УНИВЕРСИТЕТ ИТМО

ПИиКТ

Языки Системного Программирования

ЛАБОРАТОРНАЯ РАБОТА №4

Группа: P3200

Студент: Базарова Анна Александровна

2018 год

Санкт-Петербург

Цель: написать программу, реализующую связный список; написать программу, реализующую функции высшего порядка на связном списке

Ниже приведем код для второго задания, так как он содержит в себе код первого:

lib\_ho.h

#include "lib\_ll.h"

#ifndef LAB4\_HIGHER\_ORDER\_H

#define LAB4\_HIGHER\_ORDER\_H

/\*\*

\* функция по очереди выполняет функцию на всех элементах саиска по порядку

\* @param list - указатель на list элементов

\* @param f - указатель на функцию

\*/

void foreach(struct list const\* list, void (\*f)(int));

/\*\*

\* Функция создает новый список, в котором к содержимомоу каждому элементу list'а применяется функция f

\* @param list - указатель на исходный список (список не изменяется)

\* @param f - указатель на функцию

\* @return новый list с результатом приминения f к каждому элементу list'a

\*/

struct list\* map(struct list const\* list, int (\*f)(int));

/\*\*

\* Функция возвращает сущность того же типа, что и аккумулятор, запуская f т копии аккумулятора и текущим элементом списка

\* @param acc - начальное значение аккумулятора

\* @param f - указатель на функцию

\* @param list - список элементов

\* @return полученное значение

\*/

int foldl(int acc, int (\*f)(int, int), struct list const\* list);

/\*\*

\* Функция изменяет каждый элемент list'a, применяя функцию f

\* @param list

\* @param f

\*/

void map\_mut(struct list\* list, int (\*f)(int));

/\*\*

\* Ренерирует списов вида: [s, f(s), f(f(s)), ...], длинной n

\* @param s - первый элемент списка

\* @param n - длинна списка

\* @param f - указатель на функцию

\* @return новый список

\*/

struct list\* iterator(int s, size\_t n, int (\*f)(int));

#endif //LAB4\_HIGHER\_ORDER\_H

lib\_ho.c

#include <stdio.h>

#include <stdlib.h>

#include "lib\_ho.h"

void foreach(struct list const\* list, void (\*f)(int)){

while (list != NULL){

f(list->elem);//пока не дойдем до первого, применяем к каждому значению ноды нужную функцию

list = list->last;

}

}

struct list\* map(struct list const\* list, int (\*f)(int)){

struct list\* result\_list = NULL;

while (list != NULL){//пока не дойдем до первого

list\_add\_front(&result\_list, f(list->elem));//дописываем в новый список в начало новый элемент, полученный из старого применением функции

list = list->last;

}

return result\_list;

}

int foldl(int acc, int (\*f)(int, int), struct list const\* list){

while (list != NULL){//для всех эл-тов списка

acc = f(acc, list->elem);//мы для acc используем функцию от acc и элемента списка

list = list->last;

}

return acc;

}

void map\_mut(struct list\* list, int (\*f)(int)){

while(list != NULL){//для каждого элемента списка мы присваиваем значение функции от значения этого элемента

list->elem = f(list->elem);

list = list->last;

}

}

struct list\* iterator(int s, size\_t n, int (\*f)(int)){

struct list\* list = NULL;

size\_t i = 0;

for(; i < n; ++i){

list\_add\_front(&list, s);//[s,f(s),f(f(s)),f(f(f(s))),...]

s = f(s);

}

return list;

}

lib\_ll.h

#ifndef LAB4\_LINKED\_LIST\_H

#define LAB4\_LINKED\_LIST\_H

#include <stdbool.h>

#include <stdlib.h>

/\*\*

\* Структура связного списка

\* Элементом списка является int

\* Хранятся указатели на следующий и предыдущий элемент

\*/

struct list {

int elem;

struct list\* next;

struct list\* last;

};

//задумайся, человек! list\* - это всего лишь последняя нода нашего великого связного списка

//Зная хоть один list\* мы знаем весь связный список ибо этого достаточно, поэтому везде передаем ее, о человек!

/\*\*

\* Функция для создания списка с первым элементом

\* @param e - элемент списка

\* @return указатель на список

\*/

struct list\* list\_create(int e);

/\*\*

\* Функция для добавления элемента в начало списка

\* Если список пуст, создает новый список

\* @param list - указатель на список, в который добавляем элемент

\*/

void list\_add\_front(struct list\*\* list, int e);

/\*\*

\* Функция для добавления элемента в конец списка

\* Если список пуст, создает новый список

\* @param list - указатель на список, в который добавляем элеент

\*/

void list\_add\_back(struct list\*\* list, int e);

/\*\*

\* Функция возвращает значение элемента списка по индексу

\* @param list - указатель на список

\* @param index - индекс элемента списка

\* @return значение элемента списка (если индекса в списке нет, вернет 0, шо поделать ¯\\_(ツ)\_/¯)

\*/

int list\_get(struct list const\* list, unsigned int index);

/\*\*

\* Функция удаляет все элементы списка

\* @param list - указатель на список

\*/

void list\_free(struct list\*\* list);

/\*\*

\* Функция считает длинну списка

\* @param list - указатель на список

\* @return длинну списка

\*/

size\_t list\_length(struct list const\* list);

/\*\*

\* Функция возвращает элемент списка по индексу

\* @param list - указатель на список

\* @param index - индекс элемента списка

\* @return элемент списка или NULL, если индекс выходит за границы списка или list пуст

\*/

struct list\* list\_node\_at(struct list const\* list, unsigned int index);

/\*\*

\* Функция считает сумму элементов списка

\* @param list - указатель на list

\* @return сумму элементов списка

\* 0, если list пуст (в общем-то, логично)

\*/

int list\_sum(struct list const\* list);

/\*\*

\* Функция загружает list с файла

\* @param list - указатель на list

\* @param filename - имя файла

\* @return true - успешное выполнение функции

\* false - завершение выполнения с ошибкой

\*/

bool load(struct list\*\* list, const char\* filename);

/\*\*

\* Функция сохраняет list в файл

\* @param list - указатель на list

\* @param filename - имя файла

\* @return true - успешное выполнение функции

\* false - завершение выполнения с ошибкой

\*/

bool save(struct list\* list, const char\* filename);

#endif //LAB4\_LINKED\_LIST\_H

lib\_ll.c

#include <stdlib.h>

#include <stdio.h>

#include "lib\_ll.h"

struct list \* list\_create(int e) {

struct list \*new\_elem\_list = (struct list\*)malloc(sizeof(struct list));//динамически выделяем память под первую ноду нашего списка (стд)

new\_elem\_list->elem = e;//нынешний элемент, т.е. первый - e

new\_elem\_list->last = NULL;//других элемнентов нет ни до ни после

new\_elem\_list->next = NULL;

return new\_elem\_list;

}

void list\_add\_front(struct list\* \*list, int e){

if (\*list == NULL){//если такого списка нет - создадим

\*list = list\_create(e);

return;

}

struct list \*new\_elem\_list = (struct list\*)malloc(sizeof(struct list));//выделим еще памяти под новую ноду

new\_elem\_list->elem = e;

new\_elem\_list->last = NULL;//так как дописали вперед, то перед ним ничего нет

struct list\* tmp\_list = \*list;

while(tmp\_list->last != NULL){

tmp\_list = tmp\_list->last;//tmp\_list это первый элемент \*list

}

new\_elem\_list->next = tmp\_list;//говорим нашей ноде что ее следующий элемент - это первый элемент \*list

tmp\_list->last = new\_elem\_list;//говорим нашему первому элементу что его предыдущий элемент наша нода

}

void list\_add\_back(struct list\*\* list, int e){

if (\*list == NULL){//если такого списка нет - создадим

\*list = list\_create(e);

return;

}

struct list \*new\_elem\_list = (struct list\*)malloc(sizeof(struct list));//выделим еще памяти под новую ноду

new\_elem\_list->elem = e;

new\_elem\_list->next = NULL;//так как дописали назад, то после него ничего нет

new\_elem\_list->last = \*list;

(\*list)->next = new\_elem\_list;//последний элемент теперь знает что он последний

\*list = new\_elem\_list;//теперь е - последняя нода, муа ха ха

}

int list\_get(struct list const\* list, unsigned int index){

struct list\* ret = list\_node\_at(list, index);//ret - либо указатель на ноду с нужным индексом, либо null если индекс выходит за пределы списка

return ret == NULL ? 0 : ret->elem;

}

void list\_free(struct list\*\* list){

while (\*list != NULL) {//пока список не пуст, т.е. пока последняя нода есть

struct list\* last = (\*list)->last;//запоминаем предыдущую ноду, а текущую "сжигаем"

free(\*list);//освобождаем ранее выделенную динамичесую память(стд)

(\*list) = last;//переходим к предыдущей ноде

}

}

size\_t list\_length(struct list const\* list){

size\_t i = 0;

while (list != NULL) {

struct list\* last = list->last;//доходим с конца до первого элемента

list = last;

++i;//параллельно считаем, сколько нод уже прошли

}

return i;

}

struct list\* list\_node\_at(struct list const\* list, unsigned int index){

unsigned int i = 0;

if (list == NULL){

return NULL;//тут все логично

}

while (list->last != NULL){

list = list->last;//находим первый элемент

}

while (i != index){

++i;

list = list->next;

if (list == NULL) {

return NULL;//если следующего элемента нет, то index за пределами списка

}

}

return list;//возвращаем нужную ноду

}

int list\_sum(struct list const\* list){

int sum = 0;

while (list != NULL){//пока не дошли до первого

sum += list->elem;//суммируем значение

list = list->last;//переходим к предыдущему

}

return sum;

}

bool load(struct list\*\* list, const char\* filename){

int e;

FILE\* input\_file = fopen(filename, "r");

if (input\_file == NULL) {

return false;

}

while ( fscanf(input\_file, "%d", &e) != EOF) {

list\_add\_back(list, e);

}

fclose(input\_file);

return true;

}

bool save(struct list\* list, const char\* filename){

FILE\* output\_file = fopen(filename, "w+");

if (output\_file == NULL){

return false;

}

while (list != NULL) {

fprintf(output\_file, "%d ", list->elem);

list = list->last;

}

fclose(output\_file);

return true;

}

Lab4b.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <limits.h>

#include "lib\_ll.h"

#include "lib\_ho.h"

/\*\*

\* Функция для вывода list'а через пробелы

\*/

void fun\_print\_with\_space(int x) {

printf("%d ", x);

}

/\*\*

\* Функция для вывода list'а через переносы строки

\*/

void fun\_print\_with\_n(int x){

printf("%d\n", x);

}

/\*\*

\* функция возводит в квадрат

\*/

int fun\_square(int x){

return x\*x;

}

/\*\*

\* Функция возводит в куб

\*/

int fun\_cube(int x){

return x\*x\*x;

}

/\*\*

\* Функция возввращает сумму элементов

\*/

int fun\_sum(int acc, int x){

return acc + x;

}

/\*\*

\* Функция для поиска максимального значения в list

\*/

int fun\_max(int acc, int x){

return acc > x ? acc : x;

}

/\*\*

\* Функция для поиска минимального в list

\*/

int fun\_min(int acc, int x){

return acc < x ? acc : x;

}

/\*\*

\* Функция возвращает модуль числа

\*/

int fun\_abs(int x){

return abs(x);

}

/\*\*

\* Функция для степент 2

\*/

int fun\_pow\_two(int x){

return x\*2;

}

int main() {

int e;

struct list\* list = NULL;

printf("Введите - 1 чтобы считать из файла\n"

" - 2 чтобы считать с консоли\n");

int k;

scanf("%d", &k);

switch (k){

case 1: {

load(&list, "input.txt");

printf("Считали список из файла 'input.txt'.\n");

break;

}

case 2: {

printf("Введите список:\n");

while (1){

int s = scanf("%d", &e);

if (s == EOF){

break;

}

if (s != 1){

printf("%c - не является числом. Введите, пожалуйста число.\n", getchar());

} else {

printf("Элемент '%d' добавлен в список\n", e);

list\_add\_back(&list, e);

}

}

break;

}

default: {

printf("Неверно. Нужно было вводить правильные чиселки ;(");

return 0;

}

}

printf("\n\nFOREACH:\nВыведем список через перевод строки:\n");

void (\*pf)(int) = &fun\_print\_with\_n;

foreach(list, pf);

printf("\nИ через пробел:\n");

pf = &fun\_print\_with\_space;

foreach(list, pf);

printf("\n\nMAP:\nВозведем каждый элемент списка в квадрат:\n");

int (\*ipf)(int) = &fun\_square;

struct list\* res\_list = map(list, ipf);

foreach(res\_list, pf);

printf("\nВозведем каждый элемент списка в куб:\n");

ipf = &fun\_cube;

res\_list = map(list, ipf);

foreach(res\_list, pf);

list\_free(&res\_list);

int (\*ipfii)(int, int) = &fun\_sum;

printf("\n\nFOLDL:\nСумма элементов, посчитанная с помощью foldl: %d\n", foldl(0, ipfii, list));

ipfii = &fun\_max;

printf("Максимальный элемент в list: %d\n", foldl(INT\_MIN, ipfii, list));

ipfii = &fun\_min;

printf("Минимальный элемент в list: %d\n", foldl(INT\_MAX, ipfii, list));

printf("\nMAP\_MUT:\nМодуль всех чисел:\n");

int (\*ipfi)(int) = &fun\_abs;

map\_mut(list, ipfi);

foreach(list, pf);

printf("\n\nITERATOR:\n10 первых степеней двойки:\n");

ipfi = &fun\_pow\_two;

res\_list = iterator(1, 10, ipfi);

foreach(res\_list, pf);

list\_free(&res\_list);

save(list, "output.txt");

printf("\n\nКонечный список сохранен в 'output.txt'\n");

printf("\nПамять очищена.\n");

list\_free(&list);

return 0;

}