ГУАП

КАФЕДРА № 14

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | A. Ю. Петров |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №2 |
|  |
| по курсу: |
| ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 1242 |  |  |  | А. А. Костюкевич |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

**1. Постановка задачи**

Меню пользователя должно содержать возможность редактирования любой записи любого объекта, удаление любого объекта, добавления нового объекта на любую позицию.

1. Стандартные потоки

Определить класс с именем TRAIN, содержащий следующие поля:

● название пункта назначения;

● номер поезда;

● время отправления.

Определить методы доступа к этим полям и перегруженные операции извлечения и вставки для объектов типа TRAIN. Заранее число объектов не известно.

Написать программу, выполняющую следующие действия:

● записи должны быть упорядочены по номерам поездов;

● вывод на экран информации о поезде, номер которого введен с

клавиатуры;

● если таких поездов нет, выдать на дисплей соответствующее

сообщение.

2. Файловые и строковые потоки

С использованием файловых и строковых потоков написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

**2. Формализация задачи**

Для решения 1 задачи сначала необходимо создать класс Train. В файле train.h будет содержаться определение, а в train.cpp – реализация. Класс Train будет иметь три private поля: пункт назначения, номер поезда, время отправления. Также класс Train будет иметь конструктор без параметров, с параметрами, копирования и деструктор, set и get методы для каждого поля и функции отображения всех поездов и изменения поезда в области public.

Дальше нужно создать класс Container, который будет хранить объекты Train. Этот класс будет иметь структуру связного списка, в файле Container.h будет определение, а в файле Container.cpp – реализация. Container будет содержать private поля: указатель на голову, на хвост и количество элементов. Элементом класса будет структура Node, каждый узел которой имеет указатель на следующий элемент и указатель на объект Train. В области public Container будет содержать конструктор без параметров, с параметрами, копирования и деструктор, методы добавления поезда, удаления, изменения, отображения всех поездов, сортировки поездов по номеру, поиска поезда по номеру и перегруженный оператор обращения по индексу. Данная программа работает со стандартными потоками iostream

Для решения 2 задачи нужно создать класс SentenseFilter. В файле sentence\_filter.h будет определение, а в файле sentence\_filter.cpp – реализация. Этот класс в области private будет иметь три поля: текст, количество слов, флаг подключения файла, а также два метода: разделение текста на слова, подсчет количества слов в предложении. В области public будут находиться конструктор без параметров, с параметрами, копирования, деструктор и метод отображения результата обработки. Данная программа работает со строковыми потоками sstream и файловыми fstream.

Файл check.h содержит описание функции проверки на корректность вводимых пользователей данных, файл .cpp содержит реализацию этой функции. Файл main.cpp – основной файл, содержащий пользовательский интерфейс взаимодействия с программами 1 и 2, а также обработку исключений. Пользовательский интерфейс позволяют пользователю иметь максимальные возможности при работе с программой, все параметры, которые могут быть введены с пользователем, вводятся с клавиатуры. Данная программа работает с динамическим выделением памяти, без использования STL контейнеров.

**3. Таблица описания классов**

Таблица 1. Описание классов

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Назначение | Наследование | Тип |
| Train | Базовый класс для представления объектов Поезд | Агрегация с Container | class |
| Container | Класс-контейнер для хранения объектов Train | Агрегация с Train | class |
| Node | Вспомогательная структура для хранения данных в классе Container | Композиция с Container | struct |
| SentenceFilter | Класс для работы с фильтрацией текста | - | class |

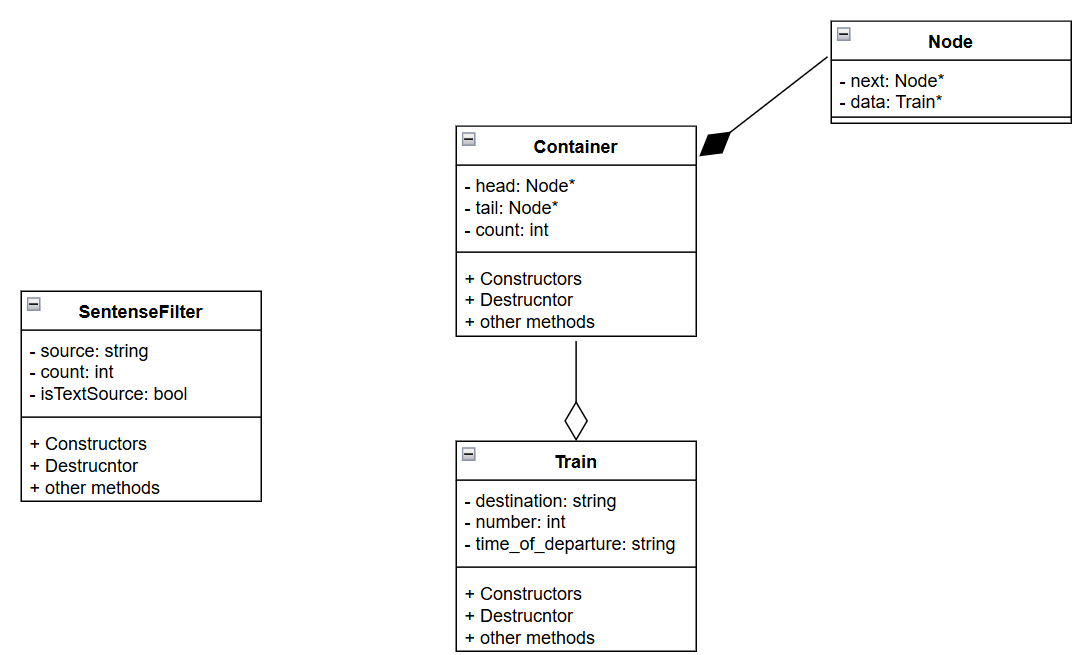


Рисунок 1 – иерархия классов

**4. Исходный код**

<https://github.com/qwatrix69/lab2>

check.h:

#pragma once

#include <iostream>

#include <limits>

#include <string>

using namespace std;

int check\_input();

check.cpp:

#include "check.h"

int check\_input() {

int value;

while (true) {

cin >> value;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Некорректный ввод! Пожалуйста, введите число: " << endl;

} else {

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return value;

}

}

}

container.h:

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include "train.h"

using namespace std;

struct Node

{

Node\* next;

Train\* data;

};

class Container

{

private:

Node\* head;

Node\* tail;

int count;

public:

Container();

Container(Node\* h, Node\* t, const int c);

Container(const Container& other);

~Container();

Node\* get\_head();

Node\* get\_tail();

void add\_train(Train\* Tr, int index);

Container& delete\_train(int index);

Container& edit\_train(int index);

void display\_trains();

void sort\_trains\_by\_number();

void search\_train(const int number);

Container& operator[](int index);

};

container.cpp:

#include "container.h"

Container::Container() : head(nullptr), tail(nullptr), count(0) {cout << "Вызван конструктор без параметров для класса Container\n";}

Container::Container(Node\* h, Node\* t, const int c) : head(h), tail(t), count(c) {

cout << "Вызван конструктор с параметрами для класса Container\n";

}

Container::Container(const Container& other) : head(other.head), tail(other.tail), count(other.count) {

cout << "Вызван конструктор копирования для класса Container\n";

}

Container::~Container() {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp->data;

delete temp;

}

tail = nullptr;

count = 0;

cout << "Вызван деструктор для класса Container" << endl;

}

Node\* Container::get\_head() {

return this->head;

}

Node\* Container::get\_tail() {

return this->tail;

}

void Container::add\_train(Train\* Tr, int index) {

if (index < 0 || index > count) {

throw out\_of\_range("Index out of range");

}

Node\* node\_to\_add = new Node;

node\_to\_add->data = Tr;

node\_to\_add->next = nullptr;

if (index == 0) {

node\_to\_add->next = head;

head = node\_to\_add;

if (count == 0) {

tail = node\_to\_add;

}

} else {

Node\* prev = head;

for (int i = 0; i < index - 1; ++i) {

prev = prev->next;

}

node\_to\_add->next = prev->next;

prev->next = node\_to\_add;

if (node\_to\_add->next == nullptr) {

tail = node\_to\_add;

}

}

++count;

}

Container& Container::delete\_train(int index) {

if (index < 0 || index >= count) {

throw out\_of\_range("Index out of range");

}

Node\* temp = head;

if (index == 0) {

head = head->next;

delete temp->data;

delete temp;

if (head == nullptr) {

tail = nullptr;

}

} else {

Node\* prev = nullptr;

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

prev = temp;

temp = temp->next;

}

prev->next = temp->next;

if (temp == tail) {

tail = prev;

}

delete temp->data;

delete temp;

}

--count;

return \*this;

}

void Container::display\_trains() {

Node\* temp = head;

if (count == 0) {

cout << "Поездов нет" << endl;

return;

}

int index = 1;

while (temp != 0) {

cout << index << " - Информация о поезде:\n";

temp->data->display\_train();

temp = temp->next;

index++;

}

}

void Container::sort\_trains\_by\_number() {

if (count < 2) return;

for (Node\* i = head; i != nullptr; i = i->next) {

for (Node\* j = head; j->next != nullptr; j = j->next) {

if (j->data->get\_number() > j->next->data->get\_number()) {

Train\* temp = j->data;

j->data = j->next->data;

j->next->data = temp;

}

}

}

}

void Container::search\_train(const int number) {

Node\* temp = head;

while (temp != nullptr) {

if (temp->data->get\_number() == number) {

temp->data->display\_train();

return;

}

temp = temp->next;

}

cout << "Поезд с номером " << number << " не найден." << endl;

}

Container& Container::edit\_train(int index) {

if (index < 0 || index >= count) {

throw out\_of\_range("Index out of range");

}

Node\* temp = head;

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

temp = temp->next;

}

temp->data->edit\_train();

return \*this;

}

Container& Container::operator[](int index) {

if (index < 0 || index >= count) {

throw out\_of\_range("Index out of range");

}

Node\* temp = head;

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

temp = temp->next;

}

return \*this;

}

train.h:

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Train {

private:

string destination;

int number;

string time\_of\_departure;

public:

Train();

Train(const string& d, const int& n, const string& t);

Train(const Train& other);

~Train();

string get\_destination();

void set\_destination(const string& d);

int get\_number();

void set\_number(const int& n);

string get\_time();

void set\_time(const string& t);

void display\_train();

void edit\_train();

};

train.cpp:

#include "train.h"

Train::Train() : destination(""), number(0), time\_of\_departure("") {cout << "Вызван конструктор без параметров для класса Train\n";}

Train::Train(const string& d, const int& n, const string& t) : destination(d), number(n), time\_of\_departure(t) {

cout << "Вызван конструктор с параметрами для класса Train\n";

}

Train::Train(const Train& other) : destination(other.destination), number(other.number), time\_of\_departure(other.time\_of\_departure) {

cout << "Вызван конструктор копирования для класса Train\n";

}

Train::~Train() {cout << "Вызван деструктор для класса Train\n";}

string Train::get\_destination() {

return this->destination;

}

void Train::set\_destination(const string& d) {

this->destination = d;

}

int Train::get\_number() {

return this->number;

}

void Train::set\_number(const int& n) {

this->number = n;

}

string Train::get\_time() {

return this->time\_of\_departure;

}

void Train::set\_time(const string& t) {

this->time\_of\_departure = t;

}

void Train::display\_train() {

cout << "Пункт назначения: " << this->get\_destination() << endl;

cout << "Номер поезда: " << this->get\_number() << endl;

cout << "Время отправления: " << this->get\_time() << endl;

cout << endl;

}

void Train::edit\_train() {

int choice;

cout << "Выберите, что хотите изменить:\n";

cout << "1. Пункт назначения\n";

cout << "2. Номер поезда\n";

cout << "3. Время отправления\n";

cout << "Введите ваш выбор: ";

cin >> choice;

switch (choice) {

case 1: {

string new\_destination;

cout << "Введите новый пункт назначения: ";

cin.ignore();

getline(cin, new\_destination);

set\_destination(new\_destination);

break;

}

case 2: {

int new\_number;

cout << "Введите новый номер поезда: ";

cin >> new\_number;

set\_number(new\_number);

break;

}

case 3: {

string new\_time;

cout << "Введите новое время отправления: ";

cin.ignore();

getline(cin, new\_time);

set\_time(new\_time);

break;

}

default:

cout << "Неверный выбор!" << endl;

}

}

sentence\_filter.h:

#pragma once

#include <iostream>

#include <fstream>

#include <sstream>

#include <cstring>

#include <cctype>

#include <string>

using namespace std;

class SentenceFilter {

private:

string source;

int count;

bool isTextSource;

void split\_into\_sent(const string& text, string\*& sentences, int& sentenceCount) const;

int count\_words(const string& sentence) const;

public:

SentenceFilter();

SentenceFilter(const string& filename, int wordCount);

SentenceFilter(const string& text, int wordCount, bool isText);

SentenceFilter(const SentenceFilter& other);

~SentenceFilter();

void result() const;

};

sentense\_filter.cpp:

#include "sentense\_filter.h"

SentenceFilter::SentenceFilter() : source(""), count(0), isTextSource(false) {

cout << "Вызван конструктор без параметров для класса SentenceFilter\n";

}

SentenceFilter::SentenceFilter(const string& filename, int count)

: source(filename), count(count), isTextSource(false) {

cout << "Вызван конструктор с параметрами для класса SentenceFilter\n";

}

SentenceFilter::SentenceFilter(const string& text, int count, bool isText)

: source(text), count(count), isTextSource(isText) {

cout << "Вызван конструктор с параметрами для класса SentenceFilter\n";

}

SentenceFilter::SentenceFilter(const SentenceFilter& other)

: source(other.source), count(other.count), isTextSource(other.isTextSource) {

cout << "Вызван конструктор копирования для класса SentenceFilter\n";

}

SentenceFilter::~SentenceFilter() {

cout << "Вызван деструктор для класса SentenceFilter\n";

}

void SentenceFilter::result() const {

string text;

if (isTextSource) {

text = source;

} else {

ifstream file(source);

if (!file.is\_open()) {

cerr << "Не удалось открыть файл: " << source << endl;

return;

}

ostringstream buffer;

buffer << file.rdbuf();

text = buffer.str();

file.close();

}

cout << "Считанный текст:\n" << text << "\n\n";

string\* sentences = nullptr;

int sentenceCount = 0;

split\_into\_sent(text, sentences, sentenceCount);

cout << "Всего найдено предложений: " << sentenceCount << endl;

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

int words = count\_words(sentences[i]);

if (words == count) {

cout << sentences[i] << endl;

}

}

delete[] sentences;

}

void SentenceFilter::split\_into\_sent(const string& text, string\*& sentences, int& sentenceCount) const {

const int maxSentences = 100;

sentences = new string[maxSentences];

sentenceCount = 0;

ostringstream sentenceStream;

for (char ch : text) {

sentenceStream << ch;

if (ch == '.' || ch == '!' || ch == '?') {

if (sentenceCount < maxSentences) {

sentences[sentenceCount++] = sentenceStream.str();

sentenceStream.str("");

sentenceStream.clear();

}

}

}

if (!sentenceStream.str().empty() && sentenceCount < maxSentences) {

sentences[sentenceCount++] = sentenceStream.str();

}

}

int SentenceFilter::count\_words(const string& sentence) const {

int count = 0;

istringstream ss(sentence);

string word;

while (ss >> word) {

++count;

}

return count;

}

main.cpp:

#include "container.h"

#include "train.h"

#include "check.h"

#include "sentense\_filter.h"

void display\_menu() {

cout << "\n===== Меню управления поездами =====" << endl;

cout << "1. Добавить поезд" << endl;

cout << "2. Удалить поезд" << endl;

cout << "3. Редактировать поезд" << endl;

cout << "4. Показать все поезда" << endl;

cout << "5. Сортировать поезда по номеру" << endl;

cout << "6. Найти поезд по номеру" << endl;

cout << "0. Выйти" << endl;

cout << "Введите ваш выбор: ";

}

int trains\_program() {

Container trains;

int choice;

while (true) {

display\_menu();

choice = check\_input();

switch (choice) {

case 1: {

string destination, time;

int number, index;

cout << "Введите пункт назначения: ";

cin.ignore();

getline(cin, destination);

cout << "Введите номер поезда: ";

number = check\_input();

cout << "Введите время отправления: ";

getline(cin, time);

cout << "Введите индекс, куда вставить поезд: ";

index = check\_input();

Train\* new\_train = new Train(destination, number, time);

try {

trains.add\_train(new\_train, index - 1);

cout << "Поезд добавлен." << endl;

} catch (const out\_of\_range& e) {

cout << e.what() << endl;

delete new\_train;

}

break;

}

case 2: {

int index;

cout << "Введите индекс удаляемого поезда: ";

index = check\_input();

try {

trains.delete\_train(index - 1);

cout << "Поезд удален." << endl;

} catch (const out\_of\_range& e) {

cout << e.what() << endl;

}

break;

}

case 3: {

int index;

cout << "Введите индекс редактируемого поезда: ";

index = check\_input();

try {

trains.edit\_train(index - 1);

cout << "Поезд отредактирован." << endl;

} catch (const out\_of\_range& e) {

cout << e.what() << endl;

}

break;

}

case 4: {

trains.display\_trains();

break;

}

case 5: {

trains.sort\_trains\_by\_number();

cout << "Поезда отсортированы по номеру." << endl;

trains.display\_trains();

break;

}

case 6: {

cout << "Введите номер поезда для поиска: ";

int number = check\_input();

trains.search\_train(number);

break;

}

case 0: {

cout << "Выход из программы." << endl;

return 0;

}

default: {

cout << "Неверный выбор! Попробуйте снова." << endl;

break;

}

}

}

return 0;

}

int words\_program() {

try {

int wordCount;

cout << "Введите количество слов в предложении: ";

if (!(cin >> wordCount)) {

throw invalid\_argument("Ошибка: некорректный ввод количества слов.");

}

int choice;

cout << "Выберите источник (1 - строка, 2 - файл): ";

if (!(cin >> choice)) {

throw invalid\_argument("Ошибка: некорректный ввод выбора источника.");

}

if (choice == 1) {

cin.ignore();

char text[8192];

cout << "Введите текст: ";

cin.getline(text, sizeof(text));

if (cin.fail()) {

throw overflow\_error("Ошибка: превышена максимальная длина текста.");

}

SentenceFilter filter(text, wordCount, true);

filter.result();

} else if (choice == 2) {

char filename[256];

cout << "Введите имя файла: ";

cin >> filename;

ifstream file(filename);

if (!file) {

throw runtime\_error("Ошибка: файл не найден или не может быть открыт.");

}

file.close();

SentenceFilter filter(filename, wordCount);

filter.result();

} else {

throw out\_of\_range("Ошибка: выбран несуществующий источник.");

}

}

catch (const invalid\_argument& e) {

cerr << e.what() << endl;

return 1;

}

catch (const overflow\_error& e) {

cerr << e.what() << endl;

return 2;

}

catch (const runtime\_error& e) {

cerr << e.what() << endl;

return 3;

}

catch (const out\_of\_range& e) {

cerr << e.what() << endl;

return 4;

}

catch (...) {

cerr << "Произошла неизвестная ошибка." << endl;

return -1;

}

return 0;

}

int main() {

int choice;

while (true) {

cout << "Выберите тип задания:" << endl;

cout << "1 - Стандартные потоки" << endl;

cout << "2 - Файловые и строковые потоки" << endl;

cout << "3 - Выход" << endl;

choice = check\_input();

switch (choice) {

case 1:

trains\_program();

break;

case 2:

words\_program();

break;

case 3:

cout << "Завершение работы." << endl;

return 0;

default:

cout << "Неверный выбор! Попробуйте снова." << endl;

break;

}

}

}

**5. Результаты работы программы**

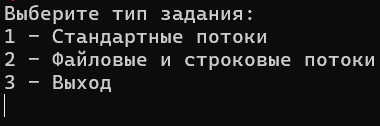


Рисунок 2 – пользовательское меню

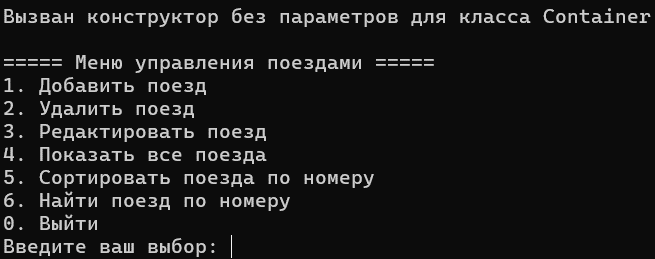


Рисунок 3 – меню 1 программы

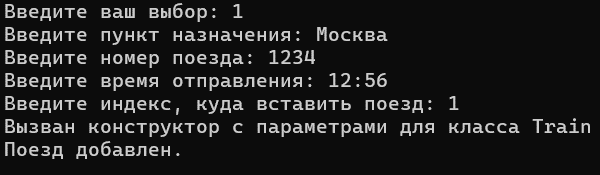


Рисунок 4 – добавление поезда

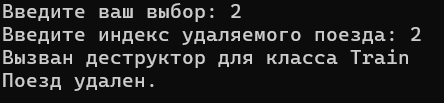


Рисунок 5 – удаление поезда

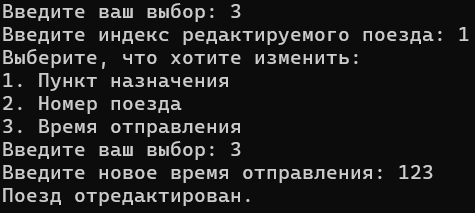


Рисунок 6 – редактирование поезда

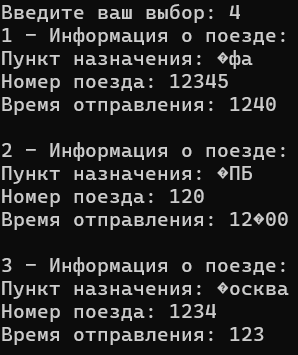


Рисунок 7 – отображение списка поездов

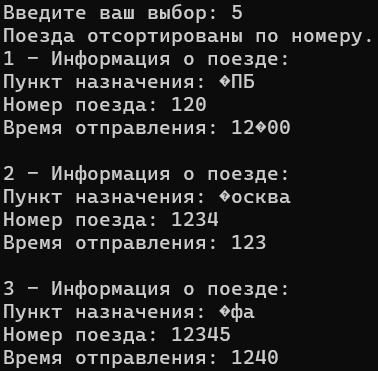


Рисунок 8 – сортировка поездов по номеру

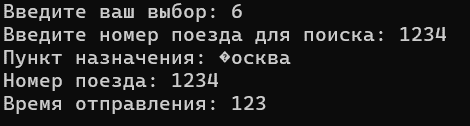


Рисунок 9 – поиск поезда по номеру

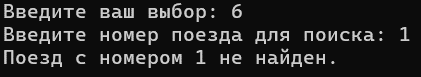


Рисунок 10 – попытка поиска поезда с несуществующим номером

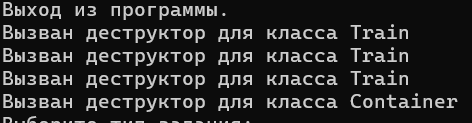


Рисунок 11 – завершение программы 1

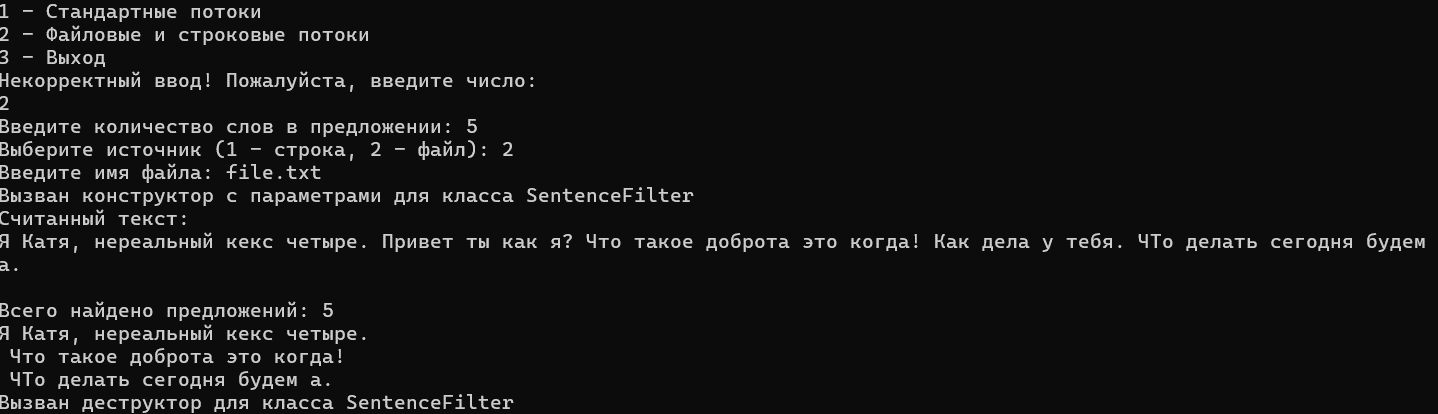


Рисунок 12 – работа программы 2 с файлом

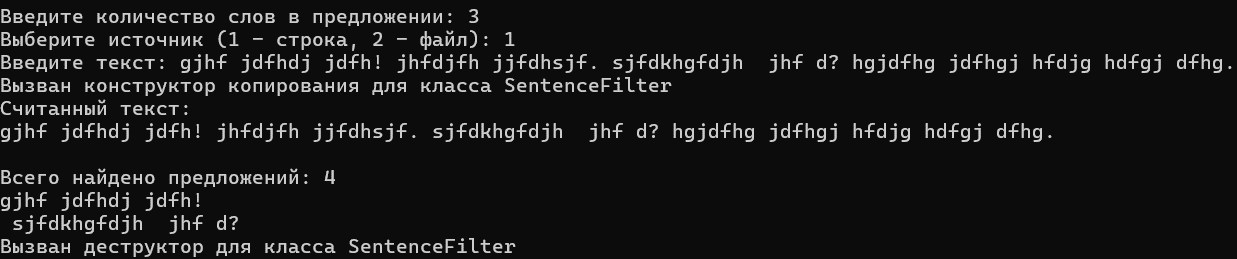


Рисунок 13 – работа программы 2 со строкой

**6. Выводы**

В ходе выполнения данной лабораторной работы, я выполнил поставленную задачу. Я повторил работу с классами, структурами, наследованием, перегрузкой операторов, работой со стандартными, строковыми и файловыми потоками. Также я изучил способы составления UML-диаграмм для графического отображения иерархии классов.