МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе № 1
по дисциплине
«Тестирование программного обеспечения»

Инспекция кода

Обучающийся: _____ Шихалев А.О.

Преподаватель: _____ Курочкин М.А.

«_____»_____20____г.

Санкт-Петербург, 2025

Содержание

Введение

В разработке программного обеспечения, помимо основной задачи — реализовать заявленную в спецификации функциональность, существует не менее важная задача — обеспечить качество разработанного решения.

Тестирование программного обеспечения — это проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов.

На практике ни один метод тестирования не может выявить все ошибки в программе. Это связано с тем, что ресурсы проекта (деньги, время, персонал), в том числе и на тестирование, ограничены. Однако правильно проведенное тестирование позволяет обнаружить большинство ошибок, что позволяет их оперативно исправить и тем самым повысить качество программного обеспечения.

В данной лабораторной работе используется ручное тестирование — процесс проверки программного обеспечения, выполняемый специалистами без использования каких-либо специальных автоматизированных средств.

Ручное тестирование применяется не вместо компьютерного тестирования, а вместе с ним, что позволяет выявить ошибки в программе на более ранних стадиях.

1 Постановка задачи

Требуется: провести инспекцию кода, выступая в роли разработчика программы.
Для выполнения данной задачи необходимо:

- Изучить методы ручного тестирования;
- Провести инспекцию кода, проанализировать ее результаты;
- Исправить программу в соответствии с рекомендациями специалиста по тестированию.

2 Описание методов инспекции кода

Существуют три основных метода ручного тестирования:

- инспекция кода;
- сквозной просмотр;
- тестирование удобства использования.

Эти методы могут применяться на любой стадии разработки ПО, как к отдельным готовым модулям или блокам, так и к приложению в целом.
Инспекция и сквозной просмотр включают в себя чтение или визуальную проверку исходного кода программы группой лиц. Оба метода предполагают выполнение определенной подготовительной работы. Завершающим этапом является обмен мнениями между участниками проверки на специальном заседании. Цель такого заседания — нахождение ошибок, но не их устранение (т.е. тестирование, а не отладка).

2.1 Инспекция кода

Инспекция кода — это набор процедур и методик обнаружения ошибок путем анализа (чтения) кода группой специалистов.

2.1.1 Группа инспектирования кода

Обычно в состав группы входят четыре человека, один из которых играет роль *координатора*. Координатор должен быть квалифицированным программистом, но не автором тестируемой программы, детальное знание которой от него не требуется. Вторым участником группы является программист, а остальными — проектировщик программы (если это не сам программист) и специалист по тестированию.
В рамках выполнения лабораторной работы в инспекции кода участвовало всего три человека: программист, специалист по тестированию и координатор.

2.1.2 Человеческий фактор

Если программист воспринимает инспектирование своей программы как деятельность, направленную против него лично, и занимает оборонительную позицию, то процесс инспектирования не будет эффективным. Программист должен оставить самолюбие в стороне и рассматривать инспекцию только в позитивном и конструктивном ключе, не забывая о том, что целью инспекции является нахождение ошибок и, следовательно, улучшение качества программы.

2.2 Сквозной просмотр

Тестирование программы методом сквозного просмотра также как и в инспекции кода включает в себя проверку программного кода группой лиц.

При сквозном просмотре код проверяется группой разработчиков (оптимально — 3-4 человека), лишь один из которых является автором программы. Таким образом, большую часть программы тестирует не ее создатель, а другие члены команды разработчиков, что согласуется со вторым принципом, согласно которому тестирование программистом собственной программы редко бывает эффективным.
Преимуществом сквозных просмотров, снижающим стоимость отладки (исправления ошибок), является возможность точной *локализации ошибки*. Кроме того, в процессе сквозного просмотра обычно удается выявить целую группу ошибок, которые впоследствии можно устранить все вместе. При тестировании программы на компьютере обычно проявляются лишь признаки ошибок (например, программа не может корректно завершиться или выводит бессмысленные результаты), а сами они обнаруживаются и устраняются по отдельности.

2.3 Проверка за столом

Метод ручного тестирования «проверка за столом» может рассматриваться как инспекция или сквозной просмотр кода, выполняемые *одним человеком*, который вычитывает код программы, проверяет его, руководствуясь контрольным списком ошибок, и (или) прогоняет через логику программы тестовые данные.
Для большинства людей проверка за столом является относительно непродуктивной. Это объясняется прежде всего тем, что такая проверка представляет собой полностью неупорядоченный процесс. Вторая, более важная причина заключается в том, что проверка за столом вступает в противоречие со *вторым принципом тестирования*, согласно которому тестирование программистом собственных программ обычно оказывается неэффективным. Следовательно, оптимальный вариант состоит в том, чтобы такую проверку выполнял человек, не являющийся автором программы (например, два программиста могут обмениваться программами для взаимной проверки, а не проверять собственные программы), но даже в этом случае такая проверка *менее эффективна*, чем сквозные просмотры или инспекции. В основном именно по этой причине лучше, чтобы сквозные просмотры или инспекции осуществлялись в группе.

2.4 Рецензирование

Рецензирование — это процедура анонимной оценки общих характеристик качества, обслуживаемости, расширяемости, удобства использования и ясности программного обеспечения. *Цель* данного метода — предоставить программисту возможность получить стороннюю оценку результатов своего труда.
Общее руководство процессом осуществляет *администратор*, выбираемый из числа программистов. В свою очередь, администратор отбирает в группу рецензентов от 6 до 20 участников (6 — это необходимый минимум, обеспечивающий анонимность оценок). Пред-

полагается, что все участники специализируются в одной области. Каждый из участников предоставляет для рецензирования две своих программы, одну из которых он считает наилучшей, а вторую — наихудшей по качеству.

Когда будут собраны все программы, их распределяют случайным образом между участниками. Каждому участнику дают для рецензирования четыре программы. Две из них относятся к категории «наилучших», а две — к категории «наихудших» программ, но рецензенту не сообщают, какой именно является каждая из них. Любой участник тратит на просмотр одной программы 30 минут и заполняет ее оценочную анкету. После просмотра всех четырех программ рецензент оценивает их относительное качество.

Рецензента также просят предоставить свои замечания к программе и дать рекомендации по ее улучшению.

После просмотра всех программ каждому участнику передают анкеты с оценками двух его программ. Кроме того, участники получают *статистическую сводку*, отражающую общие и детализированные данные о рейтинге их собственных программ среди всего набора, а также анализ того, насколько оценки, данные участником чужим программам, близки к оценкам тех же программ со стороны других рецензентов.

3 Описание тестируемой программы

3.1 Формальное описание

Название программы: Словарь на основе хэш-таблицы.

Дано: Слово, введенное пользователем и номер команды для выполнения операций.

Требуется: Реализовать хэш-таблицу, поддерживающую следующие операции:

1. Добавление слова в словарь.
2. Удаление слова из словаря.
3. Вывод на экран всего словаря.
4. Поиск слова в словаре.
5. Вставка слова из файла.
6. Очистка всего словаря.

Ограничения:

1. Номер команды в меню, введенный пользователем должен лежать в диапазоне от 0 до 6 включительно.
2. Слово, которое может быть добавлено в словарь, должно быть обязательно состоять из символов кириллицы.

Спецификация:

№	Входные данные		Результат работы программы	Комментарий
	Команда	Слово (прочерк – отсутствие значения)		
1	1	-	Словарь пуст.	Поскольку мы только запустили программу и слов не добавляли – следовательно, словарь пуст.
2	2	-	Слова из текстового файла были успешно добавлены в хэш-таблицу.	После добавления всех слов из текстового файла выводится весь словарь на экран.

№	Входные данные		Результат работы программы	Комментарий
	Команда	**Слово (прочерк – отсутствие значения)**		
3	add	–	Вывод ошибки о некорректном вводе на экран.	На консоль выведется сообщение о некорректном вводе команды меню.
4	3	“hello”	Вывод ошибки на экран.	Слово не добавится в словарь, так как слово не состоит из символов кириллицы.
5	3	“привет”	Слово успешно добавлено в словарь.	Поскольку слово состоит только из кириллицы, оно успешно добавляется в словарь.
6	3	“привет мир”	Вывод ошибки на экран.	Слово не добавится в словарь, поскольку содержит пробелы.
7	3	“”	Добавление прошло успешно	В процессе инспекции кода выявилось отсутствие проверки введенного пользователем слова на пустую строку.
8	4	“привет”	Вывод индекса бакета слова.	Поскольку слово присутствует в словаре, программа выводит соответствующее сообщение и индекс бакета, в котором это слово находится.
9	5	“привет”	Вывод об успешном удалении слова.	Слово успешно удаляется из словаря, если оно присутствует в нем.
10	6	-	Словарь очищается.	Все слова полностью удаляются из словаря.
11	0	-	Выход из программы.	Программа завершает свою работу.

3.2 Исходный код программы

HashTable.h

```
#pragma once
#include <vector>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cctype>

#include "LinkedList.h"

using namespace std;

class HashTable {

    private:
    LinkedList* table;

    int total_buckets;
    int total_elements = 0;
    double fillability;

    public:

    HashTable(int size);
    void displayHashTable();
    void insert(const string& key);
    void remove(const string& key);
    int search(const string& key);
    unsigned int hashFunction(const string& key);
    void insertFromFile(const string& filename);
    void rehash();
    void clear();
    int getCount();

    ~HashTable();

};

```

HashTable.cpp

```cpp
#include "HashTable.h"
#include <locale>

bool isAlphaNum(char c) {

    if (std::isalnum(static_cast<unsigned char>(c))) {
        return true;
    }

    return (c >= 'A' && c <= 'Я') || (c >= 'а' && c <= 'я');
}


HashTable::HashTable(int size) {
    total_buckets = size;
    table = new LinkedList[total_buckets];
}

void HashTable::insert(const string& key) {

    int index = hashFunction(key);

    if (search(key) == -1) {

        table[index].insert(key);
        total_elements++;
        fillability = static_cast<double>(total_elements) / static_cast<
    ↪double>(total_buckets);
    }

    if (fillability >= 0.9) {
        rehash();
    }
}

void HashTable::remove(const string& key) {

    int index = hashFunction(key);
    if (table[index].remove(key)) {
        total_elements--;
        fillability = static_cast<double>(total_elements) / static_cast<
    ↪double>(total_buckets);
    }
```

```
42
43      else cout << "\n Элемент не найден! " << endl;
44  }
45
46  int HashTable::search(const string& key) {
47
48      int index = hashFunction(key);
49      if (table[index].search(key)) return index;
50      else return -1;
51
52  }
53
54
55  void HashTable::displayHashTable() {
56      cout << endl;
57
58      if (total_elements == 0) {
59          cout << "Хэш-таблица пуста!" << endl;
60      }
61      else {
62          for (int i = 0; i < total_buckets; i++) {
63              cout << "[" << i << "]: ";
64              table[i].display();
65              cout << endl;
66          }
67      }
68  }
69
70
71  HashTable::~HashTable() {
72      delete[] table;
73  }
74
75
76  unsigned int HashTable::hashFunction(const string& key) {
77
78      unsigned int hash_value = 0;
79      int a = 33; // основание полинома
80
81      for (char c : key) {
82          hash_value = (hash_value * a + static_cast<unsigned int>(c)) %
    ↪total_buckets;
83      }
84
```

```
85      return hash_value;
86  }
87
88
89  void HashTable::insertFromFile(const string& filename) {
90
91      ifstream file(filename);
92
93      if (!file.is_open()) {
94          cout << "Ошибка! Не удалось открыть файл!" << filename << endl;
95          return;
96      }
97
98      string line, word;
99      while (getline(file, line)) {
100
101         stringstream ss(line);
102
103         while (ss >> word) {
104
105             transform(word.begin(), word.end(), word.begin(), ::tolower);
106             word.erase(remove_if(word.begin(), word.end(), [](char c) {
107                 return !isAlphaNum(c);
108             }), word.end());
109
110             if (!word.empty()) {
111                 insert(word);
112                 // cout << "Добавили " << word << endl;
113             }
114         }
115
116     }
117
118     cout << "Слова успешно добавлены! " << endl << endl;
119
120     // cout << "Количество элементов: " << total_elements << endl;
121     // cout << "Коэф. заполняемости хэш-таблицы: " << fillability << endl;
122 }
123
124
125 void HashTable::rehash() {
126
127     // cout << "Пересоздаем таблицу..." << endl;
128
```

```
129     int old_buckets = total_buckets;
130 
131     total_buckets *= 2;
132 
133     LinkedList* old_table = table;
134     table = new LinkedList[total_buckets];
135     total_elements = 0;
136 
137     for (int i = 0; i < old_buckets; i++) {
138 
139         ListNode* temp = old_table[i].getHead();
140 
141         while (temp != nullptr) {
142             insert(temp->data);
143             temp = temp->next;
144         }
145 
146     }
147 
148     delete[] old_table;
149 
150 }
151 
152 void HashTable::clear() {
153 
154     for (int i = 0; i < total_buckets; i++) {
155         table[i].clear();
156     }
157 
158     total_elements = 0;
159     fillability = 0.0;
160 
161     cout << "Хэш-таблица полностью очищена!" << endl;
162 
163 }
164 
165 int HashTable::getCount() {
166     return total_elements;
167 }
```

main.cpp

```
1 #include <iostream>
```

```cpp
#include <algorithm>
#include <cctype>
#include <locale>
#include <windows.h>

#include "HashTable.h"
#include "B-plus-tree.h"
#include "check.h"


using namespace std;

int main() {

    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Russian");


    HashTable myHashTable(10);
    Bplus myTree(2);

    while (true) {

        cout << endl << " Хэш-таблица\n";

        cout << "\n  1. Вывести словарь на экран \n";
        cout << "  2. Дополнить словарь из текстового файла \n";
        cout << "  3. Добавить новое слово в словарь \n";
        cout << "  4. Поиск слова в словаре \n";
        cout << "  5. Удалить слово из словаря \n";
        cout << "  6. Очистить словарь \n";

        cout << "  0. Выход из программы\n";

        cout << "\n Выберите действие: ";

        int b = checkingInput(6);
        string word;
        bool flag;
        int index;

        switch (b) {

```

```
46              case 1:
47
48              myHashTable.displayHashTable();
49              cout << endl << "Количество слов в словаре: " << myHashTable.
    ↪getCount() << endl;
50              break;
51
52              case 2:
53
54              myHashTable.insertFromFile("story.txt");
55              break;
56
57              case 3:
58              cout << "Введите слово (без пробелов), которое хотите добавить:
    ↪ " << endl;
59              clear();
60              word = checkingString();
61              transform(word.begin(), word.end(), word.begin(), ::tolower);
62
63              index = myHashTable.search(word);
64              if (index != -1) {
65                  cout << "Слово уже есть в словаре!" << endl;
66              }
67              else {
68                  myHashTable.insert(word);
69                  cout << "Слово успешно добавлено! " << endl;
70
71              }
72
73              break;
74
75              case 4:
76
77              cout << "Введите слово, которое хотите найти: " << endl;
78              clear();
79              word = checkingString();
80              transform(word.begin(), word.end(), word.begin(), ::tolower);
81
82              index = myHashTable.search(word);
83              if (index != -1) {
84                  cout << "Слово " << word << " присутствует в словаре! " <<
    ↪endl;
85                  cout << "Индекс бакета: " << index << endl << endl;
86              }
```

```
87              else cout << "Слово " << word << " отсутствует в словаре! " <<
    ↪endl << endl;
88
89              break;
90
91              case 5:
92
93              cout << "Введите слово, которое хотите удалить: " << endl;
94              clear();
95              word = checkingString();
96              transform(word.begin(), word.end(), word.begin(), ::tolower);
97
98              myHashTable.remove(word);
99              break;
100
101             case 6:
102             myHashTable.clear();
103             break;
104
105             case 0:
106             return 0;
107
108             default:
109             cout << "Ошибка!";
110         }
111     }
112     return 0;
113 }
```

4 Описание проведенной инспекции кода

В заседании участвовали следующие люди:

- программист, разработчик программы: Шихалев Алексей;
- специалист по тестированию: Кондраев Дмитрий;
- секретарь: Тищенко Артем.

В начале заседания программистом была описана логика программы и представлена ее спецификация. В процессе инспекции были заданы вопросы (32 шт.) с целью выявления несоответствий спецификации. Программист, разработчик программы, ответил на вопросы и получил обратную связь от специалиста по тестированию. Секретарем был составлен протокол заседания с записанными вопросами и ответами на них, а также рекомендации по исправлению несоответствий. Далее представлены заданные вопросы и ответы на них.

4.1 Протокол заседания

1. Программа реализует словарь на основе хэш-таблицы?

   Ответ: Да.

2. На каком языке написана программа?

   Ответ: C++.

3. Какие типы переменных используются?

   Ответ: HashTable, LinkedList, bool, int, double, char, string.

4. Все ли переменные объявлены?

   Ответ: Да.

5. Используются ли переменные с неинициализированными значениями?

   Ответ: Нет, не используются, но есть одна переменная fillability, поле класса HashTable, которая не инициализируется в конструкторе, а только при добавлении слова.

   Замечание: Следует ее инициализовать в конструкторе.

6. Выходят ли индексы за границы массива в хэш-таблице?

   Ответ: Нет.

7. Используются ли нецелочисленные индексы?

   Ответ: Нет, не используются.

8. Есть ли “висячие” ссылки?

   Ответ: Нет.

9. Корректно ли используются атрибуты памяти, адресуемой с помощью ссылок и указателей?

   Ответ: Да.

10. Корректно ли названы все методы и переменные?

    Ответ: Да, кроме переменной b, отвечающей за номер команды меню, введенной пользователем.

    Замечание: Следует более осмысленно назвать переменные.

11. Корректно ли используются операторы сравнения?

    Ответ: Нет, не совсем.

    Замечание: Согласно условию Йоды, при использовании оператора сравнения “==”, слева от оператора пишется константный член выражения. Такой стиль призван предотвратить свойственную данным языкам ошибку — использование операции присваивания “=” вместо сравнения “==”.

12. Корректно ли используются булевы выражения?

    Ответ: Да.

13. Есть ли в программе “магические” числа?

    Ответ: Да.

    Замечание: Следует вынести их в константы.

14. Правильно ли инициализированы массивы и строки?

    Ответ: Да.

15. Соблюдены ли правила наследования объектов?

    Ответ: Наследование не используется.

16. Может ли какая-нибудь переменная типа int переполняться?

    Ответ: Теоретически да, если добавить слишком много слов, переменная total_elements может переполниться.

    Замечание: Стоит установить ограничение на максимальное количество добавляемых слов.

17. Присутствуют ли в программе потенциально бесконечные циклы?

    Ответ: Нет.

18. Проверяется ли введенное слово на пустую строку?

    Ответ: Нет, не проверяется.

    Замечание: Стоит добавить проверку на пустую строку.

19. Осуществляется ли проверка корректности входных данных?

    Ответ: Да.

20. Используются ли файлы для ввода-вывода?

    Ответ: Да, для считывания и добавления слов в словарь из предоставленного текста в файле формата .txt

21. Правильно ли заданы атрибуты файлов?

    Ответ: Да.

22. Корректно ли обрабатываются признаки конца файла?

    Ответ: Да.

23. Открываются ли файлы перед обращением к ним?

    Ответ: Да.

24. Закрываются ли файлы после обращения к ним?

    Ответ: Да.

25. Обрабатываются ли ошибки ввода вывода?

    Ответ: Да. В случае, если файл не найден, выведется ошибка.

26. Удаляются ли неиспользуемые переменные из памяти?

    Ответ: Да, в классах HashTable и LinkedList есть деструкторы.

27. Нет ли пропущенных функций?

    Ответ: Нет. Все функции реализованы.

28. Выдаются ли предупреждения при компиляции?

    Ответ: Нет.

29. Выдаются ли ошибки при компиляции?

    Ответ: Тоже нет.

30. Выполняются ли вычисления с присваиванием несовпадающих типов?

    Ответ: Нет.

31. Есть ли комментарии в программе?

    Ответ: Да, один-два есть.

    Замечание: Стоит добавить разъяснительные комментарии.

32. Оформлен ли код в соответствии с некоторым регламентом (стандартом оформления)?

    Ответ: Да, код оформлен в соответствии с регламентом Google C++ Style Guide.

4.2 Итоги проведения заседания

На первом этапе инспекции кода при обсуждении логики программы, разработчиком программы, был получен опыт обнаружения несоответствий спецификации на этапе чтения программного кода специалистом по тестированию. Были получены рекомендации в написании программного кода от специалиста по тестированию, такие как осмысленное название переменных, рекомендация по грамотному использованию операторов сравнения, установка ограничения на добавления слов в словарь, добавление проверки на пустую строку и null, добавление разъяснительных комментариев.

5 Исправление кода программы

5.1 Рекомендация: добавить инциализацию переменной fillability в конструктор класса HashTable

Исходный вариант кода:

```
HashTable::HashTable(int size) {
    total_buckets = size;
    table = new LinkedList[
    ↪total_buckets];
}
```

Исправленный вариант кода:

```
HashTable::HashTable(int size) {
    total_buckets = size;
    fillability = 0.0;
    table = new LinkedList[
    ↪total_buckets];
}
```

5.2 Рекомендация: переименовать переменную b, отвечающую за номер пункта меню

Исходный вариант кода:

```
int b = checkingInput(12);
```

Исправленный вариант кода:

```
int menuChoice = checkingInput(12);
```

5.3 Рекомендация: убрать все “магические” числа в программе, вынести их в константы

Исходный вариант кода:

```
int menuChoice = checkingInput(12);
```

Исправленный вариант кода:

```
const int MAX_MENU_ITEMS = 12;
int menuChoice = checkingInput(
    ↪MAX_MENU_ITEMS);
```

Исходный вариант кода:

```
void HashTable::insert(const string&
    ↪ key) {
    int index = hashFunction(key);
    if (search(key) == -1) {
        table[index].insert(key);
        total_elements++;
        fillability = static_cast<
    ↪double>(total_elements) /
    ↪static_cast<double>(
    ↪total_buckets);
    }
    if (fillability >= 0.9) {
    rehash(); }
```

Исправленный вариант кода:

```
static const int NOT_FOUND = -1;
static const double REHASH_THRESHOLD
    ↪;

void HashTable::insert(const string&
    ↪ key) {
    int index = hashFunction(key);
    if (search(key) == NOT_FOUND) {

        table[index].insert(key);
        total_elements++;
        fillability = static_cast<
    ↪double>(total_elements) /
```

```
        / static_cast<double>(
    ↪total_buckets); }
    if (fillability >=
    ↪REHASH_THRESHOLD) {
        rehash(); }
}
```

5.4 Рекомендация: в местах, где используется оператор сравнения, слева от него записывать константы, чтобы предотвратить ошибку присваивания “=”

Исходный вариант кода:

```
if (search(key) == NOT_FOUND) {...}
```

Исправленный вариант кода:

```
if (NOT_FOUND == search(key)) {...}
```

5.5 Рекомендация: установить ограничение на максимальное количество добавляемых слов

Исходный вариант кода:

```
void HashTable::insert(const string&
    ↪ key) {
    int index = hashFunction(key);
    if (NOT_FOUND == search(key)) {
        table[index].insert(key);
        total_elements++;
        fillability = static_cast<
    ↪double>(total_elements) /
    ↪static_cast<double>(
    ↪total_buckets);
    }
    if (fillability >=
    ↪REHASH_THRESHOLD) {
        rehash();
    }
}
```

Исправленный вариант кода:

```
void HashTable::insert(const string&
    ↪ key) {
    if (total_elements >= MAX_WORDS)
    ↪ {
        cout << "Словарь переполнен
    ↪!\n" << endl;
        return;
    }
    int index = hashFunction(key);
    if (NOT_FOUND == search(key)) {
    ↪
        table[index].insert(key);
        total_elements++;
        fillability = static_cast<
    ↪double>(total_elements) /
    ↪static_cast<double>(
    ↪total_buckets);
    }
    if (fillability >=
    ↪REHASH_THRESHOLD) {
        rehash(); } }
```

5.6 Рекомендация: добавить проверку введенного пользователем слова на пустую строку

Исходный вариант кода:

```
string checkingString() {
    string str;
    while (true) {
        getline(cin, str);
        bool valid = true;
        for (char c : str) {
            if (isspace(c) || !((c >= 'A' && c <= 'Я') || (c >= 'a' && c <= 'я'))) {
                valid = false;
                break;
            }
        }
        if (valid) {
            return str;
        }
        else {
            cout << "Ошибка! Введите слово, содержащее только русские буквы и без пробелов!" << endl;
        }
    }
}
```

Исправленный вариант кода:

```
string checkingString() {
    string str;
    while (true) {
        getline(cin, str);
        if (str.empty()) {
            cout << "Ошибка! Введена пустая строка. Пожалуйста, введите слово." << endl;
            continue;
        }
        bool valid = true;
        for (char c : str) {
            if (isspace(c) || !((c >= 'A' && c <= 'Я') || (c >= 'a' && c <= 'я'))) {
                valid = false;
                break;
            }
        }
        if (valid) {
            return str;
        }
        else {
            cout << "Ошибка! Введите слово, содержащее только русские буквы и без пробелов!" << endl;
        }
    }
}
```

5.7 Исправленный код программы

HashTable.h

```
#pragma once
#include <vector>
#include <fstream>
#include <sstream>
#include <algorithm>
#include <cctype>

#include "LinkedList.h"

using namespace std;

class HashTable {

    private:

    static const int NOT_FOUND = -1;
    static const double REHASH_THRESHOLD;
    static const int MAX_WORDS = 10000;

    LinkedList* table;

    int total_buckets;
    int total_elements = 0;
    double fillability;

    unsigned int hashFunction(const string& key);

    public:

    HashTable(int size);
    void displayHashTable();
    void insert(const string& key);
    void remove(const string& key);
    int search(const string& key);
    void insertFromFile(const string& filename);

    void rehash();
    void clear();

    int getCount();
```

```
41
42      ~HashTable();
43
44  };
```

HashTable.cpp

```
1   #include "HashTable.h"
2   #include <locale>
3
4   bool isAlphaNum(char c) {
5
6       if (std::isalnum(static_cast<unsigned char>(c))) {
7           return true;
8       }
9
10      return (c >= 'A' && c <= 'Я') || (c >= 'a' && c <= 'я');
11  }
12
13
14  HashTable::HashTable(int size) {
15      total_buckets = size;
16      fillability = 0.0;
17      table = new LinkedList[total_buckets];
18  }
19
20  void HashTable::insert(const string& key) {
21
22      if (total_elements >= MAX_WORDS) {
23          cout << "Словарь переполнен!\n" << endl;
24          return;
25      }
26
27      int index = hashFunction(key);
28
29      if (NOT_FOUND == search(key)) {
30
31          table[index].insert(key);
32          total_elements++;
33          fillability = static_cast<double>(total_elements) / static_cast<
        ↪double>(total_buckets);
34      }
35
```

```
36      if (fillability >= REHASH_THRESHOLD) {
37          rehash();
38      }
39  }
40
41  void HashTable::remove(const string& key) {
42
43      int index = hashFunction(key);
44      if (table[index].remove(key)) {
45          total_elements--;
46          fillability = static_cast<double>(total_elements) / static_cast<
    ↪double>(total_buckets);
47      }
48
49      else cout << "\n Элемент не найден! " << endl;
50  }
51
52  int HashTable::search(const string& key) {
53
54      int index = hashFunction(key);
55      if (table[index].search(key)) return index;
56      else return NOT_FOUND;
57
58  }
59
60
61  void HashTable::displayHashTable() {
62      cout << endl;
63
64      if (total_elements == 0) {
65          cout << "Хэш-таблица пуста!" << endl;
66      }
67      else {
68          for (int i = 0; i < total_buckets; i++) {
69              cout << "[" << i << "]: ";
70              table[i].display();
71              cout << endl;
72          }
73      }
74  }
75
76
77  HashTable::~HashTable() {
78      delete[] table;
```

```
79 }
80
81
82 unsigned int HashTable::hashFunction(const string& key) {
83
84     unsigned int hash_value = 0;
85     int a = 33; // основание полинома
86
87     for (char c : key) {
88         hash_value = (hash_value * a + static_cast<unsigned int>(c)) %
    ↪total_buckets;
89     }
90
91     return hash_value;
92 }
93
94
95 void HashTable::insertFromFile(const string& filename) {
96
97     ifstream file(filename);
98
99     if (!file.is_open()) {
100         cout << "Ошибка! Не удалось открыть файл!" << filename << endl;
101         return;
102     }
103
104     string line, word;
105     while (getline(file, line)) {
106
107         stringstream ss(line);
108
109         while (ss >> word) {
110
111             transform(word.begin(), word.end(), word.begin(), ::tolower);
112             word.erase(remove_if(word.begin(), word.end(), [](char c) {
113                 return !isAlphaNum(c);
114             }), word.end());
115
116             if (!word.empty()) {
117                 insert(word);
118                 // cout << "Добавили " << word << endl;
119             }
120         }
121
```

```
122     }
123
124     cout << "Слова успешно добавлены! " << endl << endl;
125
126     // cout << "Количество элементов: " << total_elements << endl;
127     // cout << "Коэф. заполняемости хэш-таблицы: " << fillability << endl;
128 }
129
130
131 void HashTable::rehash() {
132
133     // cout << "Пересоздаем таблицу..." << endl;
134
135     int old_buckets = total_buckets;
136
137     total_buckets *= 2;
138
139     LinkedList* old_table = table;
140     table = new LinkedList[total_buckets];
141     total_elements = 0;
142
143     for (int i = 0; i < old_buckets; i++) {
144
145         ListNode* temp = old_table[i].getHead();
146
147         while (temp != nullptr) {
148             insert(temp->data);
149             temp = temp->next;
150         }
151
152     }
153
154     delete[] old_table;
155
156 }
157
158 void HashTable::clear() {
159
160     for (int i = 0; i < total_buckets; i++) {
161         table[i].clear();
162     }
163
164     total_elements = 0;
165     fillability = 0.0;
```

```
166
167     cout << "Хэш-таблица полностью очищена!" << endl;
168
169 }
170
171 int HashTable::getCount() {
172     return total_elements;
173 }
```

main.cpp

```
1
2 #include <iostream>
3 #include <algorithm>
4 #include <cctype>
5 #include <locale>
6 #include <windows.h>
7
8 #include "HashTable.h"
9 #include "check.h"
10
11 const int MAX_MENU_ITEMS = 6;
12 using namespace std;
13
14 int main() {
15     SetConsoleCP(1251);
16     SetConsoleOutputCP(1251);
17     setlocale(LC_ALL, "Russian");
18     HashTable myHashTable(10);
19
20     while (true) {
21
22         cout << endl << " Хэш-таблица\n";
23
24         cout << "\n  1. Вывести словарь на экран \n";
25         cout << "  2. Дополнить словарь из текстового файла \n";
26         cout << "  3. Добавить новое слово в словарь \n";
27         cout << "  4. Поиск слова в словаре \n";
28         cout << "  5. Удалить слово из словаря \n";
29         cout << "  6. Очистить словарь \n";
30
31         cout << "\n B+ дерево  \n\n";
32
```

```
33          cout << "  7. Вывести словарь на экран \n";
34          cout << "  8. Дополнить словарь из текстового файла \n";
35          cout << "  9. Добавить новое слово в словарь \n";
36          cout << "  10. Поиск слова в словаре \n";
37          cout << "  11. Удалить слово из словаря \n";
38          cout << "  12. Очистить словарь \n";
39
40          cout << "  0. Выход из программы\n";
41
42          cout << "\n Выберите действие: ";
43
44          int menuChoice = checkingInput(MAX_MENU_ITEMS);
45          string word;
46          string path;
47          bool flag;
48          int index;
49
50          switch (menuChoice) {
51
52              case 1:
53
54              myHashTable.displayHashTable();
55              cout << endl << "Количество слов в словаре: " << myHashTable.
    ↪getCount() << endl;
56              break;
57
58              case 2:
59
60              myHashTable.insertFromFile("story.txt");
61              break;
62
63              case 3:
64              cout << "Введите слово (без пробелов), которое хотите добавить:
    ↪ " << endl;
65              clear();
66              word = checkingString();
67              transform(word.begin(), word.end(), word.begin(), ::tolower);
68
69              index = myHashTable.search(word);
70              if (index != -1) {
71                  cout << "Слово уже есть в словаре!" << endl;
72              }
73              else {
74                  myHashTable.insert(word);
```

```
75                cout << "Слово успешно добавлено! " << endl;
76
77            }
78
79            break;
80
81            case 4:
82
83            cout << "Введите слово, которое хотите найти: " << endl;
84            clear();
85            word = checkingString();
86            transform(word.begin(), word.end(), word.begin(), ::tolower);
87
88            index = myHashTable.search(word);
89            if (index != -1) {
90                cout << "Слово " << word << " присутствует в словаре! " <<
    ↪endl;
91                cout << "Индекс бакета: " << index << endl << endl;
92            }
93            else cout << "Слово " << word << " отсутствует в словаре! " <<
    ↪endl << endl;
94
95            break;
96
97            case 5:
98
99            cout << "Введите слово, которое хотите удалить: " << endl;
100           clear();
101           word = checkingString();
102           transform(word.begin(), word.end(), word.begin(), ::tolower);
103
104           myHashTable.remove(word);
105           break;
106
107           case 6:
108           myHashTable.clear();
109           break;
110
111           case 0:
112           return 0;
113
114           default:
115           cout << "Ошибка!";
116       }
```

```
117
118             system("pause");
119         }
120
121         return 0;
122 }
```

Вывод

В результате проделанной работы программный код был исправлен в соответствии с рекомендациями специалиста по тестированию. Были сделаны личные выводы о том, что ручное тестирование позволяет выявить некорректность как в логике работы программы, так и в стиле её оформления. Был получен опыт процесса инспекции кода.

Список литературы

1. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер. — Изд. 3-е. — Санкт-Петербург : Диалектика, 2012. — 272 с.