Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Выполнил:

студент ИИТММ гр. 381706-2

Антипин А.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Создание и обработка текста

Отчет по лабораторной работе

Нижний Новгород

2019 г.

**Содержание**

[1.Введение 3](#_Toc4712657)

[2.Постановка целей и задач 4](#_Toc4712658)

[3.Руководство пользователя 5](#_Toc4712659)

[4.Руководство программиста 7](#_Toc4712660)

[4.1.Описание структуры программы 7](#_Toc4712661)

[4.2.Описание структур данных 8](#_Toc4712662)

[4.3.Описание алгоритмов 11](#_Toc4712663)

[5.Эксперименты 13](#_Toc4712664)

[5.1.Время выполнения 13](#_Toc4712665)

[6.Заключение 15](#_Toc4712666)

[7.Литература 16](#_Toc4712667)

# 1.Введение

Хранение и обработка текста является очень актуальной задачей на протяжении всего существования компьютеров. Линейное хранение массива символьных элементов не является удобным, так как для доступа к последним элементам массива потребуется пройти все предыдущие элементы, а если появляется необходимость вставить какой-либо элемент внутрь массива, то необходимо сдвигать оставшуюся часть, или если текущей выделенной памяти не хватает, выделять её заново. В связи с этим, для эффективного выполнения операций с тестовой информацией необходимо выбрать её представление, обеспечивающее структурирование и быстрый̆ доступ к различным элементам текста.

В рамках лабораторной работы рассматривается задача разработки учебного редактора текстов, в котором для представления данных используется иерархический связный список. Подобная иерархическая структура представления может применяться при компьютерной реализации математических моделей в виде деревьев и, тем самым, может иметь самое широкое применение в самых различных областях приложений.

# 2.Постановка целей и задач

Целью лабораторной работы является создание структуры хранения текста типа «TTree» и обработки теста типа «TText» и методов работы с ними, таких как:

* Создание текста или добавление в уже существующий;
* Удаление элементов текста;
* Копирование;
* Поиск по тексту.

Для реализации алгоритмов будет использоваться 3 шаблонных класса:

* TTree;
* TTreeIterator;
* TText.

Для проверки правильности работы этих классов будут написаны тесты с использованием Фреймворка Google Test, а также тестовый образец программы, которая будет использует класс

# 3.Руководство пользователя

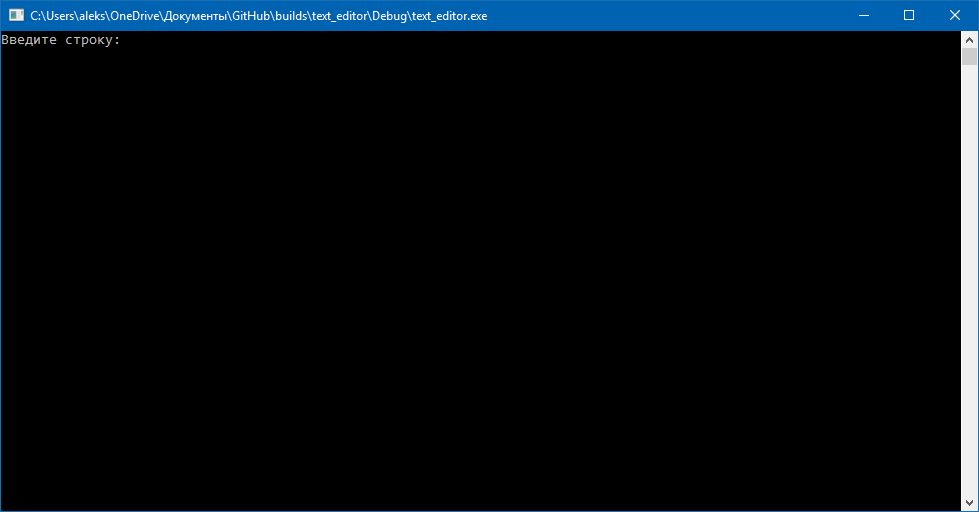
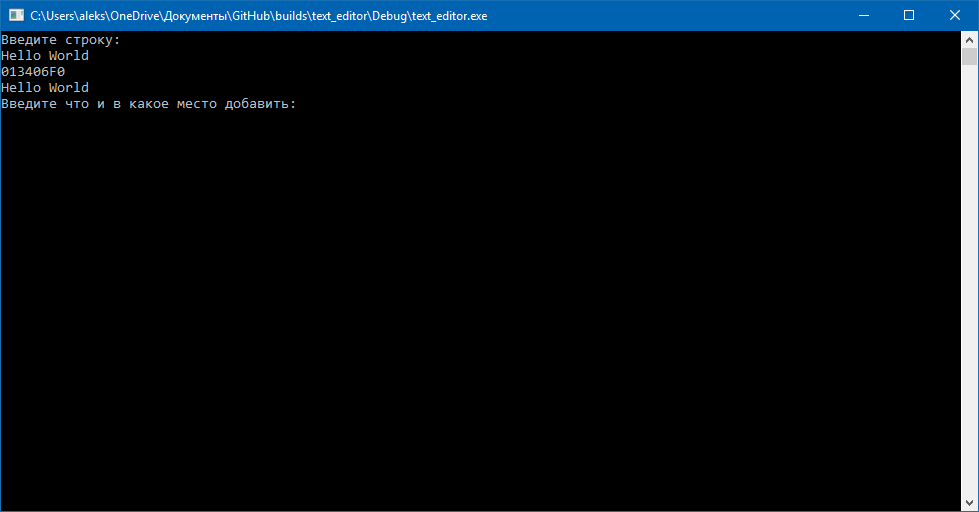
После запуска программы пользователя встречает консольное окно (рис. 1):

рис. 2 (тестирование создания и вывода структуры текст)

рис. 1 (вывод программы тестирования списка для пользователя)

После ввода строки и нажатия Enter, будет создана структура «текст» и вывод ее содержания на экран (рис. 2).

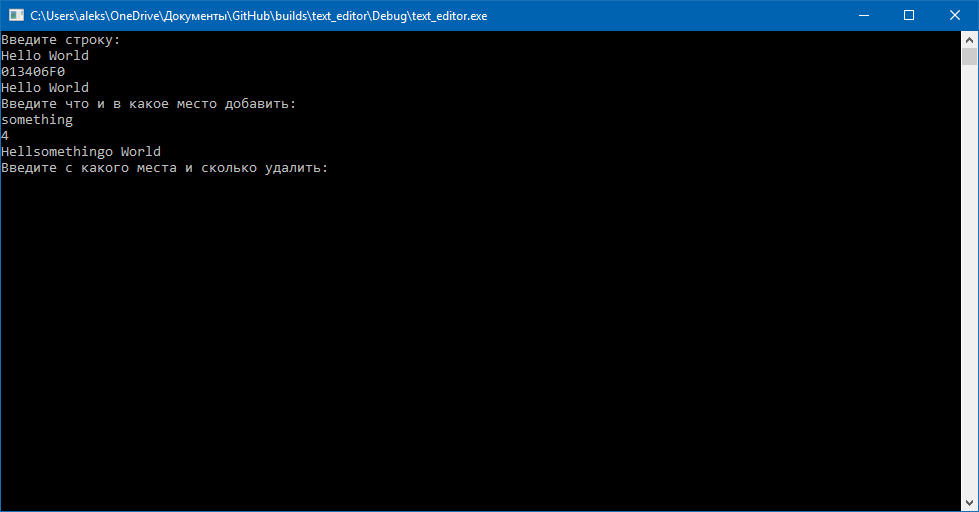
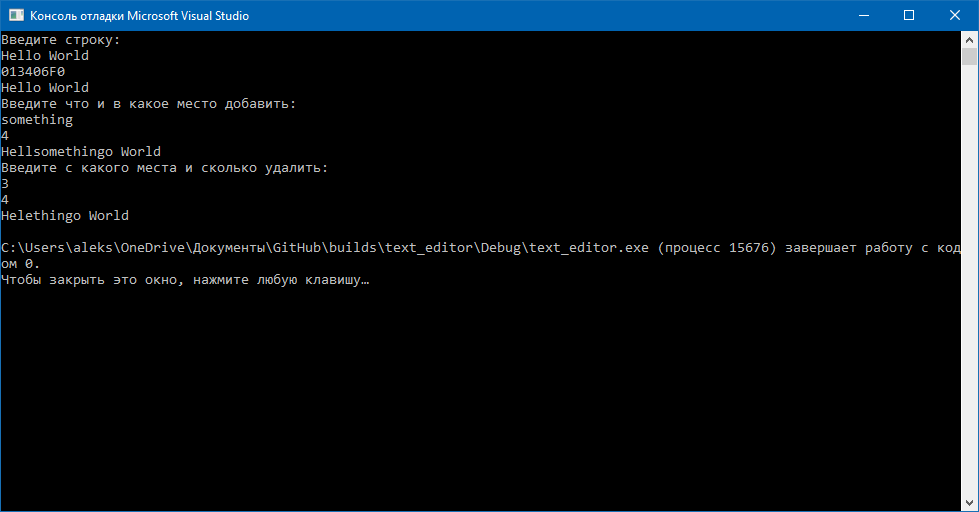
После этого пользователь может проверить работу вставки в тест. Для этого нужно ввести слово, которое надо вставить и позицию куда (рис. 3).

рис. 4 (тестирование удаления текста)

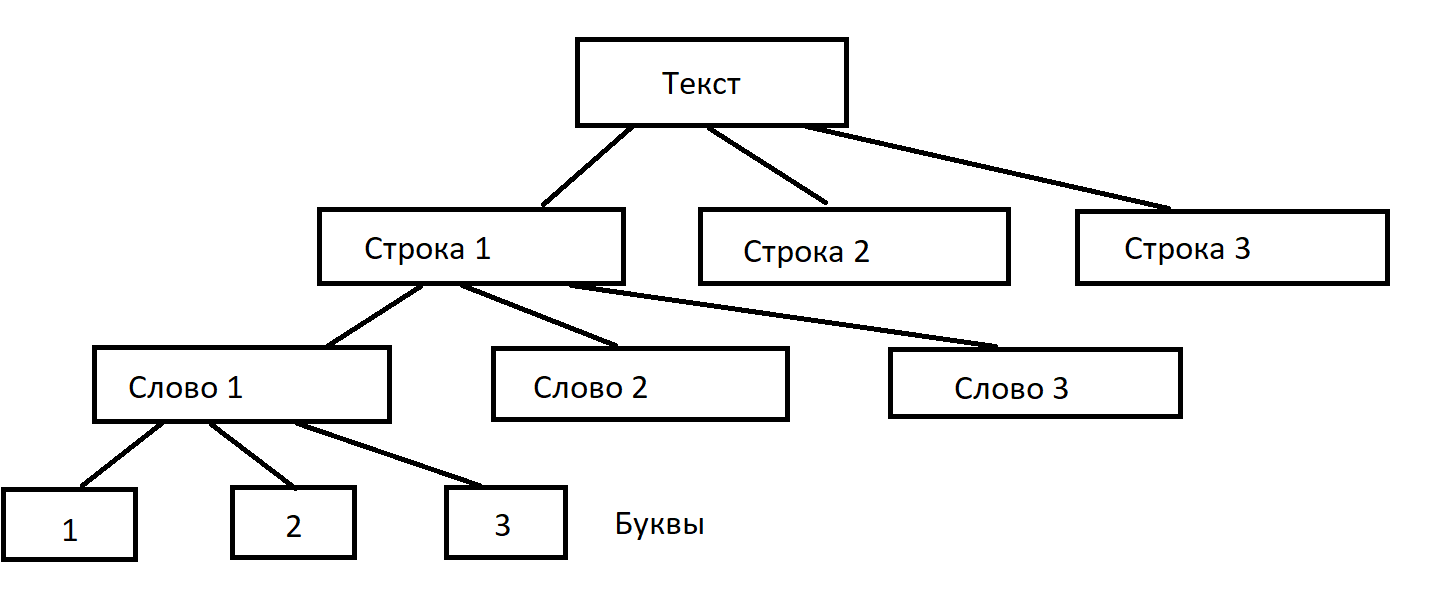
рис. 3 (тестирование вставки текста в структуру)

После проверки на вставку будет идти проверка на удаление (рис. 4), где надо ввести сначала позицию, начиная с которой начнется удаление, а затем количество элементов. После последней проверки программа завершает свою работу.

# 4.Руководство программиста

## 4.1.Описание структуры программы

Структура текста будет реализована с помощью дерева, каждый элемент которого имеет свой уровень (всего их 4), букву (только на 3 уровне), указатель на элемент следующего уровня и указатель на элемент текущего уровня:



То есть для реализации алгоритмов будет использовано 3 класса:

* Класс «Дерево» (TTree);
* Класс «Итератор» (TTreeIterator);
* Класс «Текст» (TText).

А также проект использующий фреймворк Google Test, для проверки правильности работы этих классов и тесовый проект, который будет показываться пользователю.

**Класс TTree:**

Класс TTree будет реализован с помощью линейного списка элементов дерева. Каждый элемент имеет параметр уровня:

* 1 уровень — уровень текста;
* 2 уровень — уровень строки;
* 3 уровень — уровень слова;
* 4 уровень — уровень буквы.

Все элементы дерева на разных уровнях имеют одинаковую структуру, но только на уровне буквы можно хранить буквы. Остальные уровни служат для правильной связи теста.

**Класс TTreeIterator:**

Класс TTreeIterator будет осуществлять проход по дереву, используя метод поиска в глубину.

**Класс TText:**

Класс TText будет реализовывать методы, связанные с обработкой текста.

**Класс gtest:**

Класс gtest реализует тестирование классов TTree, TTreeIterator и TText, по средствам Фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов.

**Проект Text:**

В данном проекте реализован пример использования возможностей класса TText, доступный для пользователя.

## 4.2.Описание структур данных

**Класс TTree:**

TTree\* same\_level - указатель на следующий элемент текста;

TTree\* next\_level - указатель на внутренний элемент в текущем элементе текста;

char letter – буква;

int level - текущий уровень структуры;

static char\* memory – статическая общая память под все элементы дерева;

static TTree\* start – указатель на начало общей памяти;

static TTree\* end – указатель на конец общей памяти;

static TTree\* cur\_free – указатель на текущий элемент в общей памяти;

static int tree\_size – максимальное количество элементов в общей памяти;

static int busy\_tree\_size – текущее количество занятых элементов;

Exceptions\_from\_tree\_text exception – поле для вызова исключений.

**Класс TTreeIterator:**

TTree\* root – указатель на корень дерева, которое поступило в итератор;

TTree\* current – указатель на текущий элемент в итераторе;

TStackList<TTree\*> stack – стек, для реализации обхода в глубину.

**Класс TText:**

TTree\* root – указатель на корень дерева, которое образовано для реализации текста.

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| TTree::TTree(const int \_level) | Конструктор с параметром для класса TTree, который принимает уровень для образования элемента дерева. |
| TTree::TTree(const char\* word) | Конструктор дерева, который принимает массив символов, и от них формирует мини дерево – слово. |
| TTree::TTree(const char \_letter) | Конструктор дерева от буквы. |
| TTree::TTree(const TTree& tree) | Конструктор копирования дерева. |
| TTree::~TTree() | Деструктор дерева. |
| TTree& TTree::operator=(const TTree& tree) | Перегрузка оператора равно для дерева. |
| TTree& TTree::operator+=(const TTree& tree) | Перегрузка оператора «увеличения на», когда прибавляется другое дерево. |
| TTree& TTree::operator+=(const char \_letter) | Перегрузка оператора «увеличения на», когда прибавляется буква. |
| TTree& TTree::operator+=(const char\* word) | Перегрузка оператора «увеличения на», когда прибавляется слово. |
| void\* TTree::operator new (const unsigned int size) | Перегрузка оператора выделения памяти. |
| void TTree::operator delete(void\* tree) | Перегрузка оператора освобождения памяти. |
| ostream& operator<<(ostream& o, TTree& tree) | Перегрузка оператора вывода для дерева. |
| void TTree::Output() | Функция, которая позволяет вывести текст на консоль. |
| void TTree::SetSameLevel(TTree\* \_same\_level) | Метод, который позволяет задать новый same level для текущего элемента дерева. |
| void TTree::SetNextLevel(TTree\* \_next\_level) | Метод, который позволяет задать новый next level для текущего элемента дерева. |
| void TTree::SetLetter(const char \_letter) | Метод, который позволяет задать новую букву для текущего элемента дерева. |
| void TTree::SetLevel(const int \_level) | Метод, который позволяет задать новый уровень для текущего элемента дерева. |
| void TTree::SetTreeSize(const int size) | Статический метод, который позволяет задать максимальное количество элементов в дереве. |
| TTree\* TTree::GetSameLevel() | Получение same level текущего элемента. |
| TTree\* TTree::GetNextLevel() | Получение next level текущего элемента. |
| char TTree::GetLetter() | Получение буквы текущего элемента. |
| int TTree::GetLevel() | Получение уровня текущего элемента. |
| int TTree::GetTreeSize() | Получение текущего количества свободных элементов дерева. |
| int TTree::GetTreeBusySize() | Получение текущего количества занятых элементов дерева. |
| char\* TTree::ToString() | Преобразование текущего дерева в массив символов. |
| TTree\* TTree::Clone() | Создание копии текущего дерева. |
| void TTree::Initialization(const int size) | Выделение памяти для всего дерева. |
| void TTree::GarbageCollector() | Сбор «ненужных» элементов дерева, для их последующего использования. |
| void TTree::ClearMemory(void) | Очистка всей выделенной памяти. |
| TTreeIterator::TTreeIterator(TTree\* \_root) | Конструктор для класса итератор, который принимает корень дерева. |
| TTree\* TTreeIterator::GoNext() | Метод перехода к следующему элементу дерева. |
| void TTreeIterator::Reset() | Возврат всех полей итератора в исходное состояние. |
| bool TTreeIterator::IsEnd() | Проверка на окончание дерева в итераторе. |
| void TTreeIterator::PutInStack (TTree\* tree) | Метод принудительного пополнения стека итератора элементом. |
| TTree\* TTreeIterator::operator()() | Перегрузка оператора круглые скобки для итератора, который возвращает текущий элемент итератора. |
| TText::TText(const char\* string) | Конструктор с параметром для класса тест, который принимает строку из символов. |
| TText::TText(TText& text) | Конструктор копирования текста. |
| TText::TText(TTree& tree) | Конструктор преобразования типа (из дерева в текст). |
| TTree\* TText::GetRoot() | Получение корня текста. |
| void TText::Insert(const int pos, const char\* string) | Метод вставки строки в текст по заданной позиции. |
| void TText::Insert(TTree\* start, TTree\* string) | Метод ставки строки в текст по заданному элементу дерева. |
| int TText::Find(const char\* string) | Метод нахождения позиции для строки в тексте. |
| TTree\* TText::FindTree(const char\* string) | Метод нахождения элемента дерева по строке текста. |
| char\* TText::Copy(const int start, const int len) | Метод копирования строки текста по ее позиции и длине. |
| TTree\* TText::Copy(TTree\* start, const int len) | Метод копирования элементов дереве, по указателю на элемент дерева и количеству букв. |
| void TText::Delete(const int start\_del, const int lenght) | Метод удаления элементов дерева по начальной позиции и длине. |
| void TText::Delete(TTree\* start\_del, const int lenght) | Метод удаления элементов дерева по начальному элементу дерева и длине удаляемой строки. |

## 4.3.Описание алгоритмов

**Подробное описание некоторых методов**

Перевод из дерева в массив символьного типа:

* Создание стека с указателями на элементы дерева и помещение в него корня.
* Прохождение по дереву и подсчёт количества букв в переводимой строке.
* Создание массива символов необходимого размера и помещение в стек корня дерева.
* Прохождение по дереву, если буква, то она записывается в результирующий массив.
* Возвращение результирующего массива.

Нахождение дерева:

* Создаётся итератор для текста и создаётся указатель на результирующее дерево;
* Считается количество букв в переданной строке и заводится счётчик;
* Пока в итераторе присутствуют элементы:
* Если встречается буква, которая совпадает с первой буквой переданной строки, то запоминается этот элемент дерева;
* Если буква не совпадает с какой-либо буквой из переданной строки, то все обнуляется, а в стек итератора вновь заносится данный элемент дерева;
* Если количество совпавших букв совпадает с длинной переданной строки, то цикл заканчивается и метод возвращает результирующий элемент дерева.

# 5.Эксперименты

## 5.1.Время выполнения

Рассмотрим время выполнения некоторых методов теоретической сложностью O(n) (время приведено в наносекундах): вставка в список в произвольную позицию (рис. 2) и удаление произвольного элемента списка (для максимально наглядного представления вставка и удаление будут производиться в конце списка)

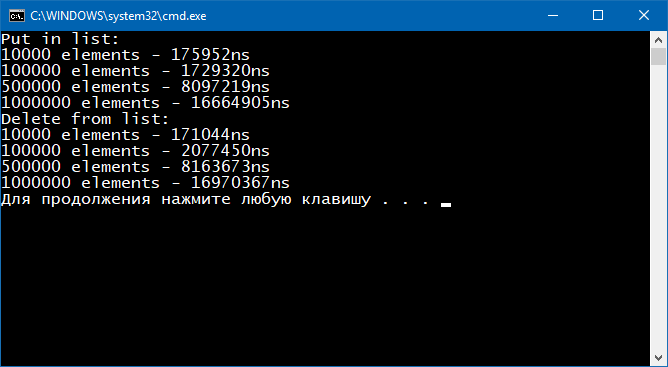
Составим диаграмму с временем работы:

рис. 2 (время работы вставки и удаления элемента из списка)

С учетом погрешности измерения, можно сказать, что теоретическая сложность алгоритмов совпадает с практическими измерениями.

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# 6.Заключение

В заключение можно сказать, что все поставленные цели и задачи были выполнены, то есть у нас получилось создать достаточно функциональный класс для работы с текстом, который по своему принципу работы схож с более профессиональными текстовыми редакторами. Все методы данных классов успешно прошли проверку с помощью Фреймворка Google Test, а значит в правильности их работы можно не сомневаться.

# 7.Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.