# Московский Авиационный институт (Национальный исследовательский университет)

Факультет №8

# «Информационные технологии и прикладная математика»

**Кафедра 806**

**«Вычислительная математика и программирование»**

**Курсовая работа**

по теме

«Разреженные матрицы»

|  |  |
| --- | --- |
| Студент: | Петров И. О. |
| Группа: | М8О-106Б-21 |
| Преподаватель: | Дубинин А.В. |
| Подпись: |  |
| Оценка: |  |

Москва, 2022

# 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 действия и четырёх стандартных действий:

# Нестандартное действие:

Поменять местами 2-й и предпоследний элементы списка

# Стандартные действия:

Печать списка

Вставка нового элемента в список Удаление элемента из списка Подсчёт длины списка

# Вид списка:

Двунаправленный линейный

# Тип элемента списка:

Вещественный

**Основная часть**

Необходимая теория

Список - структура данных, состоящая из элементов одного типа, связанных с помощью указателей. Двунаправленный означает, что двигаться можно в обе стороны, но, в отличие от кольцевого списка, в линейном нельзя перейти с последнего элемента на первый.

Описание структуры

Создаётся структура списка и записи в списке. Структура списка содержит указатели на первый и последний элементы списка, структура записи

содержит само значение элемента списка, а также указатели на предыдущий и следующий элементы.

Общий метод решения

После создания структуры списка создаётся структура итератора. Благодаря ней можно писать одинаковые функции для вариантов на указателях и на векторах. Пишутся функции для печати, удаления, ввода и подсчёта длины списка.

Алгоритм нестандартного действия

Алгоритм достаточно простой. Создается временная запись, которая будет хранить значение одного из элементов, которые нужно поменять местами. После значение одной записи приравнивают другой, а второй записи присваивают значение временной записи. Временная запись удаляется.

Функциональное назначение

Программа предназначена для демонстрации использования метода хранения линейного списка на указателях и работы с ним. Также она демонстрирует использование итераторов.

Последовательность действий

После запуска исполняемого файла выводится подсказка с командами. При вводе этих команд будет выполнена та или иная функция, которая тоже может потребовать ввести необходимые значения. Завершить работу программы можно командой “exit”.

#ifndef LIST\_H #define LIST\_H

#include <stdio.h> typedef double item;

typedef struct \_list\_node list\_node; typedef struct \_list list;

struct \_list\_node

{

# Текст программы list.h

item data; list\_node \*next; list\_node \*prev;

};

struct \_list

{

list\_node \*head; list\_node \*tail;

};

list \*list\_create();

void list\_insert(list \*lst, item val);

void list\_fprint(list \*lst, FILE \*stream); list\_node \*get\_by\_id\_list(list \*lst, int id, int size); void delete\_elem(list \*lst, int id, int size);

#endif

#include "list.h" #include <stdio.h> #include <stdlib.h>

list \*list\_create()

{

# list.c

list \*lst = (list\*) malloc(sizeof(list)); lst->head = NULL;

lst->tail = NULL; return lst;

}

void list\_insert(list \*lst, item val)

{

list\_node \*node = (list\_node\*) malloc(sizeof(list\_node)); node->next = NULL;

node->prev = lst->tail; node->data = val;

if (lst->tail != NULL) { lst->tail->next = node;

}

lst->tail = node;

if (lst->head == NULL) { lst->head = node;

}

}

void list\_fprint(list \*lst, FILE \*stream)

{

fprintf(stream, "Your list**\n**"); list\_node \*node = lst->head; while(node != NULL) {

fprintf(stream, "[%lf]->", node->data); node = node->next;

}

printf("**\n**");

}

list\_node \*get\_by\_id\_list(list \*lst, int id, int size)

{

list\_node \*node = NULL; int pos;

if (id < size / 2) { pos = 0;

node = lst->head;

while (node != NULL && pos < id) { node = node->next;

pos++;

}

} else {

pos = size - 1; node = lst->tail;

while (node != NULL && pos > id) { node = node->prev;

pos--;

}

}

return node;

}

void delete\_elem(list \*lst, int id, int size)

{

list\_node \*node = get\_by\_id\_list(lst, id, size); if (node == NULL) {

printf("Error**\n**"); return;

}

if (node->prev != NULL) {

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

}

if (node->next != NULL) {

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

}

if (node->prev == NULL) { lst->head = node->next;

}

if (node->next == NULL) { lst->tail = node->prev;

}

free(node);

}

# iterator.h

#ifndef ITERATOR\_H #define ITERATOR\_H

#include "list.h"

typedef struct \_iterator iterator;

struct \_iterator

{

list\_node \*node;

};

iterator \*iterator\_create(list \*lst); void iterator\_next(iterator \*it); void iterator\_prev(iterator \*it); list\_node \*iterator\_get(iterator \*it); int iterator\_length(list \*lst); void change\_places(list \*lst);

#endif

#include "iterator.h" #include "list.h" #include <stdio.h> #include <stdlib.h>

iterator \*iterator\_create(list \*lst)

{

# iterator.c

iterator \*it = (iterator \*)malloc(sizeof(iterator)); it->node = lst->head;

return it;

}

void iterator\_next(iterator \*it)

{

if(it->node->next) {

it->node = it->node->next;

} else {

printf("It is the last element**\n**");

}

}

void iterator\_prev(iterator \*it)

{

if(it->node->prev) {

it->node = it->node->prev;

} else {

printf("It is the first element**\n**");

}

}

list\_node \*iterator\_get(iterator \*it)

{

return it->node;

}

int iterator\_length(list \*lst)

{

if (lst->head == NULL && lst->tail == NULL) { return 0;

}

iterator \*len = (iterator \*)malloc(sizeof(iterator)); len->node = lst->head;

int count = 1;

while (len->node->next != NULL) { count++;

len->node = len->node->next;

}

return count;

}

void change\_places(list \*lst)

{

int size = iterator\_length(lst); if (size < 2 || size == 3) {

return;

}

item t = lst->head->next->data;

lst->head->next->data = lst->tail->prev->data; lst->tail->prev->data = t;

}

# main.c

#include <stdio.h> #include <stdlib.h> #include "list.h" #include "iterator.h" #include <string.h>

int main(void)

{

char arg[30]; list \*lst = list\_create();

iterator \*it = iterator\_create(lst);

printf("Enter 'add\_to\_back' to insert element in the end of list**\n**"); printf("Enter 'exit' to finish program**\n**");

printf("Enter 'func' to change second and one before last elements**\n**"); printf("Enter 'length' to show number of elemtnts**\n**");

printf("Enter 'del' to delete one element**\n**"); printf("Enter 'print' to show the list**\n**"); while (strcmp(arg, "exit") != 0) {

scanf("%s", arg);

if (strcmp(arg, "add\_to\_back") == 0 || strcmp(arg, "add") == 0) { printf("Enter number of list's elements**\n**");

int i; double g;

scanf("%d", &i);

for (int j = 0; j < i; j++) {

printf("Enter value**\n**"); scanf("%lf", &g); list\_insert(lst, g);

}

}

else if (strcmp(arg, "func") == 0) { change\_places(lst);

}

else if (strcmp(arg, "length") == 0) { printf("%d**\n**", iterator\_length(lst));

}

else if (strcmp(arg, "del") == 0) { int id;

printf("Enter id of node, beginning from 1**\n**"); scanf("%d", &id);

int size = iterator\_length(lst); if (id > size || id <= 0) {

printf("There is no such node**\n**");

} else { id--;

delete\_elem(lst, id, size);

}

}

else if (strcmp(arg, "print") == 0) { if (iterator\_length(lst) == 0) { printf("List is empty**\n**");

} else {

list\_fprint(lst, stdout);

}

} else if (strcmp(arg, "exit") == 0) { return 0;

} else {

printf("There is no such a command**\n**");

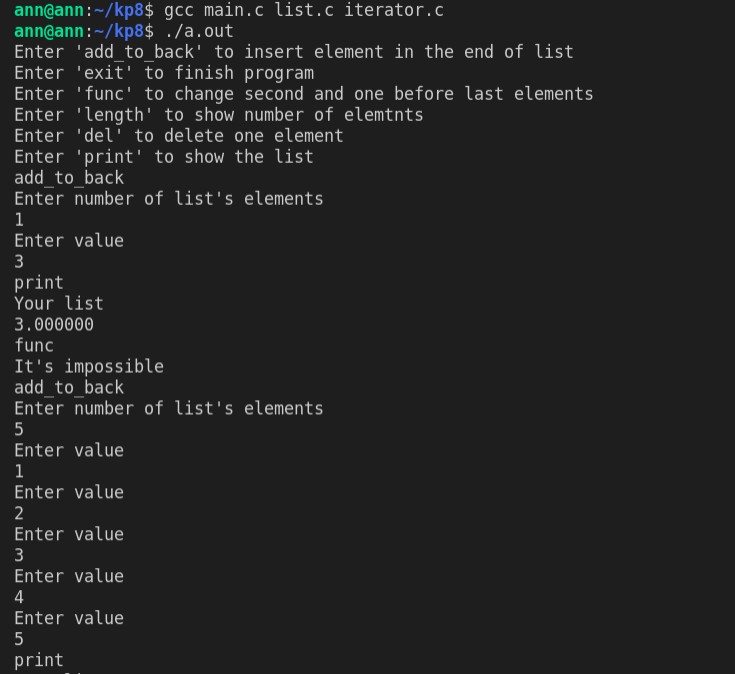
}

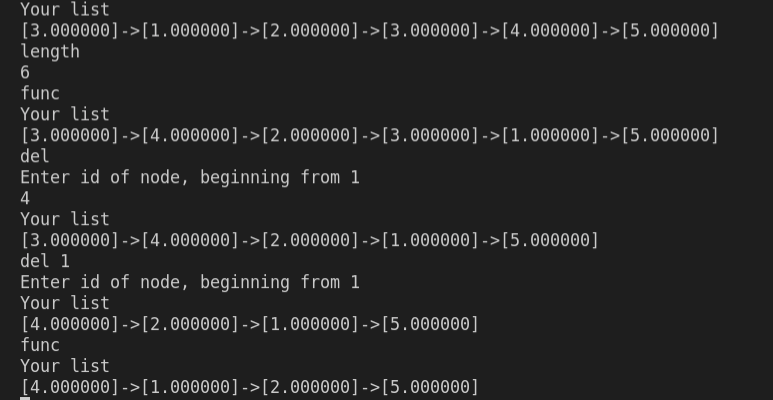
}

return 0;

}

# Протокол





**Заключение**

В задании №8 курсовой работы я познакомился с линейными списками. Списки – классическая структура данных. Мне также понравилась практика создания итераторов – чувствуются зачатки командной разработки, когда нужно в первую очередь думать о других разработчиках. Смысл в том, чтобы никто не вдавался в подробности, как работает на низком уровне программа. Это даёт представление о понятии абстракция.

# Список литературы

1. Методические указания к выполнению курсовых работ. Зайцев В. Е.
2. <https://prog-cpp.ru/data-dls/>
3. https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0% BD%D1%8B%D0%B9\_%D1%81%D0%BF%D0%B8%D1%81%D0%BE

%D0%BA