Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Курсовая работа

Дисциплина: «Практикум на ЭВМ»

II семестр

Задание 8: «Линейные списки»

|  |  |
| --- | --- |
| Группа: | М8О-108Б-18, №12 |
| Студент: | Коростелев Дмитрий Васильевич |
| Преподаватель: | Поповкин Александр Викторович |
| Оценка: |  |
| Дата: | 16.05.2019 |

Москва, 2019

**Содержание**

1. Задание...............................................................................................................2
2. Алгоритм............................................................................................................2
3. Переменные .......................................................................................................4
4. Тестирование.....................................................................................................4
5. Листинг программы..........................................................................................5
6. Протокол..........................................................................................................12
7. Заключение.......................................................................................................31
8. Список использованной литературы.............................................................31

**Задание**

Составить и отладить программу на языке Си для обработки линейного списка заданной ориентации с отображением списка на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного и четырех стандартных действий: печать списка, вставка нового элемента в список, удаление элемента из списка, подсчет длины списка. Тип элемента списка: целый. Вид списка: кольцевой однонаправленный. Нестандартное действие: выполнить циклический сдвиг на один элемент вперед.

**Алгоритм**

Прежде чем приступать к разбору алгоритма, определимся со способом задания линейного закольцованного списка. В закольцованном списке, чтобы не потерять его начало требуется хранить, либо размер списка, то есть выделить для этого специальную переменную, либо хранить ссылку на последний элемент списка, чтобы во время прохода в линейному списку сравнивать адреса на итерациях и останавливаться, когда специальная временная переменная зайдет в ячейку памяти отведенную под конец списка. В своем решении, я использовал второй способ, так как, данный способ пусть и требует немного больше памяти, однако, сам процесс написания программы становится проще, так как не нужно следить за размером переменной размера списка, однако у данного способа есть большой недостаток, требуется программно обрабатывать случаи, когда в списке находится 0 или 1 элемент.

Сам список является структурой list, у которой есть два элемента head и tail. В свою очередь head и tail являются указателями на структуру element. Структура element состоит из value - целочисленная переменная и next – указателя на структуру element, next указывает на след. элемент линейной списка.

В основе каждого алгоритма будет лежать один и тот же метод прохода по списку, в котором также отдельно, как исключения обрабатываются пустые списки и списки с одним элементом. Печать списка происходит по след алгоритму – если в списке 0 элементов, то выводим строку: «List is clear», если в списке один элемент выводит значение списка (назовем список – main) main->head->value (main – указатель на структуру list). Если же в списке больше одного элемента, создаем временную переменную, в которой, по мере итеративного прохождения по списку с помощью цикла while, будем проходит по списку, пока не зайдем в ячейку памяти отведенную для переменной tail попутно выводя значения value. Вставку нового элемента будем производить так, если в списке ноль элементов, записываем новое число в head, если в списке один элемент, записываем новое число в tail, если в списке больше одного элемента, перезаписываем указатель tail на новую ячейку памяти, туда записываем число, закольцовываем список, и присваиваем tail указатель на этот новый элемент. Процесс удаления элемента списка, абсолютно противоположен процессу вставки. Если 0 элементов, ничего не делаем, если 1, очищаем переменную head, если 2, очищаем переменную tail и указатель на след. элемент в head. Если больше 2, то ищем элемент списка, который стоит перед tail итеративным проходом по всему списку. Очищаем tail, в найденном элементе перезаписываем указатель на head и перезаписываем указатель на tail на найденный элемент. Циклический сдвиг осуществляется очень просто, нужно перезаписать указатели head и tail на след за ними элементы.

**Переменные**

Таблица 1

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Назначение** |
| main | struct list\* | Линейный список |
| head | struct element\* | Указатель на первый элемент списка |
| tail | struct element\* | Указатель на последний элемент списка |
| value | int | Значение узла списка, часть структуры element |
| next | struct element\* | Указатель на след. элемент списка |
| t\_el | struct element\* | Временный указатель на структуру element, используется для прохода по списку |
| menu | int | Используется в конструкции switch-case |

**Тестирование**

Во время тестирования следует проверит, корректно ли программа обрабатывает списки с 1, 2, более 2 элементов. Не происходит ли ошибка сегментации во время удаления элемента из списка, не остается ли нигде лишних указателей (хвостов), правильно ли происходит вставка новых элементов, не пропадает ли закольцованность списка после вставки или удаление и т.п.

**Листинг программы**

*Function.c*

#include"header.h"

#include<stdio.h>

#include<malloc.h>

void add\_element(struct list\* main) {

int value;

if ((main->head) == 0) {

scanf("%d", &value);

main->head = (struct element\*)malloc(sizeof(struct element));

main->head->value = value;

main->head->next = 0;

return;

}

if ((main->head) != 0 && (main->tail) == 0) {

scanf("%d", &value);

main->tail = (struct element\*)malloc(sizeof(struct element));

main->tail->value = value;

(main->tail->next) = (main->head);

main->head->next = main->tail;

return;

}

struct element\* t\_el;

t\_el = main->head;

while ((t\_el->next) != (main->tail)) {

(t\_el) = (t\_el->next);

}

t\_el->next = 0;

t\_el->next = (struct element\*)malloc(sizeof(struct element));

t\_el->next->value = main->tail->value;

(t\_el->next->next) = (main->tail);

scanf("%d", &value);

main->tail->value = value;

}

int size\_list(struct list\* main) {

if (main->head == 0) {

return 0;

}

if (main->head != 0 && main->tail == 0) {

return 1;

}

int size\_list = 0;

struct element\* t\_el;

t\_el = main->head;

while (t\_el != main->tail)

{

size\_list++;

t\_el = (t\_el->next);

}

size\_list++;

return size\_list;

}

void print\_list(struct list\* main) {

if (main->head == 0) {

printf("List is clear\n");

return;

}

if (main->head != 0 && main->tail == 0) {

printf("%d\n", main->head->value);

return;

}

struct element\* t\_el;

t\_el = (main->head);

while (t\_el != main->tail) {

printf("%d\n", t\_el->value);

t\_el = (t\_el->next);

}

printf("%d\n",t\_el->value);

}

void delete\_element(struct list\* main) {

if (main->head == 0) {

printf("Error: list is clear\n");

return;

}

if (main->head != 0 && main->tail == 0) {

main->head->next = 0;

free(main->head);

main->head = 0;

return;

}

struct element\* t\_el;

t\_el = main->head;

while (t\_el->next != main->tail) {

t\_el = (t\_el->next);

}

if(t\_el == main->head){

free(main->tail);

main->tail = 0;

main->head->next = 0;

return;

}

main->tail = 0;

main->tail = t\_el;

main->tail->value = t\_el->value;

main->tail->next = main->head;

}

void l\_shift(struct list\* main){

if(main->head == NULL){

printf("Error: List is clear...\n");

return;

}

if(main->head != 0 && main->tail == 0)

return;

main->head = main->head->next;

main->tail = main->tail->next;

}

*header.h*

struct list {

struct element\* head;

struct element\* tail;

};

struct element {

int value;

struct element\* next;

};

void add\_element(struct list\* main);

void print\_list(struct list\* main);

void delete\_element(struct list\* main);

void l\_shift(struct list\* main);

int size\_list(struct list\* main);

*kp8.c*

#include<stdio.h>

#include<malloc.h>

#include"header.h"

int main(void) {

struct list\* main;

main = (struct list\*)malloc(sizeof(struct list));

main->head = 0;

main->tail = 0;

while (1) {

int menu;

printf("1: Print List\n");

printf("2: Print size of List\n");

printf("3: Add new element to List\n");

printf("4: Delete element of List\n");

printf("5: Shift og List\n");

printf("6: Exit...\n");

printf("Change the action: ");

scanf("%d", &menu);

switch (menu) {

case 1:

printf("\nList:\n");

print\_list(main);

printf("\n");

break;

case 2:

printf("size of List: %d\n", size\_list(main));

break;

case 3:

add\_element(main);

break;

case 4:

delete\_element(main);

break;

case 5:

l\_shift(main);

break;

case 6:

return 0;

default:

printf("Error: wrong change...\n");

break;

}

}

return 0;

}

*MakeFileKP8*

CC = cc #имя компилятора

CCFLAGS = -c -Wall #ключи компилятора

RESULT\_NAME = kp8.out

all: $(RESULT\_NAME) clean

$(RESULT\_NAME): kp8.o function.o

$(CC) kp8.o function.o -o $(RESULT\_NAME)

kp8.o: kp8.c

$(CC) $(CCFLAGS) header.h kp8.c

function.o: function.c

$(CC) $(CCFLAGS) header.h function.c

clean:

rm \*.o \*.gch

**Протокол**

dmitry@dmitry-VirtualBox:~/lubs/KP8$ ls

function.c header.h kp8.c MakeFileKP8

dmitry@dmitry-VirtualBox:~/lubs/KP8$ make -f MakeFileKP8

cc -c -Wall header.h kp8.c

kp8.c: In function ‘main’:

kp8.c:7:15: warning: ‘main’ is usually a function [-Wmain]

struct list\* main;

^~~~

cc -c -Wall header.h function.c

cc kp8.o function.o -o kp8.out

rm \*.o \*.gch

dmitry@dmitry-VirtualBox:~/lubs/KP8$ ls

function.c header.h kp8.c kp8.out MakeFileKP8

dmitry@dmitry-VirtualBox:~/lubs/KP8$ ./kp8.out

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

List is clear

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 0

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

Error: list is clear

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

12

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

12

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 1

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

12

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

List is clear

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 0

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

123

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

123

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 2

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

123

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 1

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

23

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

145

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

23

145

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

145

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

145

3421

23

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

23

145

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

23

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

234

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

1

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 3

45

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

3421

23

234

1

45

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 2

size of List: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 5

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

234

1

45

3421

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

234

1

45

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

234

1

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

234

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

23

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 4

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 1

List:

List is clear

1: Print List

2: Print size of List

3: Add new element to List

4: Delete element of List

5: Shift og List

6: Exit...

Change the action: 6

dmitry@dmitry-VirtualBox:~/lubs/KP8$

**Заключение**

Благодаря данному курсовому проекту студент может на своем опыте понять и усвоить основные способы реализации различных видов линейных списков, а также механизмы их обработки и печати. Также во время работы с линейными списками, становится понятно, что для решения какой-либо нетривиальной задачи, человек не сможет обойтись без комбинирования базовых типов данных и составления, новых, составных типов, так как с помощью них, возможно сохранение семантики сообщения без разделения самого сообщения на отдельные части.

**Список использованной литературы**

1. Методические материалы: “Практикум по циклу дисциплин «Информатика» часть 2. 8 факультет, 1 Курс, 2 семестр 2012/13. 13стр. Зайцев В.Е.
2. Динамическое выделение памяти, динамические массивы [Эдектронный ресурс]// <https://prog-cpp.ru/>URL: https://prog-cpp.ru/c-alloc//