Федеральное государственное бюджетное образовательное
учреждение высшего образования
РЫБИНСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ П.А. СОЛОВЬЕВА

Факультет радиоэлектроники и информатики
Кафедра математического и программного обеспечения электронных
вычислительных средств

КУРСОВАЯ РАБОТА

по дисциплине
ТЕСТИРОВАНИЕ И ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

Студент группы ИПБ-18 Бавглей К.И.

Руководитель Овсянников Т.С.

18.04.22

Рыбинск 2022

Оглавление

1. Постановка задачи

1. Выбрать программу для тестирования. Кратко описать суть программы, ее функционал, интерфейсы/модули, которые предполагается тестировать.

2. Создать наборы тестов для тестирования по критериям C0, C1, C2, управляющий граф программы, описать наборы входных данных для путей графа.

3. Обосновать и реализовать набор модульных тестов (применить и предоставить результаты тестирования).

4. Обосновать и реализовать набор интеграционных тестов (применить и предоставить результаты тестирования).

5. Разработать все необходимое для проведения системного тестирования (система тестов для функционального и нефункционального тестирования, предложения по автоматизации тестирования).

2. Теоретические сведения

2.1. Структурное тестирование

Метод структурного тестирования предполагает создание тестов на основе структуры системы и ее реализации. Такой подход иногда называют тестированием методом "белого ящика". Как правило, структурное тестирование применяется к относительно небольшим программным элементам, например, к подпрограммам или методам, ассоциированным с объектами. При таком подходе испытатель анализирует программный код и для получения тестовых данных использует знания о структуре компонента.

При структурном тестировании применяется метод тестирования ветвей - метод, при котором проверяются все независимо выполняемые ветви компонента или программы. Он основывается на графе потоков управления программы. Управляющий граф программы (УГП) — связный ориентированный ациклический граф. Вершины этого графа — выражения в исходном коде, дуги — переход к следующему выражению в исходном коде.

Для структурного тестирования существуют критерии C0, C1, C2.

Критерий C0 (критерий тестирования команд) — набор тестов, в совокупности обеспечивающий прохождение каждой вершины УГП не менее одного раза.

Критерий C1 (критерий тестирования ветвей) — набор тестов, в совокупности обеспечивающий прохождение каждой ветви УГП не менее одного раза.

Критерий C2 (критерий тестирования путей) — набор тестов, в совокупности обеспечивающий прохождение каждого пути УГП не менее одного раза.

2.2. Модульное тестирование

Модульное тестирование (Unit Testing) – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Его цель заключается в том, чтобы проверить, что каждая единица программного кода работает должным образом. Данный вид тестирование выполняется разработчиками на этапе кодирования приложения. Модульные тесты изолируют часть кода и проверяют его работоспособность. Единицей для измерения может служить отдельная функция, метод, процедура, модуль или объект.

Модульное тестирование основывается на создании фиктивных объектов для тестирования фрагментов кода, которые еще не являются частью

законченного приложения. Подставные объекты заполняют недостающие части программы. Например, у вас может быть функция, которая нуждается в переменных или объектах, которые еще не созданы. В модульном тестировании они будут учитываться в форме фиктивных объектов, созданных исключительно для целей модульного тестирования, выполненного в этом разделе кода.

2.3. Интеграционное тестирование

Интеграционное тестирование – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа. Как правило, программный продукт состоит из нескольких программных модулей, написанных разными программистами.

Целью интеграционного тестирования является выявление багов при взаимодействии между этими программными модулями и в первую очередь направлен на проверку обмена данными между этими самими модулями. Интеграционное тестирование отличается от других видов тестирования тем, что он сосредоточен в основном на интерфейсах и потоке данных (между модулями). Здесь приоритет проверки присваивается интегрирующим ссылкам, а не функциям блока, которые уже проверены.

2.4. Системное тестирование

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д.

Можно выделить два подхода к системному тестированию:

- на базе требований (requirements based). Для каждого требования пишутся тестовые случаи (test cases), проверяющие выполнение данного требования.
- на базе случаев использования (use case based). На основе представления о способах использования продукта создаются случаи использования системы (Use Cases). По конкретному случаю использования можно

определить один или более сценариев. На проверку каждого сценария пишутся тест кейсы (test cases), которые должны быть протестированы.

3. Описание тестируемой системы

Для проведения тестирования была выбрана программа, реализованная для курсовой работы на тему "Хеширование". Данная программа представляет собой консольное приложение для работы с хеш-таблицами. В приложении доступны следующие функции:

- Создание таблиц с заданными параметрами;
- Добавление новой строки в таблицу;
- Поиск строки в таблице;
- Вывод статистики по таблице;
- Чтение таблиц из файла;
- Вывод всех существующих таблиц.

4. Структурное тестирование

Для проведения структурного тестирования была выбрана подпрограмма main() – основная рабочая функция программы.

Для начала строится управляющий граф подпрограммы.

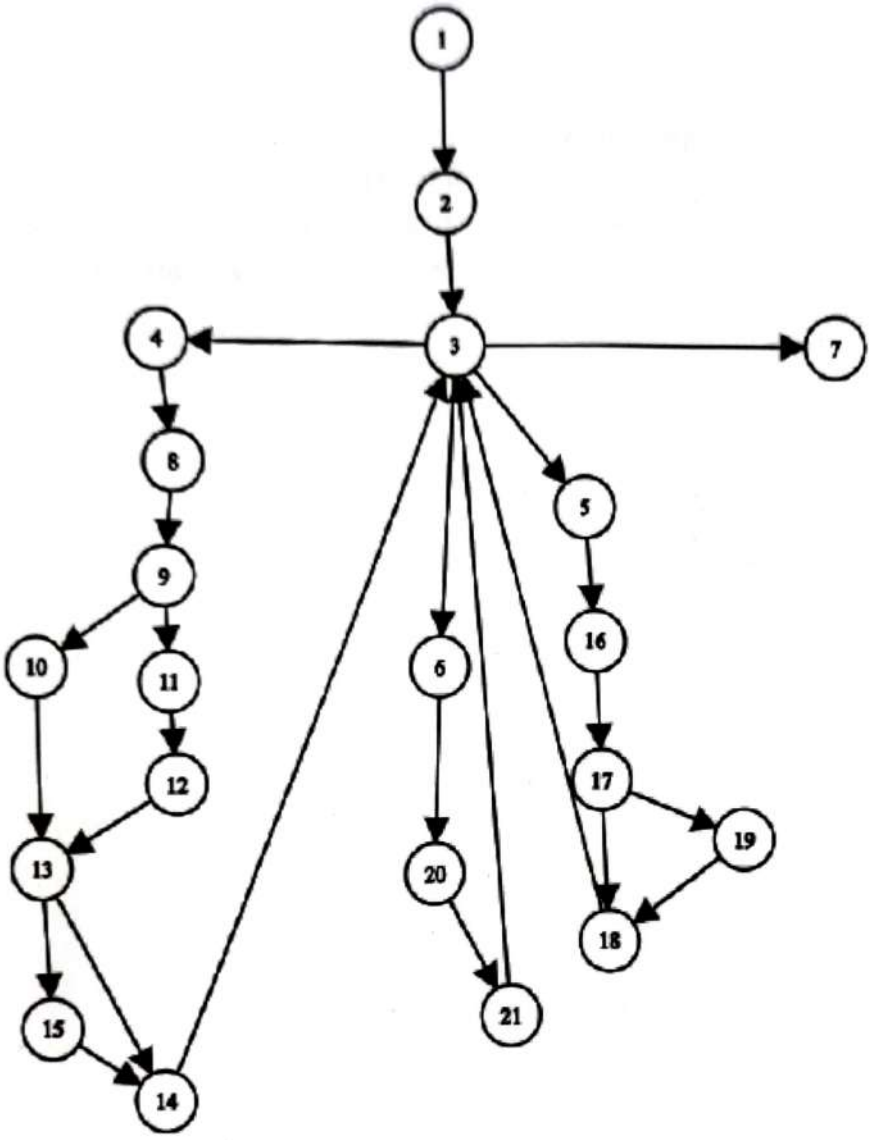

Рисунок 1 - УГП main

Список путей для выполнения критериев:

1) C0 : 1-2-3-4-8-9-10-13-15-14-3-4-8-9-11-12-13-14-3-6-20-21-3-5-16-17-19-18-3-7
2) C1: 1-2-3-4-8-9-10-13-15-14-3-6-20=21-3-5-16-17-19-18-3-7

   1-2-3-4-8-9-11-12-13-14-3-6-20-21-3-5-16-17-18-3-7

3) C2: 1-2-3-4-8-9-10-13-15-14-3-7
   1-2-3-4-8-9-11-12-13-14-3-7
   1-2-3-6-20-21-3-7
   1-2-3-5-16-17-18-3-7
   1-2-3-5-16-17-19-18-3-7
   1-2-3-7

В ходе структурного тестирования был спроектирован управляющий граф программы и были конкретизированы условия критерий тестирования команд, ветвей и путей. Так как пути тестов покрывают C0, C1 и C2, то возможно провести максимально полное тестирование этого графа.

5. Модульное тестирование

Для проведения модульного тестирования был выбран модуль Hash_table. Этот модуль предназначен для реализации хэш-таблиц. Модуль содержит в себе один класс и несколько методов, и работает корректно только в том случае, если все его части работают корректно.

Hash_table. Add – функция, добавляющая строки в таблицу.

Крайние случаи: Входные данные не строка

Общие случаи: Любая входящая строка добавляется в таблицу

Hash_table. Search – функция, которая производит поиск строки в таблице

Крайние случаи: Входные данные не строка.

Общие случаи: Если входные данные находятся в таблице

Входных данных нет в таблице

Hash_table.Delete – функция, удаляющая элементы из таблицы

Крайние случаи: Входные данные не строка.

Общие случаи: Входные данные находятся в таблице, и могут быть удалены

Входных данных нет в таблице, и они не могут быть удалены

Hash_table.hushFunc – эта функция принимает на вход строку данных, проходящих хеширование и номер метода, которым оно производится.

Крайние случаи: Некорректный формат входных данных

Общие случаи: Номер метода находится вне интервала разработанных методов

Номер метода находится в интервале разработанных методов

Hash_table.integerCastBytes – данная функция является хэш-функцией, эта функция принимает входную строку данных, проходящих хэширование, после чего возвращает результат своего выполнения. Результат получается путём побитового хеширования.

Общие случаи: Возвращается результат, полученный путём побитового хеширования

Hash_table.integerCastSymbols – данная функция является хэш-функцией, эта функция принимает входную строку данных, проходящих хеширование, после

чего возвращает результат своего выполнения. Результат получается путём сложения кодов входных символов хеширования.

Общие случаи: Возвращается результат хеширования, полученный путём суммирования кодов

Hash_table.integerFromPolynomial – данная функция является хэш-функцией, эта функция принимает входную строку данных, проходящих хеширование, после чего возвращает результат своего выполнения. Результат получается путём полиномиального хеширования.

Общие случаи: Возвращается результат хеширования, полученный путём полиномиального хеширования.

Код тестов предоставлен в Приложении к курсовой работе.

Результаты тестирования в Test Explorer:

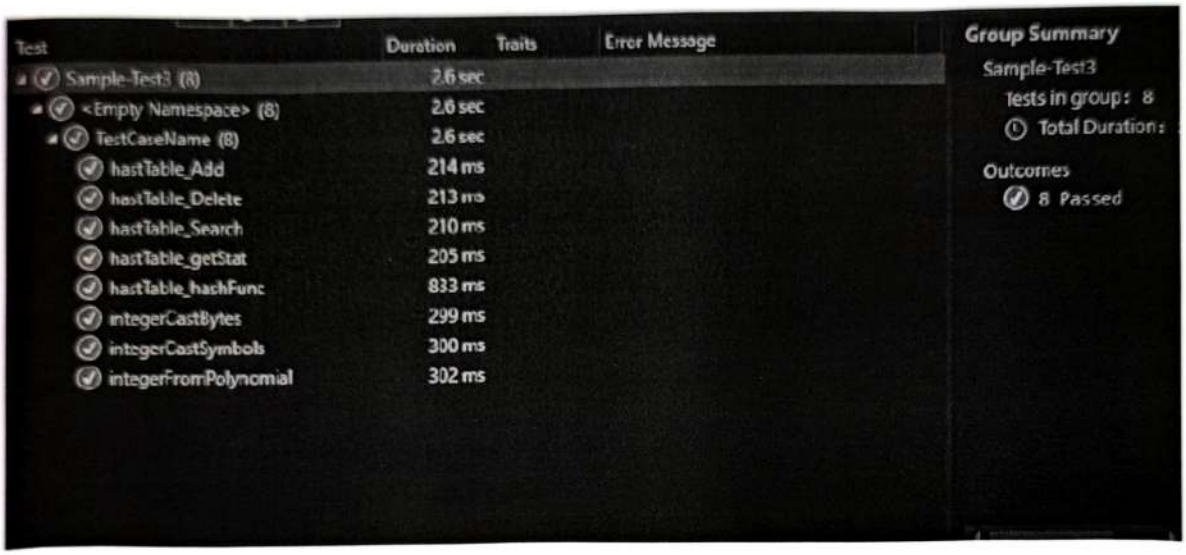

Рисунок 2 - Модульное тестирование

Результаты модульного тестирования показали, что все части модуля работают корректно.

6. Интеграционное тестирование

Для интеграционного тестирования была выбрана подпрограмма main(). Данная функция является основной исполняемой подпрограммой. Это наиболее важная функция так как она предоставляет пользователю интерфейс по работе с хэш-таблицами и вызывает остальные исполняемые функции. Для доказательства корректности работы основного модуля нужно протестировать на корректность работы все его составляющие. Подпрограммы для тестирования:

Hash_table - Функция конструктор объекта Hash_table

Общие случаи: Должна быть создана таблица с заданными входными параметрами как режим работы и её название

Ошибка при создании

Hash_table.Add - Корректность работы данной функции была протестирована модульными тестами.

Hash_table.Search - Корректность работы данной функции была протестирована модульными тестами.

Hash_table.Delete - Корректность работы данной функции была протестирована модульными тестами.

Hash_table.getStat - Данная функция производит вывод данных и статистики по таблице.

Общие случаи: Вывод статистики

Неудачный вывод статистики

Hash_table.Interface - Данная функция является интерфейсом для взаимодействия с хэш-таблицей и её методами

Общие случаи: Пользователь выбрал доступное действие, и оно корректно отработало

Пользователь выбрал доступное действие, и функция некорректно завершила свою работу

Пользователь выбрал недоступное действие программа сообщила об ошибке

Hash_table.getName - Данная функция возвращает поля name

Общие случаи: Возвращается название таблицы

Hash_table.getSize - Данная функция возвращает поля size

Общие случаи: Возвращается размер таблицы

list - Функция конструктор побочного объекта list

Общие случаи: Должен быть создан объект список

Ошибка при создании объекта

list.Add - Данная функция производит добавление новых элементов в список

Общие случаи: Любая входящая строка должна быть добавлена в таблицу.

list.Search - Данная функция производит поиск в списке элементов и возвращает соответствующее хэш-значение

Общие случаи: в случае если входные данные находятся в списке

В случае, когда входных данных нет в таблице

list.Delete - Данная функция осуществляет удаление элемента из списка.

Общие случаи: в случае если входные данные находятся в списке, и могут быть удалены

В случае, когда входных данных нет в списке, и они не могут быть удалены

Node - Функция конструктор побочного объекта Node, из которого составляется объект list, хранит в себе данные и указатели на следующие элементы в списке

Общие случаи: Должен быть создан объект список

Ошибка при создании объекта

Node.setChain -Данная функция производит установку ссылки на следующий элемент в цепи

Крайние случаи: Добавляемый элемент null

Общие случаи: Добавление элемента в цепь

Ошибка во время добавления элемента в цепь

Node.setNext - Данная функция производит установку на следующий элемент в списке

Крайние случаи: Добавляемый элемент null

Общие случаи: Добавление элемента в качестве следующего

Ошибка во время добавления элемента

Код тестов предоставлен в Приложении к курсовой работе.

Результаты тестирования в Test Explorer:

Рисунок 3 - Интеграционное тестирование

Результаты интеграционного тестирования показали, что все составляющие main работают корректно.

7. Системное тестирование

Для системного тестирования могут быть выделены следующие функциональные требования к программе:

1. Программа должна позволять создавать таблицы с тремя параметрами, названием, режимом хеширования и размером таблицы

2. Программа должна выводить текущие созданные таблицы и их параметры

3. Программа должна позволять взаимодействовать с таблицей применяя следующие методы

- Добавление новой строки
- Поиск строки
- Удаление строки
- Вывод статистики по таблице
- Считать таблицу из файла

А также нефункциональное требование:

1. Интерфейс и взаимодействие с программой должно быть понятным и простым для пользователя

Тестирование функциональных требований программы было произведено на уровне интеграционного тестирования, которое показала корректность работы модулей и их взаимосвязей. Так как приложение является консольным, и имеет мало функций было принято решение применить ручное тестирование. Во время ручного тестирования были проделаны следующие действия:

Создание таблицы с названием Table режимом работы хеширования с побитовым хешированием максимальным размером до 100 элементов.

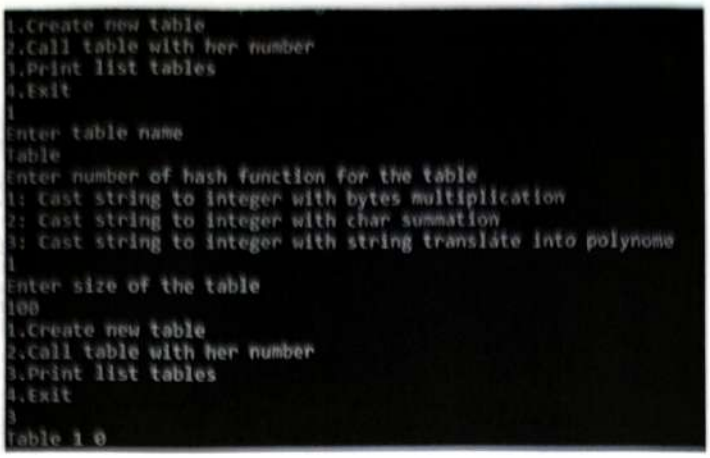

Рисунок 4 – Создание таблицы

Выведен список всех существующих таблиц, т.е. таблицы с названием Table, её режимом работы и количеством находящихся в ней элементов 0.

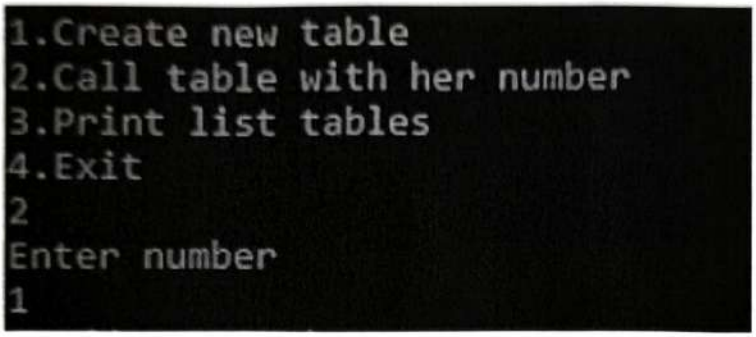

Рисунок 5 - Вывод таблицы

После этого мы обращаемся к этой таблице и получаем список возможных методов взаимодействия с таблицей. Был выбран вариант добавления нового элемента в список, строки string которая получила значение хеширования 23 и была выведена при помощи метода вывода статистики таблицы.

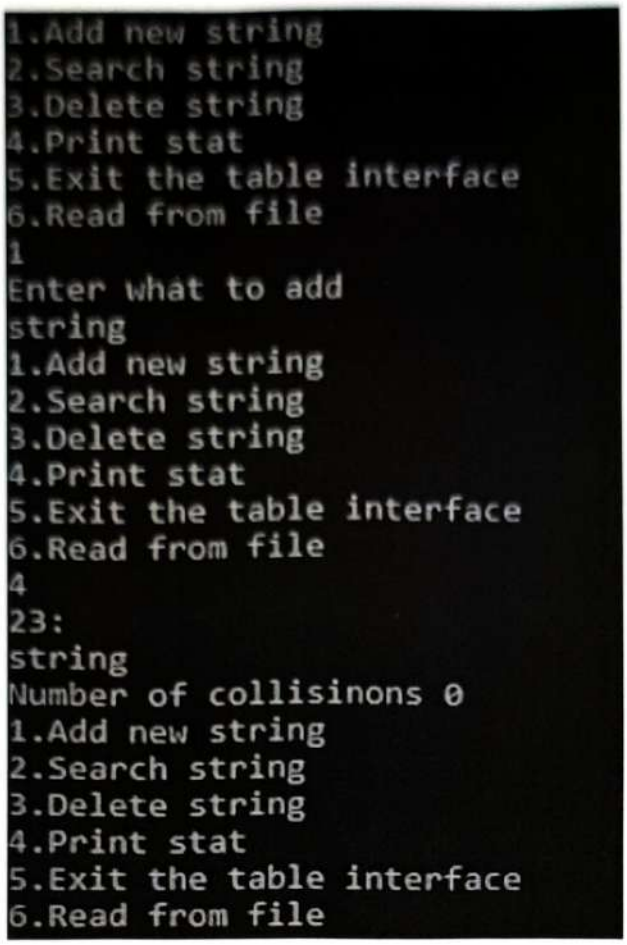

Рисунок 6 - добавление элемента

После этого был произведен поиск данной строки, и она была найдена.

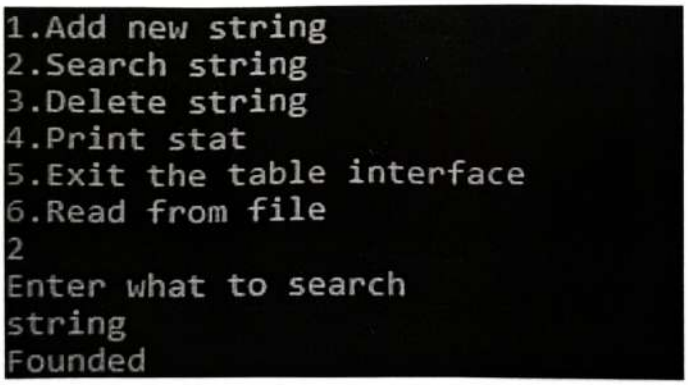

Рисунок 7 - Поиск строки

Удаляем строку string.

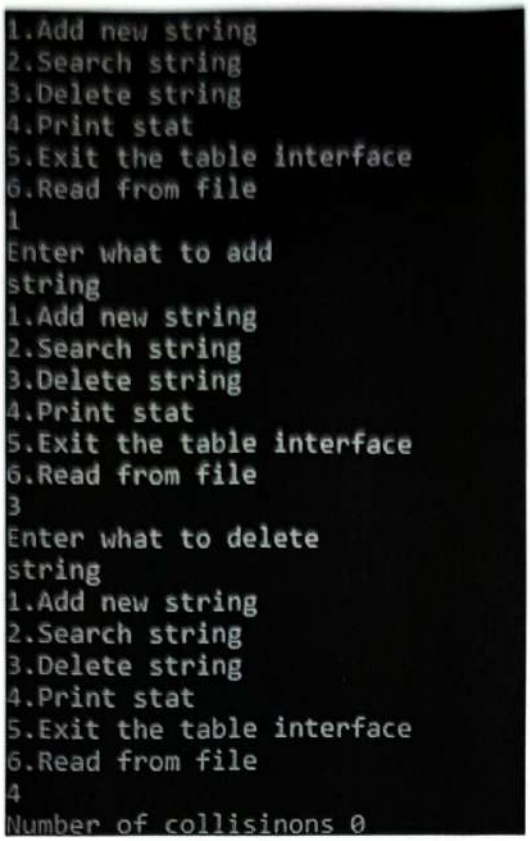

Рисунок 8 - Удаление строки

В итоге, путём ручного тестирования были проделаны действия доступные пользователю, и тем самым была подтверждена корректность работы программы. Автоматизация системного тестирования для данной программе будет излишней, так как ее объем позволяет проделать все действия вручную за короткое время.

8. Вывод

В ходе курсовой работы были получение навыки разработки тестов программного обеспечения. Были проведены структурное, модульное, интеграционное и системное тестирования. Во время структурного тестирования был спроектирован управляющий граф программы и были конкретизированы условия критерий тестирования команд, ветвей и путей. Так как пути тестов покрывали C0, C1 и C2, то это дало возможность провести максимально полное тестирование этого графа. В ходе модульного и интеграционного тестирования тестируемые модули корректно прошли все тесты. В ходе системного тестирования были проверены функциональные и нефункциональные требования к программе.

Можно сделать вывод, что программа полностью протестирована и готова для дальнейшего использования.

ПРИЛОЖЕНИЕ

```
#include "pch.h"
#include "../Sample -Test3/Hash_table
Node <string >* search_res;
TEST(Integr , hash) {
hash_table <string > hash = new hash_table <string >(1, 1, "    ");
EXPECT_NE(hash ,nullptr);
 }
TEST(Unit , hastTable_Add) {
hash_table <string > hash = new hash_table <string >(1, 1, "    ");
hash.Add("    ");
 }
TEST(Unit , hastTable_Search) {
hash_table <string > hash = new hash_table <string >(1, 1, "     ");
hash.Add("   ");
search_res=hash.Search("  ");
EXPECT_NE(search_res , nullptr);
}
TEST(Unit , hastTable_Delete) {
hash_table <string > hash = new hash_table <string >(1, 1, "   ");
hash.Add("   ");

hash.Delete("   ");
search_res = hash.Search("   ");
EXPECT_EQ(search_res , nullptr);
}
TEST(Unit , hastTable_getStat) {
hash_table <string > hash = new hash_table <string >(1, 1, "   ");
hash.getStat();
}
TEST(Integr , list) {
list <string > my_list = new list <string >(2);
EXPECT_NE(my_list , nullptr);
}
TEST(Integr , list_Add) {
```

```
list <string > my_list = new list <string >(2);
my_list.AddNewNext(1, "    ");
}
TEST(Integr , list_Search) {
list <string > my_list = new list <string >(2);
my_list.AddNewNext(1, "   ");
search_res= my_list.Search(1, "  ");
EXPECT_NE(search_res ,nullptr);
}
TEST(Integr , list_Delete) {
list <string > my_list = new list <string >(2);
my_list.AddNewNext(1, "   ");
my_list.Delete(1, "      ");
search_res = my_list.Search(1, "   ");
EXPECT_EQ(search_res , nullptr);
}
TEST(Integr , Node) {
Node <string > my_Node = new Node <string >(2);
EXPECT_NE(my_Node , nullptr);
}
TEST(Integr , Node_setNext) {
Node <string > my_Node = new Node <string >();
Node <string > secondNode = new Node <string >();
my_Node.setNext(secondNode);
EXPECT_NE(myNode.getNext(), nullptr);
}
TEST(Integr , Node_setChain) {
Node <string > my_Node = new Node <string >();
Node <string > secondNode = new Node <string >();
my_Node.setChain(secondNode);
EXPECT_NE(myNode.getChain(), nullptr);
}
TEST(Integr , hash_getName) {
hash_table <string > hash = new hash_table <string >(1, 1, "  ");
EXPECT_EQ(hash.getName()," ");
}
TEST(Integr , hash_getSize) {
```