## Университет ИТМО

## Кафедра ВТ

Лабораторная работа №4

Низкоуровневое программирование

Выполнил: Федоров Сергей

Группа: P33113

Санкт-Петербург

2020 г.

## Задание лабораторной работы:

Реализовать на языке С связный список и методы к нему:

* list\_create – создание списка
* list\_add\_front – добавление элемента к голове
* list\_add\_back – добавление элемента к хвосту
* list\_free – очистка памяти от списка
* list\_length – подсчет длинны списка
* list\_sum – подсчет суммы элементов
* list\_node\_at – взятие вершины списка на n-ой позиции
* list\_get - взятие значения элемента списка на n-ой позиции
* foreach – применение side-effect по каждому элементу
* map – отображение списка в другой по функции
* map\_mut – отображение списка в самого себя по функции
* fold\_l – свертка списка по функции и с аккумулятором
* unfold\_l (iterate) – развертка/генерация списка по правилу
* save – сохранение списка в файл (текстовое представление)
* load – загрузка списка из файла (текстовое представление)
* serialize - сохранение списка в файл (бинарное представление)
* deserialize - загрузка списка из файла (бинарное представление)

**Выполнение:**

Lab4/

├── functional.c

├── functional.h

├── linked\_list.c

├── linked\_list.h

├── list\_io.c

├── list\_io.h

└── main.c

**main.c**

#include <stdio.h>

#include "linked\_list.h"

#include "functional.h"

#include "list\_io.h"

**void** print\_val(value a){

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

}

**void** print\_val\_whitespace(value a){

printf("%d ", a);

}

value sqr(value v){

**return** v \* v;

}

value cbr(value v){

**return** v \* v \* v;

}

value abs(value v){

**return** v > 0 ? v : -v;

}

value times\_2(value v){

**return** v \* 2;

}

**int** main() {

// PART 1 - PAGE 197

linked\_list list\_1 = read\_from\_console();

puts("YOUR ARRAY:");

print\_list(list\_1);

puts("SUM OF THE ARRAY:");

print\_val(list\_sum(list\_1));

list\_free(list\_1);

// PART 2 - PAGE 218

linked\_list list\_2 = read\_from\_console();

puts("YOUR ARRAY:");

print\_list(list\_2);

puts("PRINTED BY FOREACH ' ':");

foreach(list\_2, &print\_val\_whitespace);

puts("");

puts("PRINTED BY FOREACH '\\n':");

foreach(list\_2, &print\_val);

puts("SQUARES:");

linked\_list list\_2\_sqr = map(list\_2, &sqr);

print\_list(list\_2\_sqr);

puts("CUBES:");

linked\_list list\_2\_cbr = map(list\_2, &cbr);

print\_list(list\_2\_cbr);

puts("ABS:");

map\_mut(list\_2, &abs);

print\_list(list\_2);

puts("POWERS of two:");

linked\_list twos = unfold\_l(1, 10, &times\_2);

print\_list(twos);

puts("WRITING LIST TO FILE (TEXT)...");

**const** **char**\* file\_text = "saved.txt";

save(list\_2, file\_text);

linked\_list read\_text\_list;

load(&read\_text\_list, file\_text);

puts("READ THESE NUMBERS BACK:");

print\_list(read\_text\_list);

puts("WRITING LIST TO FILE (BIN)...");

**const** **char**\* file\_bin = "saved.bin";

serialize(list\_2, file\_bin);

linked\_list read\_bin\_list;

deserialize(&read\_bin\_list, file\_bin);

puts("READ THESE NUMBERS BACK:");

print\_list(read\_bin\_list);

list\_free(list\_2);

list\_free(list\_2\_sqr);

list\_free(list\_2\_cbr);

list\_free(twos);

list\_free(read\_text\_list);

list\_free(read\_bin\_list);

puts("FREED ALL ALLOCATED MEMORY");

puts("FIN.");

}

**linked\_list.h**

#ifndef linked\_list\_h

#define linked\_list\_h

#include <stdio.h>

**typedef** **int** value;

**struct** Node {

**struct** Node\* next\_elem;

value node\_value;

};

**typedef** **struct** Node\* linked\_list;

linked\_list list\_create(value new\_value);

linked\_list list\_add\_front(value new\_value, **const** linked\_list list);

linked\_list list\_add\_back(value new\_value, **const** linked\_list list);

**void** list\_free(**const** linked\_list list); // TODO Implement through foreach

size\_t list\_length(**const** linked\_list list); // TODO Implement through foreach

value list\_sum(**const** linked\_list list); // TODO Implement through foldl

linked\_list list\_node\_at(size\_t pos, **const** linked\_list list);

value list\_get(size\_t pos, **const** linked\_list list);

**void** print\_list(**const** linked\_list); // TODO Implement through foreach

#endif /\* linked\_list\_h \*/

**linked\_list.c**

#include "stdlib.h"

#include "stdio.h"

#include "linked\_list.h"

#include "functional.h"

linked\_list list\_create(value new\_value) {

linked\_list node\_p = malloc(**sizeof**(**struct** Node));

node\_p->node\_value = new\_value;

node\_p->next\_elem = **NULL**;

**return** node\_p;

}

**void** list\_free(**const** linked\_list list) {

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

linked\_list prev\_node = next\_node;

next\_node = next\_node->next\_elem;

free(prev\_node);

}

}

linked\_list list\_add\_front(value new\_value, **const** linked\_list list) {

linked\_list new\_node = list\_create(new\_value);

new\_node->next\_elem = list;

**return** new\_node;

}

linked\_list list\_add\_back(value new\_value, **const** linked\_list list) {

linked\_list next\_node = list;

**while**(next\_node->next\_elem != **NULL**) {

next\_node = next\_node->next\_elem;

}

linked\_list new\_node = list\_create(new\_value);

next\_node->next\_elem = new\_node;

**return** list;

}

linked\_list list\_node\_at(size\_t pos, **const** linked\_list list) {

linked\_list next\_node = list;

**if** (pos < 0) **return** **NULL**;

**else** {

size\_t iter;

**for** (iter = 0; iter < pos; iter++) {

**if** (next\_node->next\_elem == **NULL**) **break**;

**else** next\_node = next\_node->next\_elem;

}

**if** (iter != pos) **return** **NULL**;

**else** **return** next\_node;

}

}

value list\_get(size\_t pos, **const** linked\_list list){

linked\_list desired\_node = list\_node\_at(pos, list);

**if** (desired\_node == **NULL**) **return** -1;

**else** **return** desired\_node->node\_value;

}

**static** **void** print\_elem(value v) {

printf("%d ", v);

}

**void** print\_list(**const** linked\_list list) {

foreach(list, &print\_elem);

puts("");

}

**static** value sum(value a, value b){

**return** a + b;

}

value list\_sum(**const** linked\_list list) {

**return** fold\_l(list, 0, &sum);

}

size\_t list\_length(**const** linked\_list list) {

linked\_list next\_node = list;

size\_t len = 0;

**while** (next\_node != **NULL**) {

len += 1;

next\_node = next\_node->next\_elem;

}

**return** len;

}

**functional.h**

#ifndef functional\_h

#define functional\_h

#include "linked\_list.h"

// For anonymous functions

#define lambda(l\_ret\_type, l\_arguments, l\_body) \

({ \

l\_ret\_type l\_anonymous\_functions\_name l\_arguments \

l\_body \

&l\_anonymous\_functions\_name; \

})

**void** foreach(**const** linked\_list list, **void**(\*func)(value));

linked\_list map(**const** linked\_list list, value(\*op)(value));

**void** map\_mut(**const** linked\_list list, value(\*op)(value));

value fold\_l(**const** linked\_list list, value acc, value(\*op)(value, value));

linked\_list unfold\_l(value init, size\_t length, value(\*op)(value)); // Basically same as iterate, but more canonical;

#endif /\* functional\_h \*/

**functional.c**

#include "stdlib.h"

#include "functional.h"

**void** foreach(**const** linked\_list list, **void**(\*func)(value)) {

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

func(next\_node->node\_value);

next\_node = next\_node->next\_elem;

}

}

linked\_list map(**const** linked\_list list, value(\*op)(value)) {

**if** (list != **NULL**) {

linked\_list next\_node = list;

linked\_list new\_node = list\_create(op(next\_node->node\_value));

linked\_list new\_list = new\_node;

next\_node = next\_node->next\_elem;

**while** (next\_node->next\_elem != **NULL**) {

new\_node = list\_add\_back(op(next\_node->node\_value), new\_node)->next\_elem;

next\_node = next\_node->next\_elem;

}; list\_add\_back(op(next\_node->node\_value), new\_node);

**return** new\_list;

} **else** {

**return** **NULL**;

}

}

**void** map\_mut(**const** linked\_list list, value(\*op)(value)) {

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

next\_node->node\_value = op(next\_node->node\_value);

next\_node = next\_node->next\_elem;

};

}

value fold\_l(**const** linked\_list list, value acc, value(\*op)(value, value)) {

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

acc = op(acc, next\_node->node\_value);

next\_node = next\_node->next\_elem;

}

**return** acc;

}

linked\_list unfold\_l(value init, size\_t length, value(\*op)(value)) {

linked\_list new\_list = list\_create(init);

linked\_list new\_node = new\_list;

size\_t iter;

**for** (iter = 1; iter < length; iter++) {

init = op(init);

new\_node = list\_add\_back(init, new\_node)->next\_elem;

}

**return** new\_list;

}

**list\_io.h**

#ifndef list\_io\_h

#define list\_io\_h

#include <stdio.h>

#include "linked\_list.h"

**int** save(**const** linked\_list list, **const** **char**\* filename);

**int** load(linked\_list\* list, **const** **char**\* filename);

**int** serialize(**const** linked\_list list, **const** **char**\* filename);

**int** deserialize(linked\_list\* list, **const** **char**\* filename);

linked\_list read\_from\_console(**void**);

#endif /\* list\_io\_h \*/

**list\_io.c**

#include <stdio.h>

#include <errno.h>

#include "list\_io.h"

**static** **int** check\_file(FILE\* file) {

**if** (errno || ferror(file)) {

fclose(file);

**return** 0;

} **else** **return** 1;

}

**static** **int** close\_file(FILE\* file) {

fclose(file);

**if** (errno) **return** 0;

**else** **return** 1;

}

**static** FILE\* open\_file(**const** **char**\* filename, **const** **char**\* mode) {

errno = 0;

FILE\* file = fopen(filename, mode);

**if** (errno) **return** **NULL**;

**else** **return** file;

}

**int** save(linked\_list list, **const** **char**\* filename) {

FILE\* file = open\_file(filename, "w");

**if** (file == **NULL**) **return** 0;

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

fprintf(file, "%d ", next\_node->node\_value);

**if** (!check\_file(file)) **return** 0;

next\_node = next\_node->next\_elem;

}

**return** close\_file(file);

}

**int** load(linked\_list\* list, **const** **char**\* filename) {

FILE\* file = open\_file(filename, "r");

**if** (file == **NULL**) **return** 0;

linked\_list next\_node = **NULL** , new\_list = **NULL**;

value read\_val;

**while** (1) {

fscanf(file, "%d", &read\_val);

**if** (feof(file)) **break**;

**if** (!check\_file(file)) **return** 0;

**if** (new\_list == **NULL**) {

new\_list = list\_create(read\_val);

next\_node = new\_list;

} **else** {

next\_node = list\_add\_back(read\_val, next\_node)->next\_elem;

}

}

\*list = new\_list;

**return** close\_file(file);

}

**int** serialize(linked\_list list, **const** **char**\* filename) {

FILE\* file = open\_file(filename, "wb");

**if** (file == **NULL**) **return** 0;

linked\_list next\_node = list;

**while** (next\_node != **NULL**) {

fwrite(&next\_node->node\_value, **sizeof**(value), 1, file);

**if** (!check\_file(file)) **return** 0;

next\_node = next\_node->next\_elem;

}

**return** close\_file(file);

}

**int** deserialize(linked\_list\* list, **const** **char**\* filename) {

FILE\* file = open\_file(filename, "rb");

**if** (file == **NULL**) **return** 0;

linked\_list next\_node = **NULL** , new\_list = **NULL**;

value read\_val;

**while** (1) {

fread(&read\_val, **sizeof**(value), 1, file);

**if** (feof(file)) **break**;

**if** (!check\_file(file)) **return** 0;

**if** (new\_list == **NULL**) {

new\_list = list\_create(read\_val);

next\_node = new\_list;

} **else** {

next\_node = list\_add\_back(read\_val, next\_node)->next\_elem;

}

}

\*list = new\_list;

**return** close\_file(file);

}

linked\_list read\_from\_console() {

value read\_val;

linked\_list new\_list = **NULL**;

puts("ENTER N:");

**if** (scanf("%d", &read\_val) != 1) **return** **NULL**;

puts("ENTER LIST NUMBERS:");

**int** n = read\_val;

**int** i;

**for**(i = 0; i < n; i++) {

**if** (scanf("%d", &read\_val) != 1) **break**;

**else** new\_list = list\_add\_front(read\_val, new\_list);

}

**return** new\_list;

}

**Вывод:**

На самом деле не особо много мыслей по поводу данной лабораторной работе, но вот следующая ремарка найдется:

Даже несмотря на то что на языке С приходится писать больше кода чем на других более высокоуровневых языках, а отсутствие некоторых синтаксических возможностей достаточно сильно бьет по читаемости кода и его удобству, все равно при выполнении именно таких задач проявляется сильная сторона данного языка - мы можем с легкостью реализовывать различные алгоритмы и структуры данных, настолько близко и настолько оптимизированно к «машине» насколько возможно, что в свою очередь может быть очень даже полезно.