Московский авиационный институт

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

КУРСОВОЙ ПРОЕКТ

по курсу “Практикум на ЭВМ”

II семестр

Тема “Линейные списки”

Работу выполнила:

Студентка группы М8О-106Б-21

Деревянко Екатерина Андреевна

Работу проверил:

Старший преподаватель

Дубинин Алексей Владимирович

Москва 2022

Оглавление

[Введение 3](#_Toc768)

[Теоретическая часть 4](#_Toc9699)

[Итераторы 4](#_Toc17702)

[Кольцевой двунаправленный список 5](#_Toc26636)

[Практическая часть 6](#_Toc29696)

[Об алгоритме 6](#_Toc20001)

[Протокол 6](#_Toc29172)

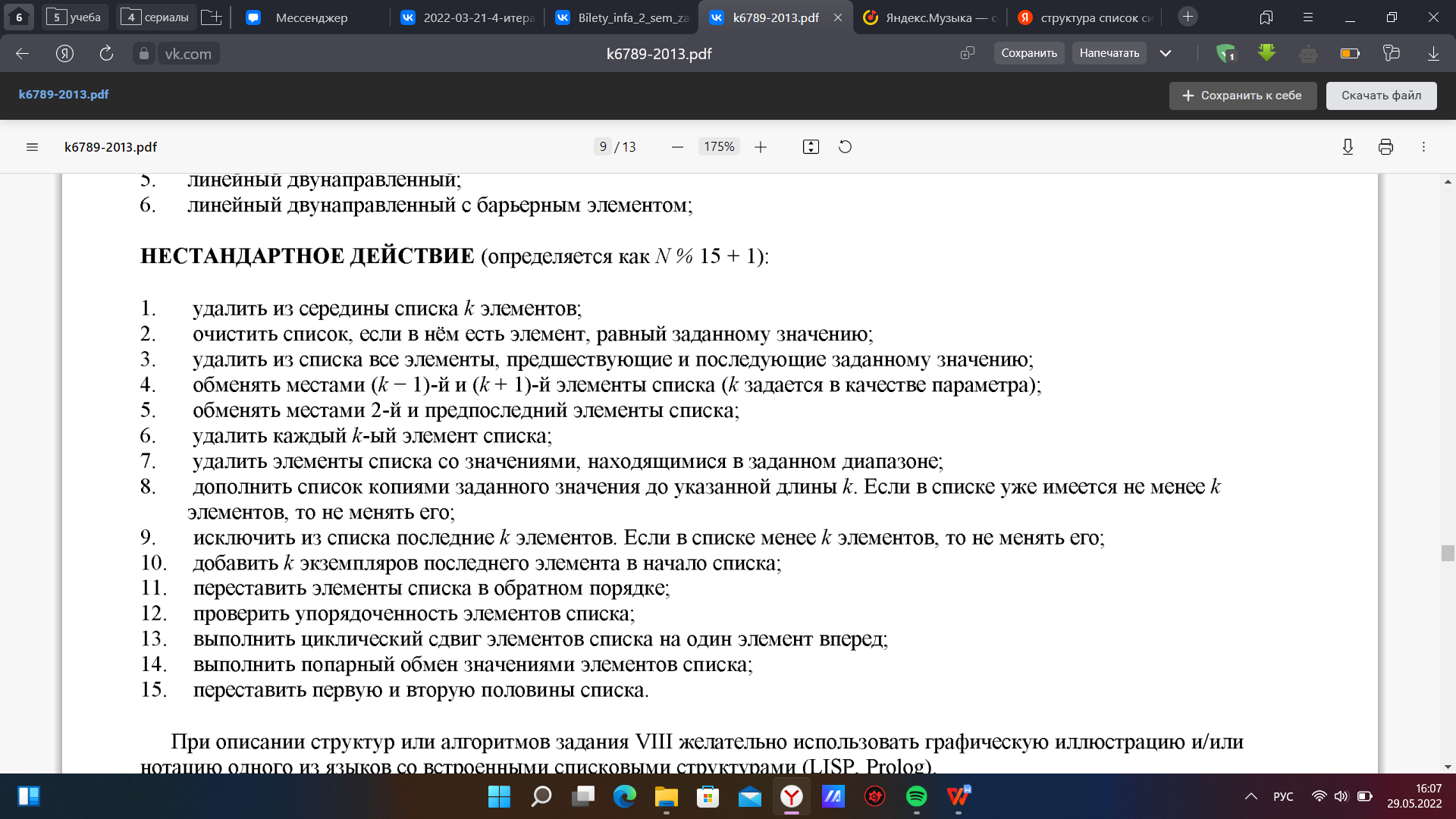
[Заключение 15](#_Toc11074)

[Список источников 16](#_Toc7858)

# Введение

Помимо таких динамических структур данных как стеки, деки и очереди существуют списки. Их надо уметь реализовывать и работать с ними. Эта работа будет посвящена кольцевому двунаправленному списку.

*Вариант:*



# **Теоретическая часть**

## Итераторы

Итератор - это объект, который позволяет перемещаться (итерироваться) по элементам некоторой последовательности.

При работе с массивом для обращению к какому-либо элементу используются индексы, но некоторые структуры не позволяют сделать это эффективно, но позволяют перебирать элементы подряд. С помощью итераторов у пользователя появляется возможность обращаться к элементу структуры при сокрытии ее внутреннего устройства.

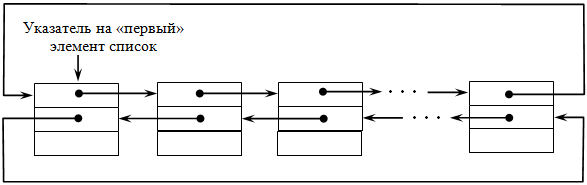
Итераторы удобны при работе со списками и деревьями, и дают возможность не только обратиться к определенному элементу, но и добавить новый элемент в любое место или, например,  проверить отсутствия пропусков элементов или защитить от повторного перебора одного и того же элемента.

## Кольцевой двунаправленный список

Циклический (кольцевой) список – это *структура данных*, представляющая собой последовательность элементов, последний элемент которой содержит *указатель* на первый *элемент списка*, а первый (в случае *двунаправленного* списка) – на последний.

Основная особенность такой организации состоит в том, что в этом списке нет элементов, содержащих пустые указатели, и, следовательно, нельзя выделить крайние элементы.

Циклический двунаправленный список похож на линейный двунаправленный список, но его любой элемент имеет два указателя, один из которых указывает на следующий элемент в списке, а второй указывает на предыдущий элемент.



Основные операции, осуществляемые с циклическим двунаправленным списком:

* создание;
* печать;
* вставка элемента;
* удаление элемента;
* поиск элемента;
* проверка пустоты;
* удаление списка.

# Практическая часть

## Об алгоритме

В программе реализован двунаправленный кольцевой список на указателях, нестандартной функцией которого является перемена местами второго и предпоследнего элементов. Для реализации функций списка используются итераторы. Сложность добавления элемента O(1), сложность вывода списка O(n), где n - количество элементов списка, сложность обмена двух элементов - O(1).

## Протокол

*Реализация списка*

#include "list.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

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

{

iterator\* it = (iterator\*)malloc(sizeof(iterator));

it->nod = lst->head;

return it;

}

iterator\* iterator\_next(iterator\* it)

{

if (it != NULL) {

it->nod = it->nod->next;

return it;

}

return NULL;

}

iterator\* iterator\_prev(iterator\* it)

{

if (it != NULL) {

it->nod = it->nod->prev;

return it;

}

return NULL;

}

node\* iterator\_get(iterator\* it)

{

return it->nod;

}

list\* list\_create()

{

list\* lst = (list\*) malloc(sizeof(list));

lst->head = lst->tail = NULL;

lst->size = 0;

return lst;

}

void print(list\* lst)

{

if (lst == NULL || lst->head == NULL) {

return;

}

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

if (iterator\_get(iter) != NULL) {

print\_colors(iterator\_get(iter)->data);

printf(" ");

}

iter = iterator\_next(iter);

while (iterator\_get(iter) != lst->head) {

print\_colors(iterator\_get(iter)->data);

printf(" ");

iter = iterator\_next(iter);

}

free(iter);

printf("\n");

}

void back\_insert(list\* lst, value\_type c)

{

if (lst == NULL) {

printf("The list not exists\n");

return;

}

node\* new = (node\*)malloc(sizeof(node));

if (lst->head == NULL) {

lst->head = new;

lst->head->data = c;

lst->tail = lst->head;

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

lst->size++;

} else {

lst->tail->next = new;

new->prev = lst->tail;

new->next = lst->head;

lst->tail = new;

lst->tail->data = c;

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

lst->size++;

}

}

void front\_insert(list\* lst, value\_type c)

{

if (lst == NULL) {

printf("The list not exists\n");

return;

}

node\* new = (node\*)malloc(sizeof(node));

if (lst->head == NULL) {

lst->head = new;

lst->head->data = c;

lst->tail = lst->head;

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

lst->size++;

} else {

lst->head->prev = new;

new->data = c;

new->next = lst->head;

lst->tail->next = new;

lst->head = new;

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

lst->size++;

}

}

void delete(list\* lst, int k)

{

if (count(lst) == 0) {

printf("The list not exists\n");

return;

}

if (k >= count(lst)) {

printf("This node not exists\n");

return;

}

if (count(lst) == 1 && k == 0) {

lst->tail = lst->head = NULL;

lst->size--;

return;

}

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

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

iter = iterator\_next(iter);

}

node\* first = iterator\_get(iter);

if (first != lst->head && first != lst->tail) {

first->prev->next = first->next;

first->next->prev = first->prev;

free(first);

lst->size--;

} else if (first == lst->head) {

lst->head = first->next;

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

first->next->prev = lst->tail;

free(first);

lst->size--;

} else if (first == lst->tail) {

lst->tail = first->prev;

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

first->prev->next = lst->head;

free(first);

lst->size--;

}

free(iter);

}

int count(list\* lst)

{

return lst->size;

}

void destroy(list\* lst)

{

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

iter = iterator\_next(iter);

while (iterator\_get(iter) != lst->head) {

node \*pi = iterator\_get(iter);

iter = iterator\_next(iter);

free(pi);

}

free(lst->head);

free(iter);

lst->head = 0;

printf("\n");

}

void task(list \*a) {

if (count(a) > 3) {

node \*x = a->head->next;

node \*y = a->head->prev->prev;

value\_type tmp = x->data;

x->data = y->data;

y->data = tmp;

} else if (count(a) == 2) {

node \*x = a->head;

node \*y = a->head->prev;

value\_type tmp = x->data;

x->data = y->data;

y->data = tmp;

} else {

return;

}

}

Основная программа

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include "list.h"

int main(int argc, char const \*argv[])

{

printf("\nWrite:\nFRONT to add elem in list front\n");

printf("BACK to add elem in list back\n");

printf("DELETE to delete elem from list\n");

printf("PRINT to show list\n");

printf("SIZE to show size of list\n");

printf("TASK to swap second and penultimate elems\n");

printf("COLORS to show enum colors list\n");

printf("HELP to see commands again\n");

printf("BYE to exit\n\n");

list\* l;

int key;

l = list\_create();

char str[100];

char elem[100];

char \*choose;

char \*el;

bool flag = true;

while (flag) {

choose = gets\_(str);

if (!feof(stdin)) {

if (!strcmp(choose, "FRONT")) {

printf("Write color from enum to add it to list:\n");

el = gets\_(elem);

if (!feof(stdin)) {

if (str\_to\_enum(el) != -1) {

front\_insert(l, str\_to\_enum(el));

} else {

printf("This is no color\n");

}

}

} else if (!strcmp(choose, "BACK")) {

printf("Write color from enum to add it to list:\n");

el = gets\_(elem);

if (!feof(stdin)) {

if (str\_to\_enum(el) != -1) {

back\_insert(l, str\_to\_enum(el));

} else {

printf("This is no color\n");

}

}

} else if (!strcmp(choose, "DELETE")) {

printf("Enter key:\n");

scanf("%d", &key);

delete(l, key);

} else if (!strcmp(choose, "PRINT")) {

printf("\n");

print(l);

} else if (!strcmp(choose, "SIZE")) {

printf("Size: %d\n", count(l));

} else if (!strcmp(choose, "TASK")) {

task(l);

} else if (!strcmp(choose, "COLORS")) {

printf("Colors enum list:\n");

for (int i = 0; i < 43; i++) {

printf("%d: ", i);

print\_colors(i);

printf("\n");

}

} else if (!strcmp(choose, "BYE")) {

if (count(l) != 0) {

destroy(l);

}

flag = false;

} else if (!strcmp(choose, "HELP")) {

printf("\nWrite:\nFRONT to add elem in list front\n");

printf("BACK to add elem in list back\n");

printf("DELETE to delete elem from list\n");

printf("PRINT to show list\n");

printf("TASK to swap second and penultimate elems\n");

printf("COLORS to show enum colors list\n");

printf("HELP to see commands again\n");

printf("BYE to exit\n\n");

} else if (!strcmp(choose, "")) {

continue;

} else {

printf("Wrong command\n");

continue;

}

} else {

if (count(l) != 0) {

destroy(l);

}

break;

}

}

free(l);

return 0;

}

# Заключение

Список - одна из динамических структур, которая может быть реализована несколькими способами, один из которых кольцевой двунаправленный. Для «перемещения» по нему используются итераторы.

# Список источников

1. О кольцевом двунаправленном списке

<https://intuit.ru/studies/courses/648/504/lecture/11459>

1. Об итераторах

<https://www.perl.com/pub/2005/06/16/iterators.html/>