****Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовая работа**

**по курсу «Языки и методы программирования»**

**II Семестр**

**Задание 8**

**Линейные списки**

Группа: М80 – 107Б-18

Студент: Син Денис Дмитриевич

Преподаватель: Ридли Александра Николаевна

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2018.

**Содержание**

1. Постановка задачи
2. Общие сведения о программе
3. Общий метод и алгоритм решения
4. Основные файлы программы
5. Тесты
6. Демонстрация работы программы
7. Вывод

**Постановка задачи**.

Составить и отладить программу на языке СИ для обработки линейного списка заданной организации с отображением списка на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного и четырех стандартных действий.

**Тип элемента списка**: Целый.

**Вид списка**: Линейный двунаправленный.

**Нестандартное действие**: Удалить из списка все элементы, предшествующие и последующие заданному значению.

**Общие сведения о программе**

Дерево проекта:

.

├── Makefile

├── list.c

├── list.h

├── main.c

Makefile – файл для сборки проекта

List.c – файл в котором реализована структура данных линейный двунаправленный список.

Lish.h – заголовочный файл, в котором описаны структуры и функции для списка.

Main.c – главный файл для тестирования списка.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Реализовать структуру Node, которая хранит в себе 3 поля указатель на следующий узел, указатель на предыдущий узел и поле для хранения данных.
2. Реализовать структуру List, которая хранит в себе 3 поля указатель на голову, указатель на хвост и количество элементов в списке.
3. Реализовать структуру Iterator, которая хранит в себе 1 поле указатель на Node.
4. Для итератора нужно реализовать следующие функции:
   1. first – возвращает итератор на голову.
   2. Last – возвращает итератор на head.
   3. prev – возвращает итератор на предыдущий узел
   4. next – возвращает итератор на следующий узел
   5. fetch – возвращает значение узла итератора
   6. store – записывает значение в узел итератора
5. Для списка нужно реализовать следующие функции:
   1. list\_сreate – создает список
   2. list\_insert – вставляет узел перед передаваемым итератором
   3. list\_push\_back – добавляет узел в конец списка
   4. list\_del – удаляет узел
   5. list\_print – печатает список
   6. list\_destroy – отчищает память, которую занимает список
   7. list\_len – возвращает количество элементов в списке
   8. list\_is\_empty – проверка на пустоту списка
   9. list\_del\_more\_less\_value – удаляет из списка все элементы, предшествующие и последующие заданному значению.
6. Для реализации вставки необходимо правильно поменять конфигурацию указателей и при этом проверять на конец списка и на начало списка, так как при добавлении в конец необходимо смещать хвост, также и при добавлении в начало необходимо смещать голову.
7. Для удаления опять же необходимо, верно, поменять конфигурацию указателей и тоже нужно следить за удалением в конце или в начале списка. Операции вставки и удаления работают за O (1)
8. Чтобы реализовать нестандартное действие, нужно с помощью итераторов обойти список и применить операцию удаления к узлу, хранимое значение которого соответствует параметрам из условия.

**Основные файлы программы**.

**Makefile**

CC = gcc

CCKYES = -Wall -Wextra -Werror -pedantic

LD = gcc

LDKEYS =

main: main.o list.o list.h

    $(LD) $(CCKYES) -o main main.o list.o

main.o: main.c list.h

    $(CC) $(CCKYES) -c main.c

list.o: list.c list.h

    $(CC) $(CCKYES) -c list.c

fclean: clean

    @rm -rf main

clean:

    @rm -rf \*.o

**list.h**

#ifndef LIST\_H\_

#define LIST\_H\_

#include <stdlib.h>

#include <stdbool.h>

#include <stdio.h>

typedef int data\_type;

typedef struct elem {

data\_type data;

struct elem \*next;

struct elem \*prev;

} Node;

typedef struct {

Node \*head;

Node \*tail;

size\_t size;

} List;

typedef struct {

    Node \*node;

} Iterator;

/\* Iterators \*/

Iterator first(List \*lst);

Iterator last(List \*lst);

Iterator    next(Iterator \*i);

Iterator    prev(Iterator \*i);

int         fetch(const Iterator \*i);

void        store(const Iterator \*i, const data\_type elem);

/\* List \*/

void        list\_create(List \*lst);

void     list\_insert(List \*lst, Iterator \*i, data\_type num);

void list\_push\_back(List \*lst, data\_type value);

void     list\_del(List \*lst, Iterator \*i);

void        list\_print(List \*lst);

void        list\_destroy(List \*lst);

size\_t      list\_len(List \*lst);

bool list\_is\_empty(List \*lst);

void list\_del\_more\_less\_value(List \*lst, data\_type value);

#endif

**list.c**

#include "list.h"

/\* Functions for Iterators \*/

Iterator first(List \*lst)

{

Iterator res = { lst->head };

return res;

}

Iterator last(List \*lst)

{

Iterator res = { lst->tail };

return res;

}

Iterator prev(Iterator \*i)

{

i->node = i->node->prev;

return \*i;

}

Iterator next(Iterator \*i)

{

i->node = i->node->next;

return \*i;

}

data\_type fetch(const Iterator \*i)

{

return i->node->data;

}

void store(const Iterator \*i, const int elem)

{

i->node->data = elem;

}

/\* Fuctions for list \*/

void list\_create(List \*lst)

{

lst->head = NULL;

lst->tail = NULL;

    lst->size = 0;

}

void list\_insert(List \*lst, Iterator \*i, data\_type value)

{

    Iterator res = { (Node \*)malloc(sizeof(Node)) };

if (res.node == NULL)

exit(1);

res.node->next = NULL;

res.node->prev = NULL;

store(&res, value);

res.node->prev = i->node->prev;

res.node->next = i->node;

if (res.node->prev != NULL)

res.node->prev->next = res.node;

i->node->prev = res.node;

if (i->node == lst->head)

lst->head = res.node;

lst->size++;

}

void list\_push\_back(List \*lst, data\_type value)

{

Iterator res = { (Node \*)malloc(sizeof(Node)) };

if (res.node == NULL)

exit(1);

res.node->next = NULL;

res.node->prev = NULL;

store(&res, value);

if (lst->size == 0) {

lst->head = res.node;

lst->tail = res.node;

} else {

lst->tail->next = res.node;

res.node->prev = lst->tail;

res.node->next = NULL;

lst->tail = res.node;

}

lst->size++;

}

void list\_del(List \*lst, Iterator \*i)

{

if (lst->head == i->node)

lst->head = lst->head->next;

else if (lst->tail == i->node)

lst->tail = lst->tail->prev;

if (i->node->next != NULL)

i->node->next->prev = i->node->prev;

if (i->node->prev != NULL)

i->node->prev->next = i->node->next;

free(i->node);

i->node = NULL;

lst->size--;

}

size\_t list\_len(List \*lst)

{

return lst->size;

}

bool list\_is\_empty(List \*lst)

{

return list\_len(lst) == 0;

}

void list\_print(List \*lst)

{

Iterator i = first(lst);

while (i.node != NULL) {

printf(i.node != lst->tail ? "%d <=> " : "%d\n", fetch(&i));

next(&i);

}

}

void list\_destroy(List \*lst)

{

Iterator i = first(lst);

Iterator i\_del;

while (i.node) {

i\_del = i;

next(&i);

list\_del(lst, &i\_del);

}

}

void list\_del\_more\_less\_value(List \*lst, data\_type value)

{

Iterator i = first(lst);

Iterator i\_del;

while (i.node) {

i\_del = i;

next(&i);

if (fetch(&i\_del) != value)

list\_del(lst, &i\_del);

}

}

**Тесты**.

**Тест 1**

Добавить 5 элементов в начало и вывести их с помощью функций list\_insert и list\_print.

**Тест 2**

В списке из 5 элементов получить итератор на первый и на последний элементы и удалить их, затем напечатать список.

**Тест 3**

В списке из 5 элементов получить итератор на третий элемент и удалить узел, затем напечатать список.

**Тест 4**

В списке из 5 элементов получить итератор на третий элемент и вставить узел, затем напечатать список.

**Тест 5**

К списку из 20 элементов применить нестандартное действие, cо значением, которого нет в списке и напечатать список до и после.

**Тест 6**

К списку из 20 элементов применить нестандартное действие, cо значением, которое встречается 1 раз в списке и напечатать список до и после.

**Тест 7**

К списку из 20 элементов применить нестандартное действие, cо значением, которое встречается более 1 раза в списке и напечатать список до и после.

**Тест 8**

К пустому списку применить нестандартное действие.

**Демонстрация работы программы**.

**Тест 1**

➜ KP3 make

➜ KP3 ls

course\_project3.docx Makefile list.c list.h list.o main main.c main.o ~$ourse\_project3.docx

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_insert(&lst, &i, 4);

i = first(&lst);

list\_insert(&lst, &i, 1);

i = first(&lst);

list\_insert(&lst, &i, -9);

i = first(&lst);

list\_insert(&lst, &i, 0);

i = first(&lst);

list\_insert(&lst, &i, -87);

list\_print(&lst);

return 0;

}

➜ KP3 ./main

-87 <=> 0 <=> -9 <=> 1 <=> 4

**Тест 2**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

Iterator begin;

Iterator end;

list\_create(&lst);

i = first(&lst);

list\_insert(&lst, &i, 4);

i = first(&lst);

list\_insert(&lst, &i, 1);

i = first(&lst);

list\_insert(&lst, &i, -9);

i = first(&lst);

list\_insert(&lst, &i, 0);

i = first(&lst);

list\_insert(&lst, &i, -87);

list\_print(&lst);

begin = first(&lst);

end = last(&lst);

list\_del(&lst, &begin);

list\_del(&lst, &end);

list\_print(&lst);

list\_destroy(&lst);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

-87 <=> 0 <=> -9 <=> 1 <=> 4

0 <=> -9 <=> 1

**Тест 3**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_insert(&lst, &i, 4);

i = first(&lst);

list\_insert(&lst, &i, 1);

i = first(&lst);

list\_insert(&lst, &i, -9);

i = first(&lst);

list\_insert(&lst, &i, 0);

i = first(&lst);

list\_insert(&lst, &i, -87);

list\_print(&lst);

i.node = lst.head->next->next;

list\_del(&lst, &i);

list\_print(&lst);

list\_destroy(&lst);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

-87 <=> 0 <=> -9 <=> 1 <=> 4

-87 <=> 0 <=> 1 <=> 4

**Тест 4**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_insert(&lst, &i, 4);

i = first(&lst);

list\_insert(&lst, &i, 1);

i = first(&lst);

list\_insert(&lst, &i, -9);

i = first(&lst);

list\_insert(&lst, &i, 0);

i = first(&lst);

list\_insert(&lst, &i, -87);

list\_print(&lst);

i.node = lst.head->next->next;

list\_insert(&lst, &i, 200);

list\_print(&lst);

list\_destroy(&lst);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

-87 <=> 0 <=> -9 <=> 1 <=> 4

-87 <=> 0 <=> 200 <=> -9 <=> 1 <=> 4

**Тест 5**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, -34);

list\_push\_back(&lst, 309);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 352);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, -32);

list\_push\_back(&lst, -98);

list\_push\_back(&lst, 107);

list\_push\_back(&lst, -67);

list\_push\_back(&lst, -68);

list\_push\_back(&lst, 45);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 98);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 23);

list\_push\_back(&lst, 43);

list\_push\_back(&lst, 76);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, 3);

printf("Original list:\n");

list\_print(&lst);

printf("-------------------\n");

printf("New list:\n");

list\_del\_more\_less\_value(&lst, -700);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

Original list:

3 <=> -34 <=> 309 <=> 0 <=> 352 <=> 87 <=> -32 <=> -98 <=> 107 <=> -67 <=> -68 <=> 45 <=> 3 <=> 98 <=> 3 <=> 3 <=> 23 <=> 43 <=> 76 <=> 0 <=> 87 <=> 3

-------------------

New list:

**Тест 6**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, -34);

list\_push\_back(&lst, 309);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 352);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, -32);

list\_push\_back(&lst, -98);

list\_push\_back(&lst, 107);

list\_push\_back(&lst, -67);

list\_push\_back(&lst, -68);

list\_push\_back(&lst, 45);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 98);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 23);

list\_push\_back(&lst, 43);

list\_push\_back(&lst, 76);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, 3);

printf("Original list:\n");

list\_print(&lst);

printf("-------------------\n");

printf("New list:\n");

list\_del\_more\_less\_value(&lst, 23);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

Original list:

3 <=> -34 <=> 309 <=> 0 <=> 352 <=> 87 <=> -32 <=> -98 <=> 107 <=> -67 <=> -68 <=> 45 <=> 3 <=> 98 <=> 3 <=> 3 <=> 23 <=> 43 <=> 76 <=> 0 <=> 87 <=> 3

-------------------

New list:

23

**Тест 7**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

Iterator i;

list\_create(&lst);

i = first(&lst);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, -34);

list\_push\_back(&lst, 309);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 352);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, -32);

list\_push\_back(&lst, -98);

list\_push\_back(&lst, 107);

list\_push\_back(&lst, -67);

list\_push\_back(&lst, -68);

list\_push\_back(&lst, 45);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 98);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 3);

list\_push\_back(&lst, 23);

list\_push\_back(&lst, 43);

list\_push\_back(&lst, 76);

list\_push\_back(&lst, 0);

list\_push\_back(&lst, 87);

list\_push\_back(&lst, 3);

printf("Original list:\n");

list\_print(&lst);

printf("-------------------\n");

printf("New list:\n");

list\_del\_more\_less\_value(&lst, 3);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

Original list:

3 <=> -34 <=> 309 <=> 0 <=> 352 <=> 87 <=> -32 <=> -98 <=> 107 <=> -67 <=> -68 <=> 45 <=> 3 <=> 98 <=> 3 <=> 3 <=> 23 <=> 43 <=> 76 <=> 0 <=> 87 <=> 3

-------------------

New list:

3 <=> 3 <=> 3 <=> 3 <=> 3

**Тест 8**

➜ KP3 cat main.c

#include "list.h"

int main(void)

{

List lst;

list\_create(&lst);

printf("Original list:\n");

list\_print(&lst);

printf("-------------------\n");

printf("New list:\n");

list\_del\_more\_less\_value(&lst, 3);

list\_print(&lst);

return 0;

}

➜ KP3 make

➜ KP3 ./main

Original list:

-------------------

New list:

**Вывод**

В данной курсовой работе я реализовал структуру линейный двунаправленный список. До этого я уже писал эту структуру данных, но без итераторов. Итераторы дают большое удобство при навигации по списку, и при создании других функций, где необходимо было бы ходить по списку. Список также можно улучшить использованием барьерного элемента, так как в структуре списка будет всего 2 поля а не 3, и иногда при какой-то проблеме в функции, например если память не выделилась можно перекинуть указатель на барьерный элемент.