in the list.

6. Implement map_mut; using it, output the modules of the input numbers.

7. Implement iterate; using it, create and output the list of the powers of two (first 10 values: 1, 2, 4, 8, …).

8. Implement a function bool save(struct list* lst, const char* filename);, which will write all elements of the list into a text file filename. It should return true in case the write is successful, false otherwise.

9. Implement a function bool load(struct list** lst, const char* filename);, which will read all integers from a text file filename and write the saved list into *lst. It should return true in case the write is successful, false otherwise.

10. Save the list into a text file and load it back using the two functions above. Verify that the save and load are correct.

11. Implement a function bool serialize(struct list* lst, const char* filename);, which will write all elements of the list into a binary file filename. It should return true in case the write is successful, false otherwise.

12. Implement a function bool deserialize(struct list** lst, const char* filename);, which will read all integers from a binary file filename and write the saved list into *lst. It should return true in case the write is successful, false otherwise.

13. Serialize the list into a binary file and load it back using two functions above. Verify that the serialization and deserialization are correct.

14. Free all allocated memory

#Code

#main.c

```
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
#include "lkl.h"

void function(int a){
   printf("%d ",a);    return;
}
void __function(int a){
   printf("%d\n",a);    return;
}
int square(int a){
   return (a*a);
}
int cube(int a){
   return a*a*a;
}
int _func(int x,int a){
   return x+a;
}
int max(int x,int a){
   return ((x>a)?x:a);
}
int min(int x,int a){
   return ((x>a)?a:x);
}
int module(int x){
   return abs(x);
}
int power(int base){
```

```
    return base*2;
}
bool bool_save(struct LinkedList* ll,const char* filename){
    FILE* fw = fopen(filename,"w");
    if (!fw) return false;
    struct Node* newNode = ll->head;
    while (newNode!=NULL){
        fprintf(fw,"%d ",newNode->data);
        newNode = newNode->next;
    }
    fclose(fw);    return true;
}
bool bool_load(struct LinkedList** ll,const char* filename){
    int x;
    FILE* fr = fopen(filename,"r");
    if (fr){
        while (fscanf(fr," %d",&x)== 1){list_add_back(ll,x);}
        fclose(fr);       return true;
    }
    else return false;
}
bool bool_serialize(struct LinkedList* ll,const char* filename){
    FILE* fw = fopen(filename,"wb+");
    if (!fw) return false;
    struct Node* newNode = ll->head;
    while (newNode!=NULL){
        int x = newNode->data;
        fwrite(&x,sizeof(int),1,fw);
        newNode = newNode->next;
    }
    fclose(fw);    return true;


}
```

```
bool bool_deserialize(struct LinkedList** ll,const char* filename){
   int x;
   FILE* fr = fopen(filename,"rb");
   if (fr){
      while (fread(&x,sizeof(int),1,fr)>0){list_add_back(ll,x);}
      fclose(fr);       return true;
   }
   else
   {
      fclose(fr);       return false;
   }
   //fclose(fr);
}
int main() {
   struct LinkedList* ll = (struct LinkedList*) malloc(sizeof(struct LinkedList));
   ll->head = NULL;
   ll->tail = NULL;
   ll->length =0;
   input(ll);

   list_foreach(ll->head,(*function));    puts("");
   printf("The sum of the elements is %d\n", list_sum(ll));

   int in;
   printf("Which position do you want to see? \n");
   scanf(" %d",&in);
   printf("the %d th element is %d\n", in, list_get(ll,in-1) ->data);

   printf("Test map (^2): \n");
   struct LinkedList* result = map(ll,(*square));
   list_foreach(ll->head,(*function));    puts("");
   list_foreach(result->head,(*function));    puts("");
   printf("Test map (^3): \n");
```

```
result = map(ll,(*cube));
list_foreach(ll->head,(*function));    puts("");
list_foreach(result->head,(*function));    puts("");

printf("Test map_mut (^3) : \n");
ll = map_mut(ll,(*cube));
list_foreach(ll->head,(*function));    puts("");

printf("Test foldl : \n");
int (*_fu)(int,int);
_fu= _func;
int tmp = foldl(0,(*_fu),ll);
printf("SUM = %d\n",tmp);
_fu = min;
printf("MIN = %d\n",foldl(2147483647,(*_fu),ll));
_fu = max;
printf("MAX = %d\n",foldl(-2147483647,(*_fu),ll));

struct LinkedList* ll2 = (struct LinkedList*) malloc(sizeof(struct LinkedList));

printf("Test iterate \n");
ll2 = iterate(2,10,(*power));
list_foreach(ll2->head,(*function));    puts("");

printf("Saving list into file txt ...\n");
bool ok = bool_save(ll,"output.txt");
if (ok) {
   printf("List saved to file txt.\n");
} else {
   printf("Error. List is not saved.\n");
}

struct LinkedList* lil = (struct LinkedList*) malloc(sizeof(struct LinkedList));
```

```
    lil->head = NULL;
    lil->tail = NULL;
    lil->length =0;
    bool_load(lil,"output.txt");
    list_foreach(lil->head,(*function));    puts("");

    printf("Saving list into binary file ...\n");
    ok = bool_serialize(ll,"output.bin");
    if (ok) {
       printf("List saved to file binary.\n");
    } else {
       printf("Error. List is not saved.\n");
    }

    struct LinkedList* lil1 = (struct LinkedList*) malloc(sizeof(struct LinkedList));
    lil1->head = NULL;
    lil1->tail = NULL;
    lil1->length =0;
    ok = bool_deserialize(lil1,"output.bin");
    list_foreach(lil1->head,(*function));
    listFree(ll);
    return 0;
}
```

#lkl.c

```
#include <stdio.h>
#include <stdlib.h>
#include "lkl.h"
//make elements

//init newNode
struct Node* initNewNode(int x){
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
```

```
    newNode->data = x;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}
//add element to _back
void list_add_back(struct LinkedList* ll,int x){
  struct Node* newNode = initNewNode(x);
  if (ll->tail == NULL){
      ll->tail = newNode;
      if (ll->head == NULL) ll->head = newNode;
      ll->length ++;
      return;
  }
  (ll->tail)->next = newNode;
  newNode->prev = ll->tail;
  ll->tail = newNode;
  ll->length++;
}
//add element to _front
void list_add_front(struct LinkedList* ll,int x){
    struct Node* newNode = initNewNode(x);
    if (ll->head == NULL){
      ll->head = newNode;
      if (ll->tail == NULL) ll->tail = newNode;
      ll->length++;
      return;
    }
    (ll->head)->prev = newNode;
    newNode->next = ll->head;
    ll->head = newNode;
    ll->length++;
}
```

```
//free list
void listFree(struct LinkedList* ll){
    struct Node* newNode = ll->head;
    struct Node* tmp ;
    while (newNode != ll->tail){
        tmp = newNode->next;
        free(newNode);
        newNode = tmp;
    }
    free(ll->tail);
}
//length of list
int list_length(struct LinkedList* ll){
    int x = ll->length;
    return x;
}
//get n-th element of list
struct Node* list_get(struct LinkedList* ll,int n){
    if (n>=ll->length) return NULL;
    int tmp =0;
    struct Node* res = ll->head;
    while (tmp <= n){
        if (tmp==n) return res;
        tmp++;
        res = res->next;
    }
    return NULL;
}
//sum of elements
int list_sum(struct LinkedList* ll){
    int res = 0;
    if ((ll->head == NULL) && (ll->tail==NULL)) return 0;
    struct Node* tmp = ll->head;
```

```
    if (tmp == ll->tail) return tmp->data;
    do{
        res += tmp->data;
        tmp = tmp->next;
    }
    while (tmp != ll->tail);
    res+= ll->tail->data;
    return res;
}

//for-each
void list_foreach(struct Node* newNode,void (*f)(int)){

    if (newNode->next == NULL) {
        f(newNode->data);
        return;
    }
    do{
        f(newNode->data);
        newNode = newNode->next;
    }
    while (newNode!=NULL);
    return;
}
//map
struct LinkedList* map(struct LinkedList* ll,int (*_f)(int)){
    struct LinkedList* ll2 = (struct LinkedList*) malloc(sizeof(struct LinkedList));
    ll2->head = NULL;
    ll2->tail = NULL;
    ll2->length =0;

    struct Node* newNode = ll->head;
    if (ll->head == ll->tail) {int x = _f(newNode->data);list_add_back(ll2,x);return ll2;}
```

```
    do{
        int x = _f(newNode->data);
        list_add_back(ll2,x);
        newNode = newNode->next;
    }
    while (newNode!=NULL);
    return ll2;
};
//map_mut
struct LinkedList* map_mut(struct LinkedList* ll,int (*_f)(int)){
    struct Node* newNode = ll->head;
    if (ll->head == ll->tail) {
        int x = _f(newNode->data);
        newNode->data = x;
        return ll;
    }
    do{
        int x = _f(newNode->data);
        newNode->data = x;
        newNode = newNode->next;
    }
    while (newNode!=NULL);
    return ll;
};
//foldl
int find_foldl(int res,int (*_f)(int,int),struct Node* newNode){
    if (newNode == NULL ) return res;
    res = find_foldl(_f(res,newNode->data),(*_f),newNode->next);
    return res;
}

int foldl(int res,int (*_f)(int,int),struct LinkedList* ll){
    res = find_foldl(res,(*_f),ll->head);
```

```
    return res;
}
//iterate
struct LinkedList* iterate(int s,int lens,int (*f)(int)){
    struct LinkedList* ll2 = (struct LinkedList*) malloc(sizeof(struct LinkedList));
    ll2->head = NULL;
    ll2->tail = NULL;
    ll2->length =0;
    for(int i=0;i<lens;i++){
        list_add_back(ll2,s);
        s = f(s);
    }
    return ll2;
};
//delete at adress
void list_deleteAt(struct LinkedList* ll,int x){
    int i =0;
    struct Node* newNode = ll->head;
    while (i!=x){
        newNode = newNode->next;
        i++;
    }
    if (newNode==ll->head) {
        ll->head = newNode->next;
        ll->head->prev = NULL;
     }
     else if (newNode == ll->tail){
        ll->tail = newNode->prev;
        ll->tail->next = NULL;
     }
     else {
        newNode->next->prev = newNode->prev;
        newNode->prev->next = newNode->next;
```

```
        free(newNode);
    }
}
//at to ll at index
void list_addAt(struct LinkedList* ll,int index){

}
void input(struct LinkedList* ll){
   printf("Please insert your array:\n");

   int a=1;
   int number;
   while (a!= EOF ){
      a=scanf("%i", &number);
      if (a!=EOF) {
         list_add_front(ll,number);}
       else if (a==0) {
         printf("Wrong input");
         return 0;
      }
   }
}

#ifndef LKL_H
#define LKL_H

struct Node{
   int data;
   struct Node* next;
   struct Node* prev;
};
struct LinkedList{
   struct Node* head;
```

```
    struct Node* tail;
    int length;
};

struct Node* initNewNode(int x);
void list_add_back(struct LinkedList* ll,int x);
void list_add_front(struct LinkedList* ll,int x);
void listFree(struct LinkedList* ll);
void list_deleteAt(struct LinkedList* ll,int x);
int list_length(struct LinkedList* ll);
struct Node* list_get(struct LinkedList* ll,int n);
struct Node* list_get(struct LinkedList* ll,int n);
void list_foreach(struct Node* newNode,void (*f)(int));
struct LinkedList* map(struct LinkedList* ll,int (*_f)(int));
struct LinkedList* map_mut(struct LinkedList* ll,int (*_f)(int));
int find_foldl(int res,int (*_f)(int,int),struct Node* newNode);
int foldl(int res,int (*_f)(int,int),struct LinkedList* ll);
struct LinkedList* iterate(int s,int lens,int (*f)(int));
void input(struct LinkedList* ll);

#endif /* LKL_H */
```