Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики.

Кафедра вычислительной техники

Языки системного программирования

Лабораторная работа №4

Связные списки и функции высших порядков

Выполнил:

Студент группы P3210

Глушков Дмитрий Сергеевич

Санкт-Петербург
2018 г.

Задание 1.

1. Создать связный список, хранящий значений из потока stdin в обратном порядке.
2. Написать функции:
   - 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.
   - 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.

Задание 2.

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.

Текст программы (задание 1):

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

///Types definitions

typedef int data_t;
typedef int index_t;

struct element
{
  data_t data;
  struct element *next;
};
typedef struct element elem_t;


///Functions signatures
int isNull(void *pointer);
elem_t* list_create (data_t value);
int list_add_front(elem_t **begin, data_t value);
int list_add_back(elem_t **begin, data_t value);
elem_t* list_node_at(elem_t *begin, index_t id);
data_t list_get(elem_t *begin, index_t id);
index_t list_lenght(elem_t *begin);
data_t list_sum(elem_t *begin);
void list_free(elem_t *begin);

///Functions definitions

int main()
{
  const index_t n = 5; //Выводимый элемент
  elem_t *begin = NULL;
  data_t value;
  scanf("%d", &value);
  begin = list_create(value);
  while (scanf("%d",&value) != EOF)
    list_add_front(&begin, value);
  printf("Sum: %d\n", list_sum(begin));
  printf("%d element (reverse): %d\n", n, list_get(begin, n));
  list_free(begin);
  return 0;
}



int isNull(void *pointer)
{
  if (pointer) return 0;
  return 1;
}


elem_t* list_create (data_t value)
{
  elem_t *temp = NULL;
  temp = (elem_t*) malloc (sizeof (elem_t));
```

```
  if (isNull(temp)) return 0;
  temp -> next = NULL;
  temp -> data = value;
  return temp;
}


int list_add_front(elem_t **begin, data_t value)
{
  elem_t *temp = NULL;
  temp = (elem_t*) malloc (sizeof (elem_t));
  if (isNull(temp)) return 1;
  temp -> next = *begin;
  temp -> data = value;
  *begin = temp;
  return 0;
}


int list_add_back(elem_t **begin, data_t value)
{
  elem_t *temp = NULL, *p = *begin;
  temp = (elem_t*) malloc (sizeof (elem_t));
  if (isNull(temp)) return 1;
  temp -> next = NULL;
  temp -> data = value;
  while (p -> next != NULL)
    p = p -> next;
  p -> next = temp;
  return 0;
}


elem_t* list_node_at(elem_t *begin, index_t id)
{
  if (id < 0) return 0;
  index_t count = 0;
  while ((begin -> next != NULL) && (count < id))
  {
    begin = begin -> next;
    count++;
  }
  if (count == id)
    return begin;
  return NULL;
}


data_t list_get(elem_t *begin, index_t id)
{
  elem_t *temp = list_node_at(begin, id);
  if (temp) return temp -> data;
  return 0;
}


void list_free(elem_t *begin)
{
  elem_t *temp;
  while(begin != NULL)
  {
```

```
    temp = begin;
    begin = begin -> next;
    free(temp);
  }
}


index_t list_lenght(elem_t *begin)
{
  index_t size=0;
  while(begin != NULL)
  {
    begin = begin -> next;
    size++;
  }
  return size;
}


data_t list_sum(elem_t *begin)
{
  data_t sum=0;
  while(begin != NULL)
  {
    sum += begin -> data;
    begin = begin -> next;
  }
  return sum;
}
```

Текст программы (задание 2):

```
#include "list.h"
#include <stdbool.h>
#include <limits.h>

void print_nl(data_t value)
{
      printf("%d\n", value);
}

void print_sp(data_t value)
{
      printf("%d ", value);
}

void foreach( void(*f)(data_t), elem_t *begin)
{
      while (begin != NULL)
      {
            f(begin -> data);
            begin = begin -> next;
      }
}


data_t power2(data_t value)
{
  return value*value;
}
```

```
data_t power3(data_t value)
{
  return value*value*value;
}

elem_t* map(data_t(*f)(data_t), elem_t* begin)
{
  elem_t* newbegin = NULL;
  if (begin != NULL)
    newbegin = list_create(f(begin -> data));
  else
    return NULL;
  begin = begin -> next;
  while (begin != NULL)
  {
    list_add_back(&newbegin, f(begin -> data));
    begin = begin -> next;
  }
  return newbegin;
}

data_t get_abs(data_t x)
{
      return (data_t)abs(x);
}

void map_mut(data_t(*f)(data_t), elem_t *begin)
{
      while (begin != NULL)
      {
            begin -> data = f(begin->data);
            begin = begin->next;
      }
}

data_t foldl(data_t accum, data_t(*f)(data_t, data_t), elem_t *begin)
{
      while (begin != NULL)
      {
            accum = f(accum, begin -> data);
            begin = begin -> next;
      }
      return accum;
}

data_t sum(data_t x, data_t a)
{
      return x + a;
}

data_t get_min(data_t x, data_t a)
{
      if (a < x) x = a;
      return x;
}

data_t get_max(data_t x, data_t a)
{
      if (a > x) x = a;
      return x;
}
```

```
elem_t* iterate(data_t s, data_t length, data_t(*f)(data_t))
{
      index_t i;
      elem_t *new_list = list_create(f(s));
      elem_t *tmp_list = new_list;
      for (i = 1; i < length; i++)
      {
            list_add_back(&new_list, f(tmp_list->data));
            tmp_list = tmp_list->next;
      }
      return new_list;
}

data_t mul2(data_t value)
{
      return value*2;
}

bool save(elem_t *begin, const char *filename)
{
  FILE *file;
  file = fopen(filename, "w+t");
  if (!file) return false;
  while (begin != NULL)
  {
    data_t temp = begin -> data;
    fprintf(file, "%d ", temp);
    begin = begin -> next;
  }
  fclose(file);
  return true;
}

bool load(elem_t **begin, const char *filename)
{
  FILE *file;
  file = fopen(filename, "r+t");
  if (!file) return false;
  data_t temp;
  fscanf(file, "%d", &temp);
  *begin = list_create(temp);
  if (*begin == NULL) return false;
  while (fscanf(file, "%d", &temp) != EOF)
    list_add_back(begin, temp);
  fclose(file);
  return true;
}

bool serialize(elem_t *begin, const char *filename)
{
  FILE *file;
  file = fopen(filename, "w+b");
  if (!file) return false;
  while (begin != NULL)
  {
    data_t temp = begin -> data;
    fwrite(&temp, sizeof(data_t), 1, file);
    begin = begin -> next;
  }
  fclose(file);
```

```
  return true;
}

bool deserialize(elem_t **begin, const char *filename)
{
  FILE *file;
  file = fopen(filename, "r+b");
  if (!file) return false;
  data_t temp;
  fread(&temp, 1, sizeof(data_t), file);
  *begin = list_create(temp);
  if (*begin == NULL) return false;
  while (!feof(file))
  {
    fread(&temp, sizeof(data_t), 1, file);
    list_add_back(begin, temp);
  }
  fclose(file);
  return true;
}



int main()
{
  elem_t *begin = NULL;
  data_t value;
  scanf("%d", &value);
  begin = list_create(value);
  while (scanf("%d",&value) != EOF)
    list_add_front(&begin, value);

  printf("FOREACH:\n");
  foreach(&print_sp, begin);
  printf ("\n");
  foreach(&print_nl, begin);

  printf("\nMAP:\npow2: ");
  elem_t *temp = map(&power2, begin);
  foreach(&print_sp, temp);
  list_free(temp);
  printf("\npow3: ");
  temp = map(&power3, begin);
  foreach(&print_sp, temp);
  list_free(temp);
  printf("\n\n");

  printf("FOLDL:\n");
  printf("Sum: %d\n", foldl(0, &sum, begin));
  printf("Min: %d\n", foldl(INT_MAX, &get_min, begin));
  printf("Max: %d\n\n", foldl(INT_MIN, &get_max, begin));

  printf("MAP_MUT:\n");
  map_mut(&get_abs, begin);
  foreach(&print_sp, begin);
  puts("\n");

  puts("ITERATE:");
  elem_t *tempp = iterate(1, 10, &mul2);
  foreach(&print_sp, begin);
```

```
  save(begin, "filename.txt");
  puts("List saved!");

  list_free(begin);
  puts("Memory freed!");

  load(&begin, "filename.txt");
  puts("List loaded!");
  foreach(&print_sp, begin);

  serialize(begin, "filename.bin");
  puts("List serialize!");

  list_free(begin);
  puts("Memory freed!");

  deserialize(&begin, "filename.bin");
  puts("List deserialized!");
  foreach(&print_sp, begin);
  list_free(begin);
  return 0;
}
```

Вывод:
В результате проделанной работы был создан связный список и набор функций, обрабатывающих этот список. В результате выполнения задания 2 были написаны функции высшего порядка, обрабатывающие элементы созданного списка.