

Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
Высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

СОЗДАНИЕ И ОБРАБОТКА ТЕКСТА

Отчет по лабораторной работе

Выполнил:

студент ИИТММ гр. 381903-3

Алилуев А.О._____

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г._____.

Содержание

1.Введение	3
2.Постановка целей и задач	4
3.Руководство пользователя	5
4.Руководство программиста	7
4.1.Описание структуры программы	7
4.2.Описание структур данных	8
4.3.Описание алгоритмов	11
5.Эксперименты	13
5.1.Время выполнения	13
6.Заключение	15
7.Литература	16

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 быстрый 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 текста 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):

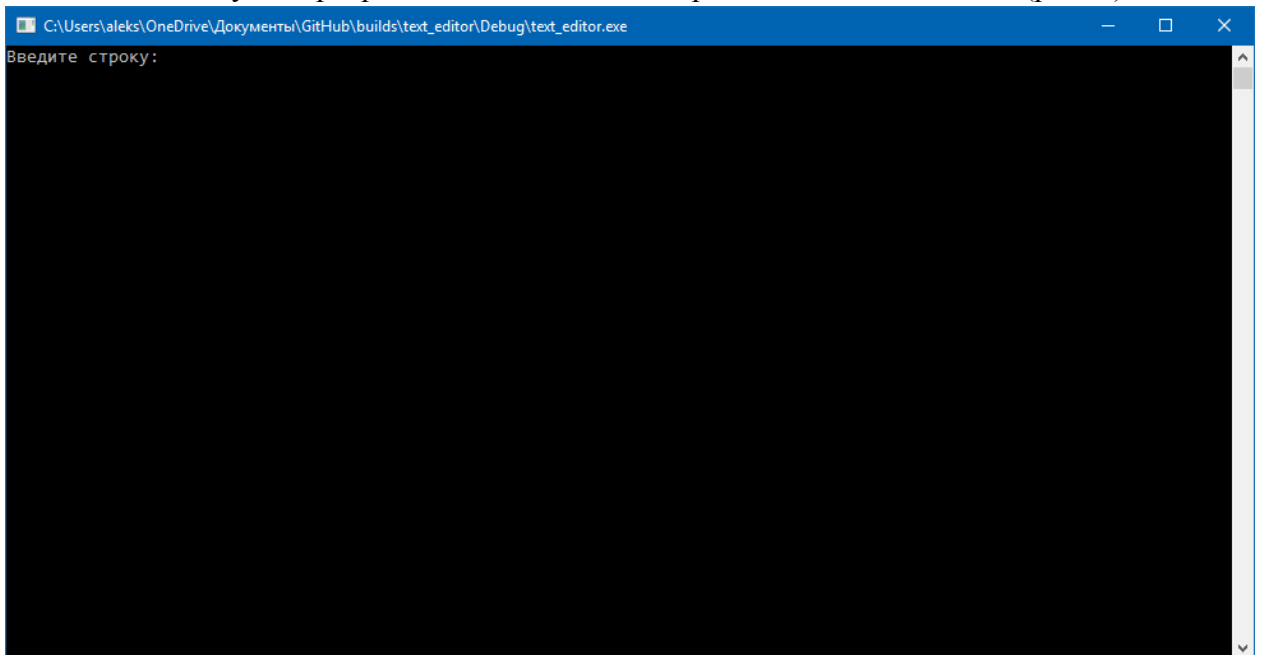


рис. 1 (вывод программы тестирования списка для пользователя)

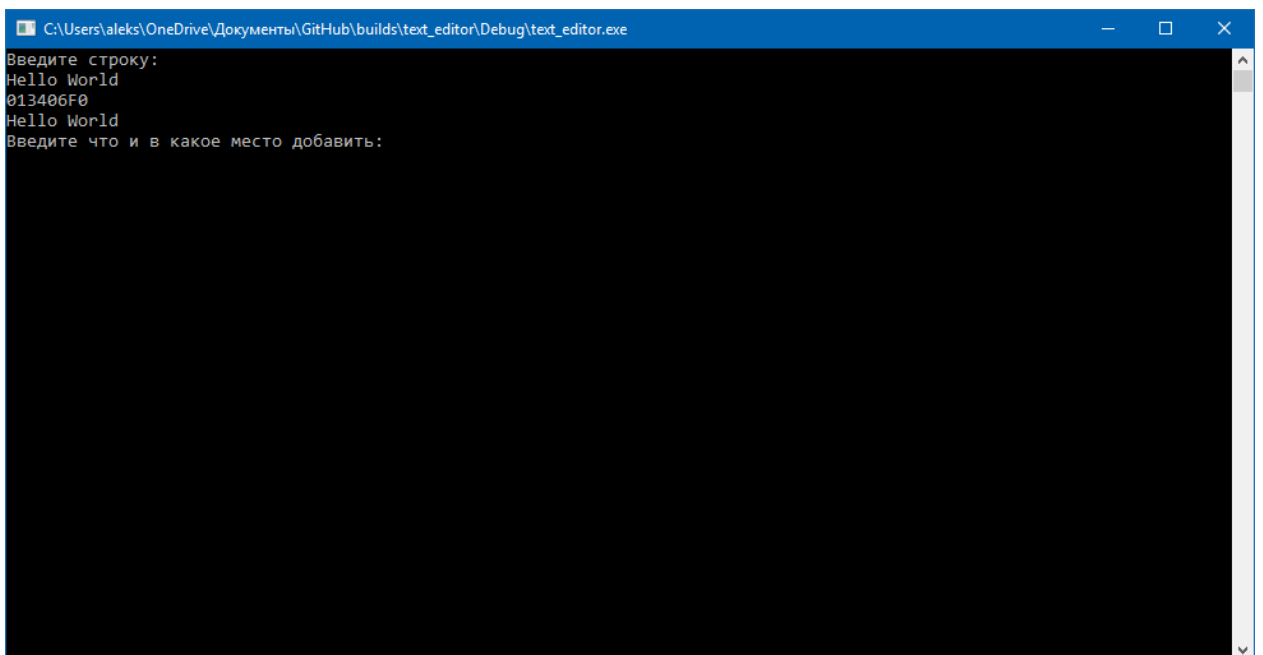
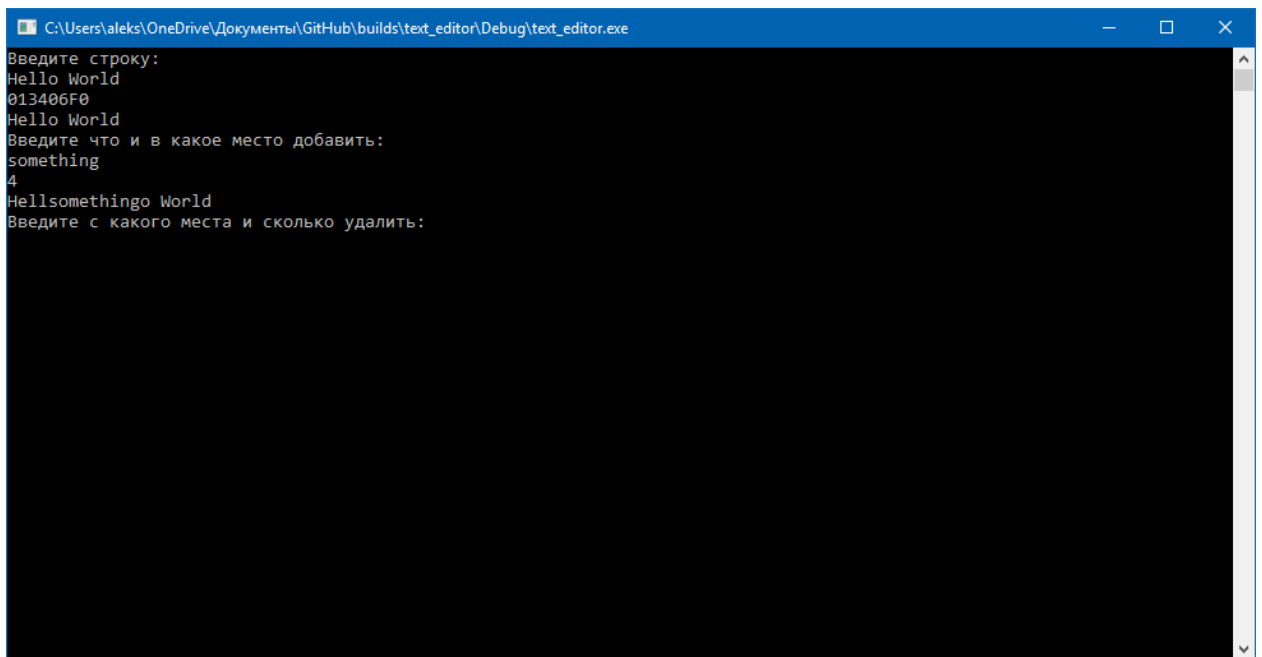


рис. 2 (тестирование создания и вывода структуры текст)

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



```
C:\Users\aleks\OneDrive\Документы\GitHub\builds\text_editor\Debug\text_editor.exe
Введите строку:
Hello World
013406F0
Hello World
Введите что и в какое место добавить:
something
4
Hellsomethingo World
Введите с какого места и сколько удалить:
```

рис. 3 (тестирование вставки текста в структуру)

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



```
Консоль отладки Microsoft Visual Studio
Введите строку:
Hello World
013406F0
Hello World
Введите что и в какое место добавить:
something
4
Hellsomethingo World
Введите с какого места и сколько удалить:
3
4
Helethingo World
C:\Users\aleks\OneDrive\Документы\GitHub\builds\text_editor\Debug\text_editor.exe (процесс 15676) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

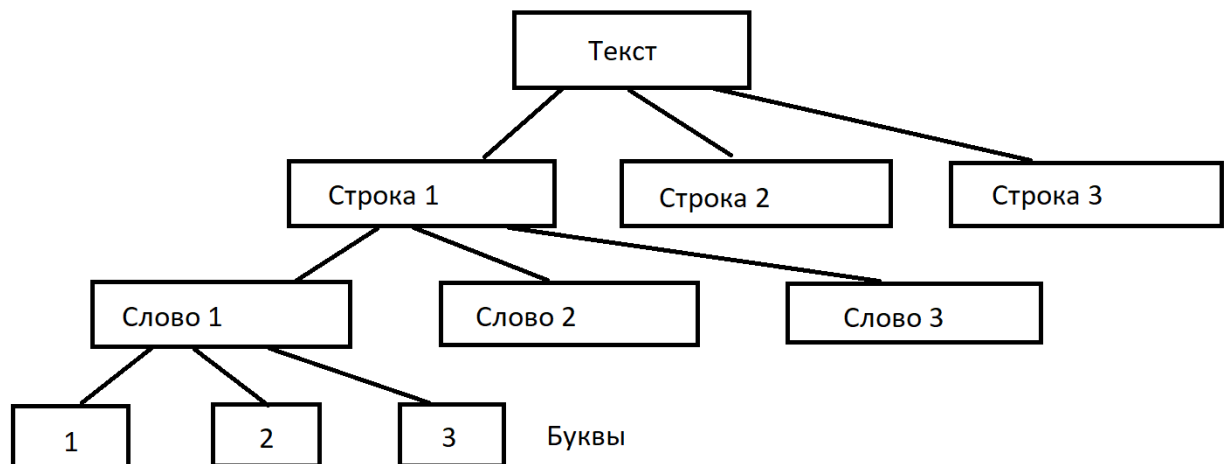
рис. 4 (тестирование удаления текста)

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 тестовый 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:
<code>TTree::TTree(const int _level)</code>	Конструктор с параметром для класса TTree, который принимает уровень для образования элемента дерева.
<code>TTree::TTree(const char* word)</code>	Конструктор дерева, который принимает массив символов, и от них формирует мини дерево – слово.
<code>TTree::TTree(const char _letter)</code>	Конструктор дерева от буквы.
<code>TTree::TTree(const TTree& tree)</code>	Конструктор копирования дерева.
<code>TTree::~~TTree()</code>	Деструктор дерева.
<code>TTree& TTree::operator=(const TTree& tree)</code>	Перегрузка оператора равно для дерева.
<code>TTree& TTree::operator+=(const TTree& tree)</code>	Перегрузка оператора «увеличения на», когда прибавляется другое дерево.
<code>TTree& TTree::operator+=(const char _letter)</code>	Перегрузка оператора «увеличения на», когда прибавляется буква.
<code>TTree& TTree::operator+=(const char* word)</code>	Перегрузка оператора «увеличения на», когда прибавляется слово.
<code>void* TTree::operator new (const unsigned int size)</code>	Перегрузка оператора выделения памяти.
<code>void TTree::operator delete(void* tree)</code>	Перегрузка оператора освобождения памяти.
<code>ostream& operator<<(ostream& o, TTree& tree)</code>	Перегрузка оператора вывода для дерева.
<code>void TTree::Output()</code>	Функция, которая позволяет вывести текст на консоль.
<code>void TTree::SetSameLevel(TTree* same_level)</code>	Метод, который позволяет задать новый same level для текущего элемента дерева.
<code>void TTree::SetNextLevel(TTree* _next_level)</code>	Метод, который позволяет задать новый next level для текущего элемента дерева.
<code>void TTree::SetLetter(const char _letter)</code>	Метод, который позволяет задать новую букву для текущего элемента дерева.
<code>void TTree::SetLevel(const int _level)</code>	Метод, который позволяет задать новый уровень для текущего элемента дерева.
<code>void TTree::SetTreeSize(const int size)</code>	Статический метод, который позволяет задать максимальное количество элементов в дереве.
<code>TTree* TTree::GetSameLevel()</code>	Получение same level текущего элемента.

<code>TTree* TTree::GetNextLevel()</code>	Получение next level текущего элемента.
<code>char TTree::GetLetter()</code>	Получение буквы текущего элемента.
<code>int TTree::GetLevel()</code>	Получение уровня текущего элемента.
<code>int TTree::GetTreeSize()</code>	Получение текущего количества свободных элементов дерева.
<code>int TTree::GetTreeBusySize()</code>	Получение текущего количества занятых элементов дерева.
<code>char* TTree::ToString()</code>	Преобразование текущего дерева в массив символов.
<code>TTree* TTree::Clone()</code>	Создание копии текущего дерева.
<code>void TTree::Initialization(const int size)</code>	Выделение памяти для всего дерева.
<code>void TTree::GarbageCollector()</code>	Сбор «ненужных» элементов дерева, для их последующего использования.
<code>void TTree::ClearMemory(void)</code>	Очистка всей выделенной памяти.
<code>TTreeIterator::TTreeIterator(TTree* _root)</code>	Конструктор для класса итератор, который принимает корень дерева.
<code>TTree* TTreeIterator::GoNext()</code>	Метод перехода к следующему элементу дерева.
<code>void TTreeIterator::Reset()</code>	Возврат всех полей итератора в исходное состояние.
<code>bool TTreeIterator::IsEnd()</code>	Проверка на окончание дерева в итераторе.
<code>void TTreeIterator::PutInStack(TTree* tree)</code>	Метод принудительного пополнения стека итератора элементом.
<code>TTree* TTreeIterator::operator()()</code>	Перегрузка оператора круглые скобки для итератора, который возвращает текущий элемент итератора.
<code>TText::TText(const char* string)</code>	Конструктор с параметром для класса тест, который принимает строку из символов.
<code>TText::TText(TText& text)</code>	Конструктор копирования текста.
<code>TText::TText(TTree& tree)</code>	Конструктор преобразования типа (из дерева в текст).
<code>TTree* TText::GetRoot()</code>	Получение корня текста.
<code>void TText::Insert(const int pos, const char* string)</code>	Метод вставки строки в текст по заданной позиции.
<code>void TText::Insert(TTree* start, TTree* string)</code>	Метод ставки строки в текст по заданному элементу дерева.
<code>int TText::Find(const char* string)</code>	Метод нахождения позиции для строки в тексте.
<code>TTree* TText::FindTree(const char* string)</code>	Метод нахождения элемента дерева по строке текста.
<code>char* TText::Copy(const int start, const int len)</code>	Метод копирования строки текста по ее позиции и длине.
<code>TTree* TText::Copy(TTree* start, const int len)</code>	Метод копирования элементов дерева, по указателю на элемент дерева и количеству букв.
<code>void TText::Delete(const int start_del, const int lenght)</code>	Метод удаления элементов дерева по начальной позиции и длине.
<code>void TText::Delete(TTree* start_del, const int lenght)</code>	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 длиной 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 миллисекундах): вставка текст строки и удаление произвольного элемента текста (для максимально наглядного представления вставка и удаление будут производиться в конце дерева):

Составим диаграмму с временем работы:



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.