Министерство общего и профессионального

образования Российской Федерации

Пермский государственный технический университет

Лабораторная работа №11

"Информационные динамические структуры"

Вариант №12

Выполнила студентка группы РИС 23-3б:

Федорова О.И.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2024

1. Постановка задачи. Записи в линейном списке содержат ключевое поле типа \*char. Сформировать двунаправленный список. Удалить из него элементы, с одинаковыми ключевыми полями. Добавить заданным ключевым полем.

Для каждого вариант разработать следующие функции:

1) Создание списка.

2) Добавление элемента в список (в соответствии со

своим вариантом).

3) Удаление элемента из списка (в соответствии со

своим вариантом).

4) Печать списка.

5) Запись списка в файл.

6) Уничтожение списка.

7) Восстановление списка из файла.

**Решение однонаправленным списком.**

1. Анализ задачи:
2. Создать структуру, состоящую из головы, которая изначально равна нулевым данным.
3. Создать функцию, которая будет добавлять и связывать новые элементы в цепочке, меняя голову.
4. Создать функцию, которая при вызове будет выводить массив.
5. Создать функцию для удаления элементов с одинаковыми ключевыми полями. Создаются две переменные, которые будут проходиться и сравнивать все элементы массива и искать совпадение ключевых полей. После нахождения совпадений начинается несколько проверок (является ли какой-либо из элементов головой или же стоят ли они рядом с друг другом), попадая в нужное условие совпадающие элементы удаляются, а разрыв между элементами устранится.
6. Создать функцию, которая находит элемент с заданным ключом, и связывает его с новым элементов (включает элемент в массив).
7. Создать функцию добавления массива в файл.
8. Создать функцию удаления массива из динамической памяти.
9. Создать функцию для восстановления массива из файла.
10. Код программы:

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

struct node {

char data;

node\* next = nullptr;

};

struct List {

node\* head = nullptr;

node\* teil = nullptr;

};

void pushBack(List& list, const int& data) { //добавляю новый элемент в конец списка

node\* new\_node = new node; //создаю новый динамический узел

new\_node->data = data; //присваиваю полю узла данные

if (list.head == nullptr) { //если список пустой

list.head = new\_node; //новый узел - головной узел списка

list.teil = new\_node; //новый узел - хвостовой узел списка

}

else { // если список не пуcтой

list.teil->next = new\_node; //связываю новый узел с хвостовым

list.teil = new\_node; //меняю хвостовой узел на новый

}

}

void printt(List& list) { // функция для вывода списка

cout << endl << "Текущий список:" << endl;

node\* cur = list.head;

if (cur != nullptr) {

while (cur != nullptr) { // пока не дойдем до конца списка

cout << cur->data << " ";

cur = cur->next; // переход к следующему элементу

}

cout << endl << endl;

delete cur; //очищаю память

}

else {

cout << "Список пустой!";

}

}

void delete\_el(List& list, char t) { //удаление элемента

if (list.head != nullptr) { //если список НЕ пустой

node\* cur = list.head;

bool fl = true;

while (cur != nullptr) { //пока не дойду до конца

node\* cur2 = cur->next;

while (cur2 != nullptr) {

if (cur->data == cur2->data) {

if (cur == list.head) { //если надо удалить первый элемент

if (cur->next != cur2) {

node\* new\_Head = list.head->next;

delete list.head; //удаляю текущий головной элемент

list.head = new\_Head; //присваиваю головному элементу новый элемент

cur = list.head;

node\* before\_deletion2 = list.head;

while (before\_deletion2->next != cur2) {

before\_deletion2 = before\_deletion2->next;

}

before\_deletion2->next = cur2->next; //связываю узлы

cur2 = before\_deletion2;

fl = false;

break;

}

else {

node\* new\_Head = cur2->next;

delete list.head; //удаляю текущий головной элемент

list.head = new\_Head; //присваиваю головному элементу новый элемент

cur = nullptr;

cur2 = nullptr;

fl = false;

break;

}

}

else { //если удалять НЕ первый элемент

if (cur->next != cur2) {

node\* before\_deletion = list.head;

node\* before\_deletion2 = list.head;

while (before\_deletion->next != cur) {

before\_deletion = before\_deletion->next;

}

while (before\_deletion2->next != cur2) {

before\_deletion2 = before\_deletion2->next;

}

before\_deletion->next = cur->next; //связываю узлы

cur = before\_deletion;

before\_deletion2->next = cur2->next; //связываю узлы

cur2 = before\_deletion2;

fl = false;

break;

}

else {

node\* before\_deletion2 = list.head;

while (before\_deletion2->next != cur) {

before\_deletion2 = before\_deletion2->next;

}

before\_deletion2->next = cur2->next; //связываю узлы

cur2 = before\_deletion2;

fl = false;

break;

}

}

if (fl = false) break;

}

else {

cur2 = cur2->next; //перехожу на следующий элемент

}

}

if (fl == false) break;

cur = cur->next;

}

}

}

void add(List& list, const int& data, char n) {

node\* new\_el = new node; // создание нового элемента

new\_el->data = data; // присваивание значения

node\* cur = list.head;

while (cur->data != n) { // поиск заданного элемента

cur = cur->next; // переход к следующему элементу

}

new\_el->next = cur->next; // связь узлов

cur->next = new\_el; // связь узлов

}

void ffile(List& list, ofstream& file) { //запись данных в файл

if (list.head != nullptr) {//если список не пустой

node\* cur = list.head; //указатель на первый элемент

while (cur != nullptr) { //пока не дойду до конца

file << cur->data << endl;

cur = cur->next; //перехожу на следующий узел

}

}

}

void del(List& list) { //освобожденеи памяти

node\* ptr\_node = list.head;

if (ptr\_node != nullptr) {

while (ptr\_node != nullptr) {

node\* new\_Head = list.head->next;

delete list.head; //удаляю текущий головной элемент

list.head = new\_Head; //присваиваю головному элементу новый элемент

ptr\_node = list.head;

}

}

}

void vost(List& list, ifstream& file) {

string p;

if (getline(file, p)) {

pushBack(list, p[0]); // добавление символа в строку

node\* new\_nod = new node;

new\_nod->data = p[0];

while (getline(file, p)) {

pushBack(list, p[0]); // добавление символа в строку

}

}

}

int main23() {

setlocale(LC\_ALL, "ru");

system("chcp 1251");

int k;

char n;

cout << "Введите количество элементов в списке: " << endl;

cin >> k;

List list; //инициализирую список

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

char a;

cout << "Введите элемент списка: " << endl;

cin >> a;

pushBack(list, a);

}

printt(list); // вывод списка

cout << "С удалением элемента: " << endl;

delete\_el(list, k); // удаление элементов

printt(list); // вывод списка

char t;

cout << "С добавлением элемента: " << endl;

cout << "Введите символ, после которого нужно добавить элемент: " << endl;

cin >> n;

cout << "Введите данные для нового элемента: " << endl;

cin >> t;

add(list, t, n); // добавление нового элемента

printt(list); // вывод списка

ofstream f;

f.open("neww.txt");

ffile(list, f); // добавление элемента в файл

del(list); // удаление списка

printt(list); // вывод списка

ifstream ff;

ff.open("neww.txt");

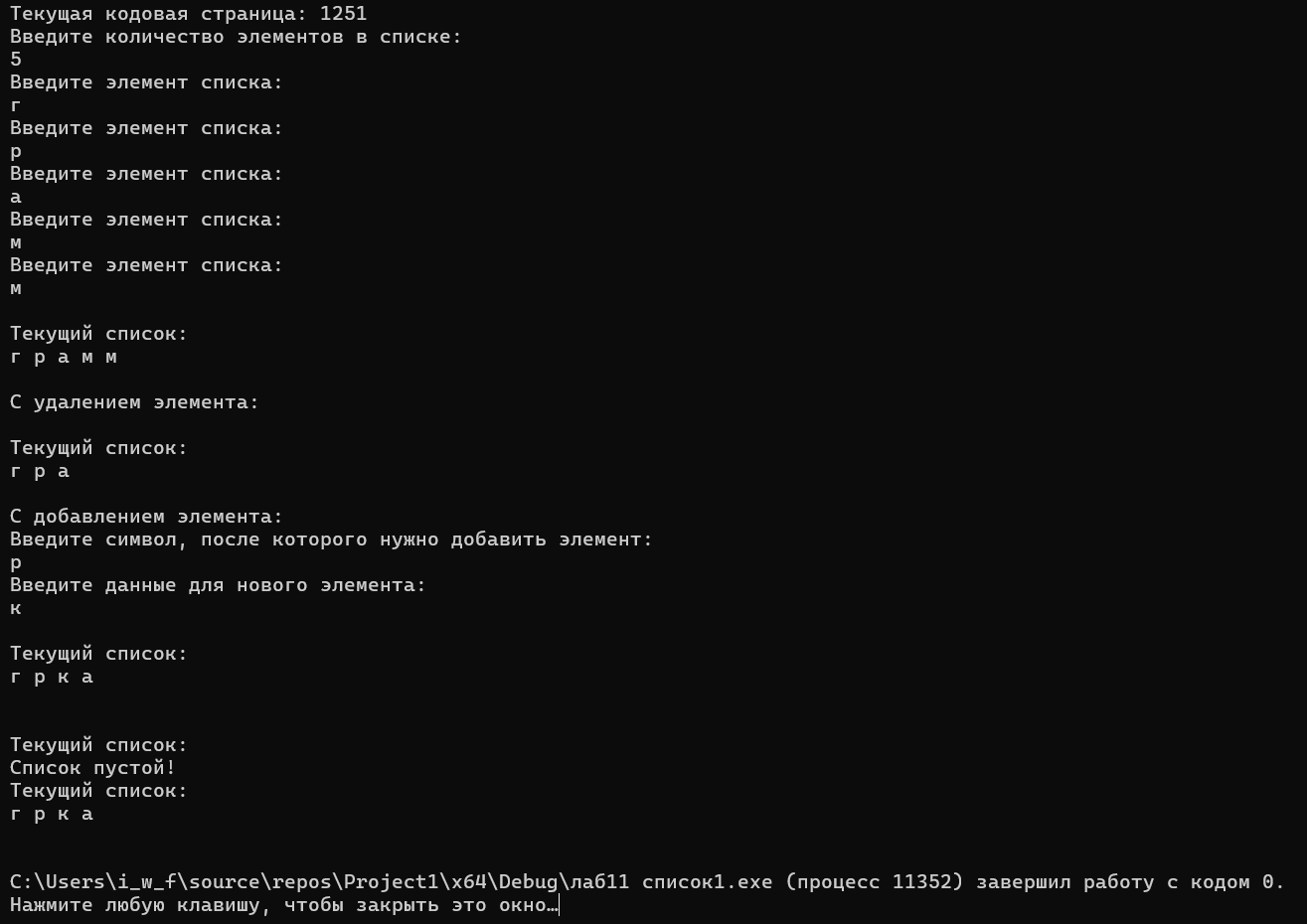
cin.ignore();

vost(list, ff); // восстановление списка из файла

printt(list); // вывод списка

return 0;

}

1. Результат работы программы:

**Решение двунаправленным списком**

1. Анализ задачи:
2. Создать структуру, состоящую из головы и хвоста, которые изначально равны нулевым данным.
3. Создать функцию, которая будет добавлять и связывать новые элементы в цепочке, добавляя элемент в конец (изменяя хвост).
4. Создать функцию, которая при вызове будет выводить массив.
5. Создать функцию для удаления элементов с одинаковыми ключевыми полями. Создаются две переменные, которые будут проходиться и сравнивать все элементы массива и искать совпадение ключевых полей. После нахождения совпадений начинается несколько проверок (является ли какой-либо из элементов головой/хвостом или же стоят ли они рядом с друг другом), попадая в нужное условие совпадающие элементы удаляются, а разрыв между элементами устранится.
6. Создать функцию, которая находит элемент с заданным ключом, и связывает его с новым элементов (включает элемент в массив).
7. Создать функцию добавления массива в файл.
8. Создать функцию удаления массива из динамической памяти.
9. Создать функцию для восстановления массива из файла.
10. Код программы:

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

struct node {

char data;

node\* next = nullptr;

node\* past = nullptr;

};

struct List {

node\* head = nullptr;

node\* teil = nullptr;

};

void pushback(List& list, const int& data) { //добавляю новый элемент в конец списка

node\* new\_node = new node; //создаю новый динамический узел

new\_node->data = data; //присваиваю полю узла данные

if (list.head == nullptr) { //если список пустой

list.head = new\_node; //новый узел - головной узел списка

list.teil = new\_node; //новый узел - хвостовой узел списка

}

else { // если список не пуcтой

list.teil->next = new\_node; //связываю новый узел с хвостовым

node\* n = list.teil;

list.teil = new\_node; //меняю хвостовой узел на новый

list.teil->past = n;

}

}

void printt(List& list) { //вывод текущего списка

cout << "Текущий список:" << endl;

node\* current\_node = list.head;

if (current\_node != nullptr) { //если список не пустой

while (current\_node != nullptr) { //пока не дойду до последнего элемента

cout << current\_node->data << ' '; //вывод данных текущего узла

current\_node = current\_node->next; //переход к следующему узлу

}

}

else {

cout << "Список пустой!";

}

cout << endl << endl;

delete current\_node; //очищаю память

}

void delel(List& list) {

node\* cur = list.head;

node\* cur2 = list.teil;

bool f = true; // флаг для выхода из циклов

while (cur->next != nullptr) { // прохождение по всем элементам

for (; cur2 != cur; cur2 = cur2->past) {

if (cur->data == cur2->data) { // если найдены совпадающие элементы

if (cur->next != cur2) { // если совпадающие элементы не находятся рядом

if (cur == list.head && cur2 != list.teil) { // если совпадающий элемент- головной

node\* old\_h = list.head;

list.head = list.head->next; // замена головного элемента

list.head->past = nullptr; // удаление мусора из старых данных

delete old\_h;

node\* nn = cur2->past;

node\* nnn = cur2->next;

nn->next = nnn; // связь узлов для исключения второго совпадающего элемента

nnn->past = nn; // связь узлов для исключения второго совпадающего элемента

delete cur2;

f = false; // смена флага для выхода из цикла

break;

}

if (cur != list.head && cur2 == list.teil) { // если совпадающий элемент- хвостовой

list.teil = list.teil->past;

list.teil->next = nullptr;

node\* nn = cur->past;

node\* k = cur->next;

nn->next = k; // связь узлов

k->past = nn; // связь узлов

delete cur;

f = false; // смена флага для выхода из цикла

break;

}

if (cur == list.head && cur2 == list.teil) { // если совпадающий элемент- головной, а второй- хвостовой

node\* old\_h = list.head;

list.head = list.head->next; // смена головного элемента

list.head->past = nullptr; // удаление старого головного элемента

delete old\_h;

node\* old\_t = list.teil;

list.teil = list.teil->past; // смена хвостового элемента

list.teil->next = nullptr; // удаление старого хвостового элемента

delete old\_t;

f = false;

break;

}

if (cur != list.head && cur2 != list.teil) { // если оба элемента находятся в середине

node\* nn = cur2->past;

nn->next = cur2->next; // связь узлов

node\* c = nn->past;

while (c->data == cur->data) c = c->past;

nn->past = c; // связь узлов

delete cur2;

node\* n = cur->past;

n->next = cur->next; // связь узлов

delete cur;

f = false; // смена флага для выхода из цикла

break;

}

}

else {

if (cur == list.head) { // если совпадающий элемент- головной

list.head = list.head->next; // перемещение головного элемента

list.head = list.head->next; // перемещение головного элемента

delete cur;

delete cur2;

f = false; // смена флага для выхода из цикла

break;

}

if (cur2 == list.teil) { // если совпадающий элемент- хвостовой

list.teil = list.teil->past; // перемещение хвостового элемента

list.teil = list.teil->past; // перемещение хвостового элемента

list.teil->next = nullptr;

delete cur;

delete cur2;

f = false; // смена флага для выхода из цикла

break;

}

if (cur != list.head && cur2 != list.teil) { // если элементы находятся в середине

node\* n = cur->past;

n->next = cur2->next; // связь узлов

delete cur;

delete cur2;

}

}

f = false; // смена флага для выхода из цикла

break;

}

}

if (f == false) break; // прерывание цикла, если флаг-ложь

cur = cur->next; // переход к следующему элементу

cur2 = list.teil;

}

}

void add(List& list, const int& data, char k) {

node\* cur = list.head;

while (cur != nullptr) {

if (cur->data == k) { // если найден заданный элемент

node\* nw = new node; // создание нового элемента

nw->data = data; // присваивание значения элементу

nw->past = cur; // связь узлов

nw->next = cur->next; // связь узлов

cur->next = nw; // связь узлов

break;

}

cur = cur->next; // переход к следующему элементу

}

}

void del(List& list) {

node\* ptr\_node = list.head;

if (ptr\_node != nullptr) {

while (ptr\_node != nullptr) {

node\* new\_Head = list.head->next;

delete list.head; //удаляю текущий головной элемент

list.head = new\_Head; //присваиваю головному элементу новый элемент

ptr\_node = list.head;

}

}

}

void ffile(List& list, ofstream& file) { //запись данных в файл

if (list.head != nullptr) {//если список не пустой

node\* pointer\_q = list.head; //указатель на первый элемент

while (pointer\_q != nullptr) { //пока не дойду до конца

file << pointer\_q->data << endl;

pointer\_q = pointer\_q->next; //перехожу на следующий узел

}

}

}

void vosst(List& list, ifstream& file) { //восстановление

string all\_str;

getline(file, all\_str); //считываю строку

pushback(list, all\_str[0]);

int counter = 0;

while (getline(file, all\_str)) { //пока не пройду весь файл

pushback(list, all\_str[0]); //добавляю

counter++;

}

}

int main() {

setlocale(LC\_ALL, "ru");

system("chcp 1251");

ifstream f("neww.txt");

ofstream ff("neww.txt");

int n;

cout << "Введите количество элементов в списке: " << endl;

cin >> n;

char symbol, k, t;

List list; //инициализирую список

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

char a;

cout << "Введите элемент списка: " << endl;

cin >> a;

pushback(list, a);

}

printt(list);

delel(list);

cout << "После удаления :" << endl;

printt(list);

cout << "Введите символ, после которого нужно добавить элемент: " << endl;

cin >> k;

cout << "Введите новые данные: " << endl;

cin >> t;

add(list, t, k);

cout << "После добавления :" << endl;

printt(list);

ffile(list, ff);

del(list);

cin.ignore();

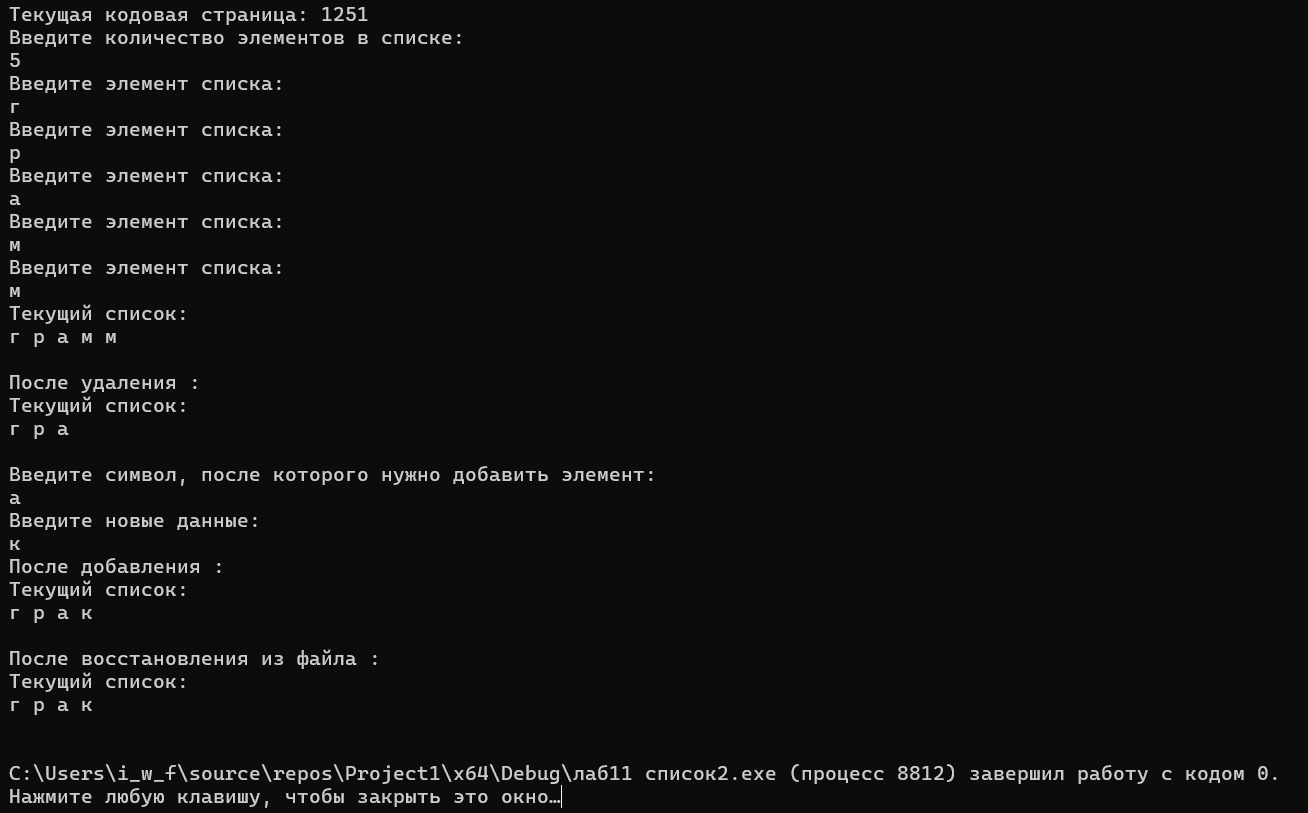
vosst(list, f);

cout << "После восстановления из файла :" << endl;

printt(list);

return 0;

}

1. Результат работы программы:

**Решение с помощью стека**

1. Анализ задачи:
2. Создать структуру, состоящую из верхнего элемента с ссылкой на предыдущий элемент.
3. Создать функцию для создания стека, которая будет добавлять элементы в массив, но в обратном порядке.
4. Создание функции для добавления нового элемента.
5. Создать функцию, которая при вызове будет выводить массив.
6. Создать функцию, которая будет доставать верхний элемент стека, сохранять его, а также менять верхний элемент массива на предыдущий.
7. Создать функцию, которая, используя функции вынимания верхнего элемента массива, при помощи функции добавления элемента в массив, будет добавлять его во второй переданный массив.
8. Создать функцию для удаления элементов с одинаковыми ключевыми полями. Создаются две переменные, которые будут проходиться и сравнивать все элементы массива и искать совпадение ключевых полей. После нахождения совпадений во второй стек перекладываются все элементы, кроме совпадающих.
9. В функции main найти в стеке элемент с заданным ключевым полем с помощью цикла, а затем, используя второй массив, передать то количество элементов в функцию для перемещения элементов во второй массив, которое идет до элемента с заданным ключевым полем (его включая). Далее добавить новый заданный элемент во второй массив, а затем туда же оставшиеся элементы из первого массива.
10. Создать функцию добавления массива в файл.
11. Создать функцию удаления массива из динамической памяти.
12. Создать функцию для восстановления массива из файла.
13. Код программы:

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

struct stack { //стек

char data; //данные

stack\* past; //адрес на предыдущий элемент

};

stack\* sozd(int n) {

if (n != 0) {

char symbol;

stack\* verh, \* cur = new stack;

verh = NULL;

cout << "Введите элемент ";

cin >> symbol;

cur->data = symbol; //присваиваю знаяение новому элементу

cur->past = NULL; //адрес на предыдущий элемент

verh = cur; //изменяю верхний элемент в стеке

for (int i = 1; i < n; i++) { //добавление новых элемнтов

stack\* new\_element = new stack;

cout << "Введите элемент ";

cin >> symbol;

new\_element->data = symbol;

new\_element->past = verh;

verh = new\_element;

}

return verh;

}

return NULL;

}

stack\* push(stack\*& verh, char symbol) { //добавление элемента

stack\* pointer = new stack; //выделяю память для нового стека

pointer->data = symbol; //присваиваю данные новому элементу

pointer->past = verh; //адрес предыдущего элемента

verh = pointer; //новый верхний элемент

return verh; //возвращаю новый верхний элемент

}

void printt(stack\* verh) { //вывод стека

cout << endl << "Текущий стек:" << endl;

if (verh != NULL) { //если стек не пустой

stack\* pointer\_el = verh; //указатель на первый элемент

while (pointer\_el != NULL) { //пока не дойду до нижнего элемента

cout << pointer\_el->data << ' ';

pointer\_el = pointer\_el->past;

}

delete pointer\_el; //освобождаю память

}

else { //если стек пуст

cout << "Стек пуст!";

}

cout << endl << endl;

}

char pop\_el(stack\*& verh) { //нахожу верхний элемент

stack\* pointer\_el = verh;

stack\* t = pointer\_el->past;

char data\_ptr = pointer\_el->data; //данные необходимого элемента

verh = t; //меняю верхний элемент

delete pointer\_el;

return data\_ptr;

}

void stst(stack\*& stackk, stack\*& stackk2, int n) { //перенос элементов из стека в стек

for (int i = 0; i < n; i++) { //переношу элементы из стека 1 в стек 2

char t = pop\_el(stackk); //извлекаю верхний элемент из первого

push(stackk2, t); //переношу элементы во второй стек

}

}

void del\_el(stack\* verh, stack\*& stackk, stack\*& stackk2, int n) {

stack\* cur = verh;

stack\* cur2 = new stack;

cur2 = verh->past;

bool fl = true; // флаг для выхода из цикла

while (cur->past != nullptr) {

while (cur2 != nullptr) {

if (cur->data == cur2->data) {

char g = cur->data;

for (int i = 0; i < n; i++) { //переношу элементы из стека 1 в стек 2

char t = pop\_el(stackk); //извлекаю верхний элемент из первого

if (t != g) push(stackk2, t); //переношу элементы во второй стек

}

fl = false; // меняем флаг для выхода из цикла

break; // прерывание цикла

}

cur2 = cur2->past; // переходим к следующему элементу для сравнения

}

if (fl == false) break;

cur = cur->past; // переходим к следующему элементу

cur2 = cur->past; // переходим к следующему элементу

}

}

void ffile(stack\* verh, ofstream& file) { //запись данных в файл

if (verh != NULL) { //если стек не пустой

stack\* pointer\_element = verh; //указатель на первый элемент

while (pointer\_element != NULL) { //пока не дойду до нижнего элемента

file << pointer\_element->data << endl;

pointer\_element = pointer\_element->past;

}

delete pointer\_element; //освобождаю память

}

}

void del(stack\*& verh) { //удаление всех элементов стека

while (verh->past != NULL) {

stack\* pointer\_element = verh->past;

delete verh; //освобождение памяти верхнего элемента

verh = pointer\_element;

}

verh = NULL;

}

void vosst(stack\*& verh, ifstream& file, int n) { //восстановление стека

stack\* the\_new\_stack, \* pointer\_element = new stack;

the\_new\_stack = NULL;

string all\_str;

getline(file, all\_str); //считываю строку

pointer\_element->data = all\_str[0]; //присваиваю знаяение новому элементу

pointer\_element->past = NULL; //адрес на предыдущий элемент

the\_new\_stack = pointer\_element; //изменяю верхний элемент в стеке

while (getline(file, all\_str)) { //пока не пройду весь файл

push(the\_new\_stack, all\_str[0]); //переношу элементы во временный стек

}

stst(the\_new\_stack, verh, n); //переношу элементы в главный стек

}

int main() {

setlocale(LC\_ALL, "ru");

system("chcp 1251");

ifstream f("neww.txt");

ofstream ff("neww.txt");

int n, m = 1;

cout << "Введите количество элементов ";

cin >> n; //количество элементов в списке

cout << endl;

stack\* stackk = sozd(n);

cout << "Вывод стека: " << endl;

printt(stackk); //вывожу текущий стек

stack\* stackk2 = sozd(0);

del\_el(stackk, stackk, stackk2, n);

cout << "Вывод стека после удаления элементов: " << endl;

for (int i = 0; i < n - 2; i++) { //переношу элементы из стека 1 в стек 2

char t = pop\_el(stackk2); //извлекаю верхний элемент из первого

push(stackk, t); //переношу элементы во второй стек

}

printt(stackk);

char a, b;

cout << "введите заданный ключ: " << endl;

cin >> a;

stack\* r = stackk;

for (int i = 0; i < n - 2; i++) {

if (r->data == a) break;

r = r->past;

m++;

}

if (m == n - 3) {

stst(stackk, stackk2, n - 2);

cout << "введите новый элементы стека:" << endl;

cin >> b;

push(stackk2, b); //добавляю символ в стек 1

}

else if (m == 0) {

stst(stackk, stackk2, 1);

cout << "введите новый элементы стека:" << endl;

cin >> b;

push(stackk2, b); //добавляю символ в стек 1

stst(stackk, stackk2, n - 3);

}

else {

stst(stackk, stackk2, n - 2 - m -1);

cout << "введите новый элементы стека:" << endl;

cin >> b;

push(stackk2, b); //добавляю символ в стек 1

stst(stackk, stackk2, n - 2 - m );

}

stst(stackk2, stackk, n - 1);

printt(stackk);

ffile(stackk, ff);

del(stackk);

cin.ignore();

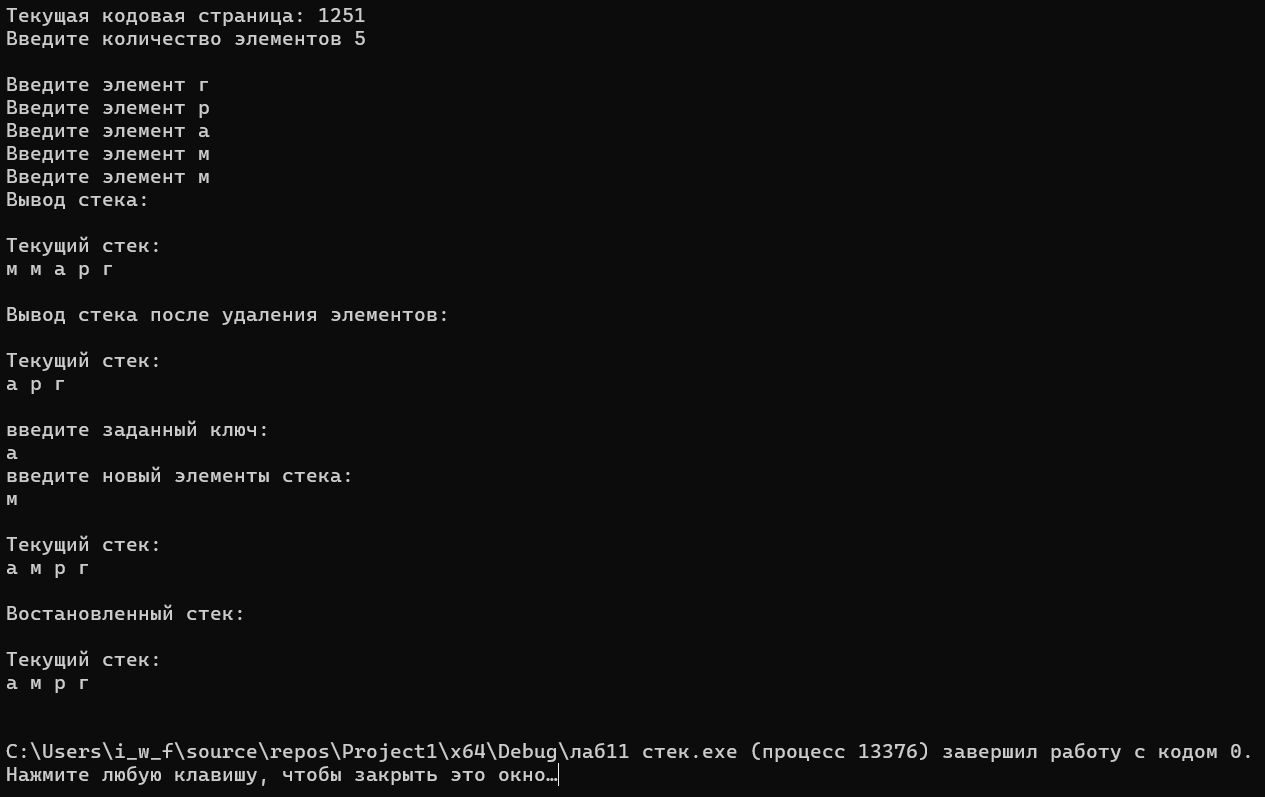
vosst(stackk, f, n - 1);

cout << "Востановленный стек:" << endl;

printt(stackk);

return 0;

}

1. Результат работы программы:

**Решение с помощью очереди**

1. Анализ задачи:
2. Создать структуру, состоящую из верхнего элемента с ссылкой на следующий элемент.
3. Создать функцию для инициализации очереди, которая присваивает элементу очереди головной и хвостовой элемент.
4. Создать функцию для создания целой очереди.
5. Создание функции для добавления нового элемента.
6. Создать функцию, которая при вызове будет выводить массив.
7. Создать функцию, которая будет удалять головной элемент очереди.
8. Создать функцию для удаления элементов с одинаковыми ключевыми полями. Создаются две переменные, которые будут проходиться и сравнивать все элементы массива и искать совпадение ключевых полей. После нахождения элементы будут по очереди заново перекладываться в конец очереди (копироваться) за исключением совпадающих элементов.
9. Создать функцию для добавления нового элемента. Элементы будут по очереди заново перекладываться в конец очереди (копироваться) до того, пока цикл не дойдет до элемента с заданным ключевым полем. После добавления элемента с заданным ключевым полем, добавится также и новый элемент, а затем копирование продолжится с того момента, на котором остановилось.
10. Создать функцию добавления массива в файл.
11. Создать функцию удаления массива из динамической памяти.
12. Создать функцию для восстановления массива из файла.
13. Код программы:

#include<iostream>

#include<fstream>

#include<string>

using namespace std;

struct node {

char data; //данные

node\* next; //указатель на следующий элемент

};

struct och { //очередь

int size; //размер

node\* head\_node = nullptr;

node\* teil\_node = nullptr;

};

void init(och& line, const char& symbol) { //инициализирую очередь

node\* new\_node = new node; //резервирую память под новый элемент

new\_node->data = symbol; //присваиваю данные

line.head\_node = new\_node; //головной элемент

line.teil\_node = new\_node; //хвостовой элемент

line.size = 1; //размер очереди, т.к. есть только один элемент

}

void push(och& line, const char& symbol) { //добавляю элемент в конец очереди

node\* new\_node = new node; //резервирую память под новый элемент

line.size++; //увеличиваю текущий размер очереди на один

new\_node->data = symbol; //присваиваю данные

new\_node->next = nullptr; //последний элемент не указывает на ч-л

line.teil\_node->next = new\_node; //прошлый последний элемент указывает на новый последний

line.teil\_node = new\_node; //новый хвостовой элемент

}

void sozd(och& line, int n) { //формирую очередь

char symbol;

cout << "Введите элемент ";

cin >> symbol;

init(line, symbol); //инициализирую первым элементом

for (int i = 1; i < n; i++) {

cout << "Введите элемент ";

cin >> symbol; //ввод элемента

push(line, symbol); //ставлю новый элемент в конец очереди

}

}

void printt(och& line) { //вывод очереди

cout << endl << "Текущая очередь: " << endl;

if (line.head\_node != nullptr) {

node\* point = line.head\_node; //указатель на первый элемент

cout << "start -> ";

while (point != nullptr) { //пока не дойду до конца

cout << point->data << ' '; //вывожу значение текущего элемента

point = point->next; //перехожу на следующий узел

}

cout << "-> end" << endl << endl;

}

else {

cout << "Очередь пуста!" << endl << endl;

}

}

void pop(och& line) { //удаляю головной элемент из очереди

node\* point = line.head\_node; //указатель на первый элемент

line.head\_node = line.head\_node->next; //голова - следующий элемент

--line.size; //уменьшаю длину очереди

delete point; //освобождаю память

}

void del\_el(och& line) {

int cnt = 0; // счетчик, чтобы пройтись по всей очереди

bool fl = true; // флаг для выхода из цикла

node\* cur = line.head\_node;

node\* cur2 = cur->next;

int h = line.size;

while (cnt < line.size-1) {

while (cur2 != nullptr) {

if (cur->data == cur2->data) { // если найдены одинаковые элементы

char g = cur->data;

node\* curr = line.head\_node;

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

if (curr->data != g) {

push(line, curr->data); // добавляем элемент в конец очереди

curr = curr->next; // переходим к следующему элементу

pop(line); // удаляем дублирующийся элемент

}

else {

curr = curr->next; // пропускаем дублирование элемента

pop(line); // удаляем элемет

}

}

fl = false; // меняем флаг для выхода из цикла

break;

}

cur2 = cur2->next; // меняем элемент для сравнения

}

if (fl == false) break; // выход из цикла, если флаг- ложь

cur = cur->next; // переходим к следующему элементу

cur2 = cur->next;

cnt++;

}

}

void add(och& line, char& k, char& t) {

int cnt = 0; // счетчик, чтобы пройтись по всей очереди

node\* cur = line.head\_node;

int h = line.size;

while (cnt < h) {

if (cur->data == k) { // если нужный элемент найден

push(line, cur->data); // добавляем элемент в конец очереди

cur = cur->next; // переходим к следующему элементу

pop(line); // удаляем дублирующийся элемент

push(line, t); // добавление нового элемента

}

else {

push(line, cur->data); // добавляем элемент в конец очереди

cur = cur->next; // переходим к следующему элементу

pop(line); // удаляем дублирующийся элемент

}

cnt++;

}

}

void ffile(och& line, ofstream& file) { //запись данных в файл

if (line.head\_node != nullptr) {

node\* point = line.head\_node; //указатель на первый элемент

while (point != nullptr) { //пока не дойду до конца

file << point->data << endl;

point = point->next; //перехожу на следующий узел

}

}

}

void del(och& line) {

while (line.head\_node->next != nullptr) { //пока не дойду до последнего элемента

pop(line); //удаляю головной элемент

}

node\* point = line.head\_node; //указатель на первый элемент

line.head\_node = nullptr;

--line.size; //уменьшаю размер

delete point; //освобождаю память от последнего элемента

}

void vosst(och& line, ifstream& file) { //восстановление

string all\_str;

getline(file, all\_str); //считываю строку

init(line, all\_str[0]); //добавляю в очередь

while (getline(file, all\_str)) { //пока не пройду фесь файл

push(line, all\_str[0]); //добавляю в очередь

}

}

int main() {

setlocale(LC\_ALL, "ru");

system("chcp 1251");

ifstream f("neww.txt");

ofstream ff("neww.txt");

och line; //создаю очередь

int n;

char symbol\_key;

cout << "Введите количество элементов ";

cin >> n; //количество элементов в списке

sozd(line, n); //формирую новую очередь

printt(line);//вывожу текущую очередь

del\_el(line);

int p = line.size;

cout << "Вывод стека после удаления элементов: " << endl;

printt(line);//вывожу текущую очередь

char k, t;

cout << "введите заданный ключ: " << endl;

cin >> k;

cout << "введите новый элемент:" << endl;

cin >> t;

add(line, k, t);

cout << "После добавления :" << endl;

printt(line);//вывожу текущую очередь

ffile(line, ff);

del(line);

cin.ignore();

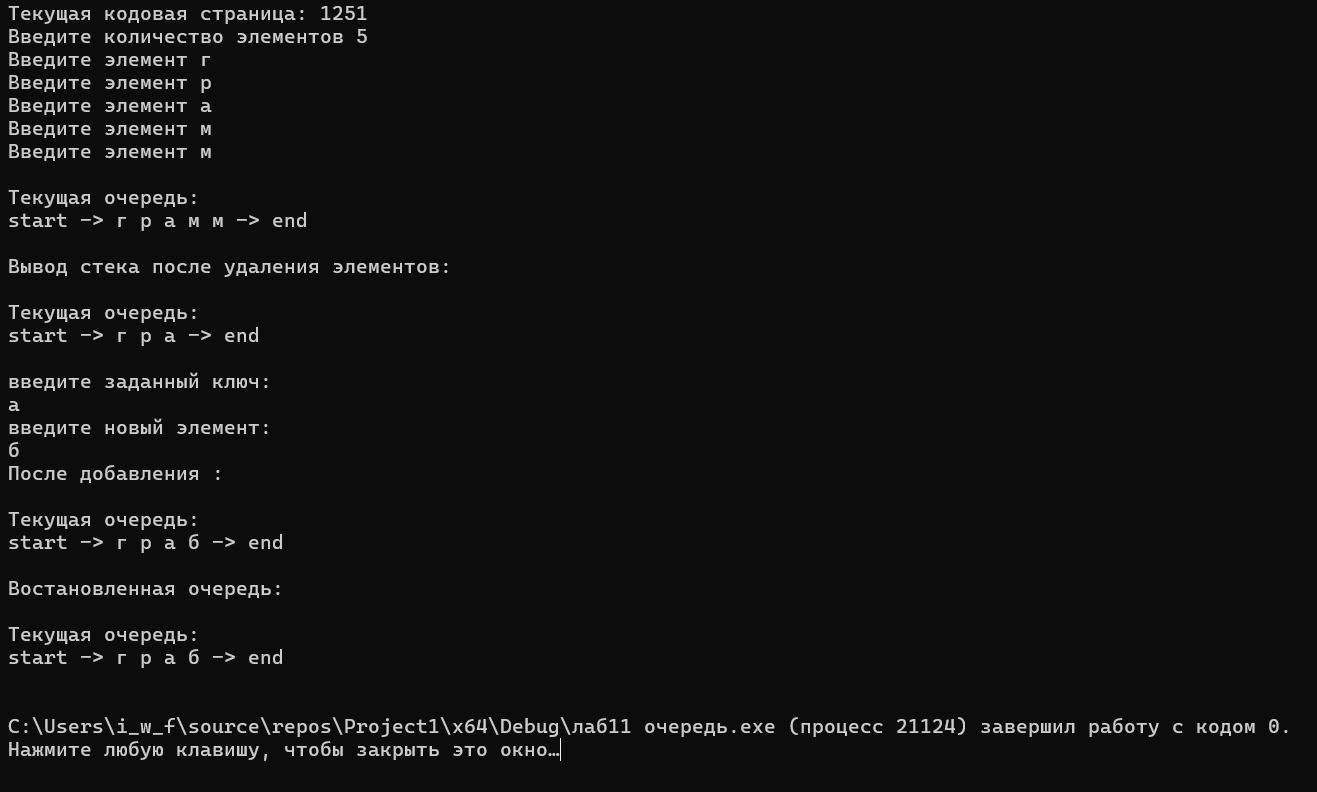
vosst(line, f);

cout << "Востановленная очередь:" << endl;

printt(line);

return 0;

}

1. Результат работы программы: