МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Самарский национальный исследовательский   
университет имени академика С.П. Королева»

(Самарский университет)

|  |  |
| --- | --- |
| Институт | информатики, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

КУРСОВОЙ ПРОЕКТ ПО ДИСЦИПЛИНЕ

«Технологии и методы программирования»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

«Разработка текстового редактора»

|  |  |
| --- | --- |
| Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | П.О. Карпачева |
|  |  |
| Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(подпись)* | Е.В. Мясников |
|  |  |

МИНИСТЕРСТВА НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования «Самарский национальный исследовательский   
университет имени академика С.П. Королева»

(Самарский университет)

|  |  |
| --- | --- |
| Институт | информатики, математики и электроники |
| Факультет | информатики |
| Кафедра | геоинформатики и информационной безопасности |

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

Студенту ***Карпачевой Полине Олеговне*** группы 6312

Тема проекта: ***«Разработка текстового редактора»***

|  |  |  |
| --- | --- | --- |
| Планируемые результаты освоения образовательной программы (компетенции) | Планируемые результаты практики | Содержание задания |
| ОПК-3 ‑ способность применять языки, системы и инструментальные средства программирования в профессиональной деятельности | Знать: основные виды программных средств, технологию разработки алгоритмов и программ и методы их отладки, основы объектно-ориентированного подхода к программированию  Уметь: работать с современными системами программирования, самостоятельно осваивать новые программные средства  Владеть: языками процедурного и объектно-ориентированного программирования, навыками разработки и отладки программ. | 1. Изучение существующих методов / алгоритмов.  2. Разработка метода / алгоритма.  3. Изучение существующих библиотек/программных средств.  4. Программная реализация метода/алгоритма.  5. Разработка программы/программного обеспечения.  6. Отладка и тестирование разработанной программы/программного обеспечения.  7. Проведение экспериментов и анализ результатов. |

Дата выдачи задания 5 сентября 2023 г.

Срок представления на кафедру отчета о практике 28 декабря 2023 г.

Руководитель курсового проекта

доцент каф. ГИиИБ, к.т.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Мясников

*(подпись)*

Задание принял к исполнению

студент группы № \_6312-100503D\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.О. Карпачева

*(подпись)*

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc154079110)

[**ГЛАВА 1. МЕТОДЫ И АЛГОРИТМЫ** 5](#_Toc154079111)

[**1.1** **Анализ методов** 5](#_Toc154079112)

[**1.1.1** **Python** 5](#_Toc154079113)

[**1.1.2 Qt** 5](#_Toc154079114)

[**1.2 Описание алгоритмов** 7](#_Toc154079115)

[**1.2.1 Структуры данных** 7](#_Toc154079116)

[**1.2.2 Строки** 7](#_Toc154079118)

[**1.2.3 Стропы** 9](#_Toc154079119)

[**1.3 Разработка алгоритма** 12](#_Toc154079120)

[**1.4 Вывод по первой главе** 13](#_Toc154079121)

[**ГЛАВА 2. ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ** 14](#_Toc154079122)

[**2.1 Выбор средств разработки и системных программных средств** 14](#_Toc154079123)

[**2.1.1 Язык программирования Python** 14](#_Toc154079124)

[**2.1.2 Графический интерфейс (GUI) PyQt** 14](#_Toc154079125)

[**2.1.3 Среда программирования для Python** 15](#_Toc154079126)

[**2.2 Структура программы** 16](#_Toc154079127)

[**2.3 Описание основных функций и процедур** 19](#_Toc154079128)

[**2.4 Описание интерфейса пользователя** 22](#_Toc154079129)

[**2.5 Контрольный пример и результаты тестирования** 26](#_Toc154079130)

[**2.5.1 Форматирование текста** 26](#_Toc154079131)

[**2.5.2 Сохранение и загрузка текста** 33](#_Toc154079132)

[**2.6 Вывод по второй главе** 34](#_Toc154079133)

[**ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ** 35](#_Toc154079134)

[**3.1** **Данные для экспериментов** 35](#_Toc154079135)

[**3.1.1 Описание - стропа** 35](#_Toc154079136)

[**3.1.2 Тип информаций - стропа** 35](#_Toc154079137)

[**3.2 Вывод по третьей главе** 39](#_Toc154079138)

[**Заключение** 40](#_Toc154079139)

[**Список литературы** 41](#_Toc154079140)

[**ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ** 42](#_Toc154079141)

**ВВЕДЕНИЕ**

Цель работы – создание удобного инструмента для редактирования и форматирования текстовых документов, который будет обладать широким набором функций и возможностей, удовлетворяющих потребности пользователей различных категорий.

В соответствии с поставленной целью были сформулированы и решены следующие задачи:

1. изучение существующих методов;
2. изучение существующих библиотек;
3. разработка алгоритма;
4. разработка программы;
5. отладка и тестирование разработанной программы;
6. проведение экспериментальных исследований.

WYSIWYG (аббревиатура: что видишь, то и получаешь) редакторы широко используются в веб-приложениях в качестве встроенного текстового редактора. В этой работе мы разработаем визуальный редактор, необходимый для дальнейшей работы.

В мире, где ответственность за создание контента для сети несут непрофессионалы, использующие веб-языки и стандарты, крайне важно наличие инструментов, поддерживающих создание доступного контента. В этом документе предлагается сделать концепции доступности Интернета более понятными для этих пользователей за счет включения рекомендаций по доступности WCAG 2.0 в редакторы HTML WYSIWYG, которые они используют. Для этого мы спроектировали и создали прототип такого редактора и провели предварительные тесты удобства использования с целевыми пользователями. Результаты показали, что предупреждения о доступности легко понять и применить, но их сложно воспринимать.

# **ГЛАВА 1. МЕТОДЫ И АЛГОРИТМЫ**

## **Анализ методов**

На рынке существуют множество языков программирования: C++, C, C#, Java, Python, Ruby.

### **Python**

Python — это язык компьютерного программирования, который часто используется для создания веб-сайтов и программного обеспечения, автоматизации задач и проведения анализа данных. Python — это язык общего назначения, то есть его можно использовать для создания множества различных программ, и он не специализирован для решения каких-либо конкретных задач. Эта универсальность, а также удобство для начинающих сделали его сегодня одним из наиболее используемых языков программирования.

Опрос разработчиков Stack Overflow в 2022 году показал, что Python является четвертым по популярности языком программирования, при этом респонденты заявили, что используют Python почти 50 процентов времени в своей работе по разработке. Результаты опроса также показали, что Python связан с Rust как с наиболее востребованной технологией: 18% разработчиков, которые его не используют, уже заявили, что они заинтересованы в изучении Python.

### **1.1.2 Qt**

Qt — это кроссплатформенная среда разработки приложений, широко используемая для разработки настольных, мобильных и встраиваемых систем. Qt предоставляет мощную, простую в использовании и гибкую библиотеку классов C++ для создания графических интерфейсов и других типов приложений. Qt имеет широкий спектр встроенных виджетов, включая кнопки, метки, списки и многое другое.

Qt также обеспечивает поддержку нескольких платформ: от Windows до macOS, Linux, iOS и Android. Qt также поддерживает библиотеки для различных языков, включая C++, Python и QML, с моделью, управляемой событиями, для создания быстро реагирующих и интерактивных приложений.

Пакет qt — это оболочка Go для библиотеки Qt, предоставляющая основу для создания кроссплатформенных приложений. Пакет qt предлагает множество возможностей и функций, включая поддержку компонентов графического интерфейса, таких как кнопки, метки и текстовые поля.

Пакет qt включает в себя библиотеку Qt (на основе C++), и вам потребуется установить пакет и компилятор C++ для сборки и запуска вашего приложения. Перейдите на страницу установки библиотеки Qt, чтобы установить библиотеку Qt и начать работу.

## **1.2 Описание алгоритмов**

### **1.2.1 Структуры данных**

Языки программирования обычно предоставляют тип «строка» или «текст», позволяющий манипулировать данными. последовательности символов. Этот тип обычно имеет решающее значение, поскольку он обычно упоминается в большинстве интерфейсов между компонентами системы. Мы утверждаем, что традиционные реализации строки и часто поддерживаемые функции не очень подходят для такого общего использования. Их следует ограничить приложениями с особыми и необычными требованиями к производительности. Мы представляем «веревки» или «тяжелые» струны как альтернативу, которая, по нашему опыту, приводит к системы, которые более надежны как по функциональности, так и по производительности.

Веревки использовались в среде Кедра почти с момента его создания, но, по-видимому, это не быть ни широко известным, ни обсуждаемым в литературе. Алгоритмы постепенно изысканный. Недавно мы также создали второй аналогичный, но несколько более легкий язык C. реализацию, которая включена в наш общедоступный дистрибутив сборщика мусора. Мы опишите алгоритмы, используемые в обоих случаях, и приведите некоторые измерения производительности для версии C.

### **1.2.2 Строки**

Языки программирования, такие как C и традиционный Паскаль, предоставляют встроенное понятие строк как массивы символов фиксированной длины. Сам язык обеспечивает примитивы массива для доступа к таким строкам, а также часто набор библиотек процедуры для операций более высокого уровня, таких как конкатенация строк. По существу, строка ограничена представлением строк как непрерывных массивов символов. с дополнительным пространством по длине, комнатой расширения или без него и т. д.

Нет сомнений в том, что такие структуры данных иногда уместны, и необходимо предоставить структуру данных «массив символов». С другой стороны, поскольку тип символьной строки будет повсеместно использоваться для связи между модулей большой системы, нам нужны следующие характеристики:

1. Неизменяемые строки, то есть строки, которые нельзя изменить на месте, должны быть хорошо поддерживается. Процедура должна иметь возможность работать со строкой, которую она передала. без опасности случайного изменения структур данных вызывающей стороны. Этот становится особенно важным при наличии параллелизма, когда на месте обновления строк часто должны быть должным образом синхронизированы. (Хотя они не являются нормой, неизменяемые строки были предоставлены SNOBOL и БЕЙСИК с середины 1960-х годов. Таким образом, эта идея вряд ли нова.)

2. Часто встречающиеся операции со строками должны быть эффективными. В частности (неразрушающее) объединение строк и неразрушающих операций подстроки. Действия должны быть быстрыми и не должны требовать чрезмерного количества места.

3. Обычные строковые операции должны масштабироваться до длинных строк. Не должно быть никаких практическое ограничение длины струн. Производительность должна оставаться приемлемой для длинных струн. (Все мы видели симптомы нарушения этого требование. Например, редактор vi в большинстве систем UNIX™ непригоден для использования. во многих текстовых файлах из-за ограничения длины строки. Непроверенный лимит ввода в Fingerd поддержал интернет-червя Морриса.2 Шестимесячный ребенок случайно набор текста на рабочей станции обычно приводил к сбою некоторых старых ядер UNIX из-за к ограничению размера буфера и т. д.)

4. Должно быть как можно проще рассматривать какое-либо другое представление «последовательности». символа» (например, файла) в виде строки. Функции над строками должны быть максимизированы. практически многоразового использования.

### **1.2.3 Стропы**

Чтобы конкатенация строк была эффективной как во времени, так и в пространстве, необходимо должна быть возможность, чтобы результат разделял большую часть структуры данных со своими аргументами. Это подразумевает, что полностью ручное управление хранилищем (например, на основе явного malloc/free) непрактично. (Можно утверждать, что это верно даже для обычных строк. представления. Ручное управление хранилищем обычно приводит к появлению множества ненужных строк. копирование.) Хотя реализация с явным подсчетом ссылок может быть построена, мы возьмет на себя автоматическую сборку мусора. Поскольку конкатенация не может копировать свои аргументы, естественной альтернативой является представляют такую строку как упорядоченное дерево, где каждый внутренний узел представляет собой конкатенация его дочерних элементов, а листья состоят из плоских строк, обычно представленных как непрерывные массивы символов. Таким образом, строка, представленная деревом представляет собой объединение его листьев в порядке слева направо, как показано на рисунке 1.1.

Мы ссылаемся на строки символов, представленные в виде дерева узлов конкатенации, как веревки. (Это немного небрежно. Веревка может содержать общие поддеревья, поэтому на самом деле она ориентированный ациклический граф, в котором исходящие ребра каждой вершины упорядочены. Мы будем продолжайте быть небрежным).

Веревки можно рассматривать как поиск деревьев, индексированные положения. Если твоя вершина содержит длину строки, представленной поддеревом, а затем минимальную модификацию. Катионы алгоритмов поиска дерева дают следующие операции над веревками:

1. Получите i-й символ. Простой поиск в дереве поиска. Вместо того, чтобы исследовать поддерево, содержащее правильный ключ, мы исследуем дерево, содержащее правильный ключ. положение, определенное полями длины.

2. Соедините две веревки. Объединение деревьев поиска, как определено в ссылке 3.

3. Подстрока. Две операции разделения дерева поиска, как определено в ссылке 3.

4. Переберите каждый символ. Обход дерева слева направо

Первые три из вышеперечисленных операций можно выполнить за логарифмическое время. длину аргумента, используя, например, в деревьях. Обратите внимание, что поскольку строки неизменяемы, любые узлы, которые будут изменены в стандарте версии алгоритма, а также их предки копируются. Только логарифмически необходимо скопировать многие узлы.

Последнее может быть выполнено за линейное время практически для любого варианта дерева поиска. Таким образом, как операции конкатенации, так и операции с подстроками (кроме очень коротких подстрок) асимптотически быстрее, чем для обычных плоских струн. Последние экспонаты примерно тот же спектакль. Первый несколько медленнее, но обычно нечастый.

На практике мы модифицируем это двумя способами. Конкатенация часто является достаточно важная операция, которая должна выполняться за единицу, а не за логарифмическое время. Длинный вывод строки обычно создаются путем объединения коротких строк. Например, компиляторы генерация вывода на ассемблере может осуществляться путем объединения строк в памяти. Следовательно, двоичная конкатенация обычно просто добавляет корневой узел и не выполняет повторную балансировку. дерево. Операция ребалансировки либо выполняется выборочно, либо вызывается явно. Фактически это меняет скорость операций подстроки и выборки на лучшая производительность конкатенации. Итерация по веревке практически не затрагивается.

(Теоретически можно гарантировать, что первые три операции выполняются в амортизированное логарифмическое время, итерационная операция выполняется за линейное время, а отдельные конкатенации выполняются за постоянное время с использованием «дерева расширения» представление, в котором каждый узел представляет собой объединение (возможно, пустой) веревки, (непустой) плоской струны и другой веревки. Это кажется непрактичным в нашей ситуации, поскольку операция выборки требует больших объемов выделения и модификация структуры данных. Более того, в многопоточной среде выборка операция, похоже, требует блокировки).

Полезно ввести дополнительные виды узлов дерева. Как минимум, мы позволяем листовой узел второго типа, содержащий как минимум длину и определяемую пользователем функцию для вычисления i-го символа в строке. Это позволяет использовать другие представления последовательности символов (например, файлы), которые следует рассматривать как веревки без копирования. Это может в дальнейшем будет полезно ввести узлы подстроки, чтобы длинные подстроки плоских веревок не требуют копирования.

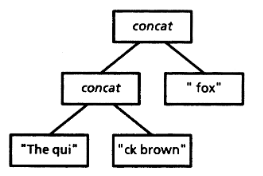


Рисунок 1.1. Веревочное изображение ‘The quick brown fox’

## **1.3** **Разработка алгоритма**

1. Определение требований пользователей. Для начала необходимо провести анализ потребностей пользователей и определить основные функции и возможности, которые должен обладать текстовый редактор.

2. Проектирование интерфейса. На основе выявленных требований пользователей необходимо разработать удобный и интуитивно понятный интерфейс текстового редактора, который будет обеспечивать удобство использования всех функций.

3. Разработка основных функций. Создание основных функций текстового редактора, таких как вставка, удаление, копирование текста, форматирование шрифта, выравнивание текста, нумерация и маркировка списков и другие.

4. Добавление дополнительных возможностей. Разработка дополнительных возможностей, таких как выбор цвета шрифта, заливки фона и т.д.

5. Тестирование и отладка. После разработки всех функций необходимо провести тестирование редактора.

## **1.4 Вывод по первой главе**

Для разработки текстового редактора я выбрала язык программирования Python и библиотеку Qt по нескольким причинам.

Во-первых, Python – это высокоуровневый язык программирования, который обладает простым и понятным синтаксисом, что делает его очень удобным для разработки. Python также предлагает большое количество библиотек и фреймворков, что облегчает процесс создания приложений.

Во-вторых, библиотека Qt предоставляет широкие возможности для создания графического интерфейса приложения. Она поддерживает множество виджетов, стилей оформления и инструментов для работы с графикой, что позволяет создавать красивые и функциональные пользовательские интерфейсы. Кроме того, Qt имеет кроссплатформенную поддержку, что означает, что разработанное приложение будет работать на различных операционных системах без изменений в исходном коде. Это важно для обеспечения доступности приложения для максимального числа пользователей. Таким образом, выбор Python и библиотеки Qt для разработки текстового редактора обусловлен их удобством использования, широкими возможностями для создания графического интерфейса и кроссплатформенной поддержкой.

В ходе первой главы была рассмотрена основа визуального редактора. Также в ходе первой главы были рассмотрены инструментарии разработки WYSIWYG-приложения.

# **ГЛАВА 2. ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

## **2.1 Выбор средств разработки и системных программных средств**

### **2.1.1 Язык программирования Python**

Среди списка языков программирования, которые были перечислены выше, был выбран язык программирования Python. Данный язык был выбран по достаточно несколько причин:

* лаконичность, на нем можно легко и быстро написать любую программу,
* требование к заданиям,
* доступность на многих платформах (Windows, Mac OS, Linux),
* не ресурсо требовательная,
* высокая поддержка со стороны сообщества, огромное количество библиотек.

### **2.1.2 Графический интерфейс (GUI) PyQt**

Среди списка можно выбрать среду PyQt5. Данный графический интерфейс имеет ряд преимуществ по сравнению с другими интерфейсами:

* открытая лицензия,
* широкая поддержка сообществом и компанией Qt,
* популярность среды использования,
* доступность на многих языках программирования,
* легкий в использовании.

Но среди данных преимуществ есть еще одно немаловажное преимущество. Это широкая поддержка визуализации текста с его обработкой. Объекты класса QTextEdit “из под коробки” уже включает возможность форматирование текста. В ходе данной работы будем использовать именно его инструментами.

### **2.1.3 Среда программирования для Python**

Для работы на Python, можно пользоваться и при помощи простого текстового редактора. Однако в нем есть некоторые недостатки:

* Времени на создание данного увеличивается,
* Можно случайно ошибиться в ходе написания программы,
* Отсутствие подсказок.

Поэтому мы будем писать программу при использовании среды программирования. В данном случае - Visual Studio Code. Данная среда является популярной и поддерживается компанией JetBrains.

## **2.2 Структура программы**

Для начала разработки необходимо будет загружать Python, Visual Studio Code и PyQt5 на компьютер.

Программа будет состоять из нескольких частей: строка меню (menu bar), панель инструментов (tool bar), текст.

В строке меню будут расположены следующие действия (таблица 2.1):

Таблица 2.1 ‑ Список строки меню

|  |  |  |
| --- | --- | --- |
| Меню | Предназначение | Действие |
| Файл | Операция с файлом | Открыть |
| Новый |
| Сохранить |
| Выход |
| Правка | Правка текста | Отменить |
| Возвращать |
| Копировать |
| Вырезать |
| Вставить |
| Шрифт | Форматирование текста | Жирный |
| Курсив |
| Подчеркнутый |
| Зачеркнутый |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| Меню | Предназначение | Действие |
| Шрифт | Форматирование текста | Цвет текста |
| Цвет выделения |
| Выравнивание по левому краю |
| Выравнивание посередине |
| Выравнивание по правому краю |
| Выравнивание по ширине |

Также у этих операций в menu bar были включены возможность операции по горячим клавишам. Список можно увидеть на рисунке 2.4.

Также в нем добавлены иконки. Иконки хранятся в директории (папка) icons. Набор иконок отображен на таблице 2.2.

Таблица 2.2 ‑ Набор иконок

|  |  |
| --- | --- |
| Новый |  |
| Открыть |  |
| Сохранить |  |
| Отменить |  |
| Возвращать |  |
| Копировать |  |
| Вырезать |  |

Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Вставить |  |
| Жирный |  |
| Курсив |  |
| Подчеркнутый |  |
| Зачеркнутый |  |
| Цвет текста |  |
| Цвет выделения |  |
| Выравнивание по левому краю |  |
| Выравнивание посередине |  |
| Выравнивание по правому краю |  |
| Выравнивание по ширине |  |

## **2.3** [**Описание основных функций и процедур**](https://docs.google.com/document/d/1FH83jtT3N2cOL0k5kQRgojxAvWK-eW5N/edit#heading=h.41mghml)

Для создания строки меню используем код QMenuBar(). Для создания меню - QMenu(). Для Создания действий в меню - QAction() . Для назначения горячей клавиши - setShortcut(“Ctrl+кнопка”). Для действий при нажатий на кнопку - triggered.connect(). Для более простых действий (если используется 1 строка) - выполнение происходит непосредственно в параметре. При более сложных (более 1 строки) - назначается отдельная функция, а в параметре будет указана соответствующая функция. Список действий будет перечислены в таблице 2.4.

Таблица 2.4 ‑ Действие команд

|  |  |
| --- | --- |
| Новый | setText("") |
| Открыть | Функция LoadTextFile.  f = QFile("file.html")  f.open(QFile.ReadOnly | QFile.Text)  istream = QTextStream(f)  text.setHtml(istream.readAll())  f.close() |
| Сохранить | Функция SaveTextFile.  document = text.document()  file = QFile("file.html")  if file.open(QFile.WriteOnly):  OutputText = document.toHtml()  file.write(OutputText.encod  e('utf-8'))  file.close()  return True  else:  return False |

Продолжение таблицы 2.4

|  |  |
| --- | --- |
| Копировать | copy() |
| Выход | quit() |
| Отменить | undo() |
| Возвращать | redo() |
| Вырезать | cut() |
| Вставить | paste() |
| Жирный | Функция bold().  if text.fontWeight != QFont.Bold:  text.setFontWeight(QFont.Bold)  return  text.setFontWeight(QFont.Normal) |
| Курсив | Функция italic().  state = text.fontItalic()  text.setFontItalic(not (state)) |
| Подчеркнутый | Функция underline().  state = text.fontUnderline()  text.setFontUnderline(not (state)) |
| Зачеркнутый | Функция strike().  f = text.font()  if (f.strikeOut()):  f.setStrikeOut(False)  else:  f.setStrikeOut(True)  text.setFont(f) |

Продолжение таблицы 2.4

|  |  |
| --- | --- |
| Цвет выделения | Функция highlight().  newColorDialog = QColorDialog()  newColor = newColorDialog.getColor()  text.setTextBackgroundColor(QColor(newColor.name())) |
| Цвет текста | Функция color().  newColorDialog = QColorDialog()  newColor = newColorDialog.getColor()  text.setTextColor(QColor(newColor.name())) |
| Выравнивание по левому краю | lambda : text.setAlignment(Qt.AlignLeft) |
| Выравнивание посередине | lambda : text.setAlignment(Qt.AlignCenter) |
| Выравнивание по правому краю | lambda : text.setAlignment(Qt.AlignRight) |
| Выравнивание по ширине | lambda : text.setAlignment(Qt.AlignJustify) |

## [**2.4 Описание интерфейса пользователя**](https://docs.google.com/document/d/1FH83jtT3N2cOL0k5kQRgojxAvWK-eW5N/edit#heading=h.2grqrue)

На рисунке 1.1 продемонстрирована строка меню.



Рисунок 1.1 – Строка меню

Теперь рассмотрим второй объект программы - tool box. В них расположены объекты класса QAction, QLabel (для добавления надписи), QSpinBox (поле для ввода значение цифры, нужен для установления размера шрифта), QComboBox (список выпадающих строк, нужен для установления) списка шрифта. Помимо них в кнопках ставится иконки. Набор иконок совпадает с наборами иконок в строке меню. Сам панель инструментов продемонстрирован на рисунке 1.2.



Рисунок 1.2 – Панель инструментов

Внизу программы ставится тестовое поле, которое занимает всю остальную часть программы.

Вставляя все эти объекты, получаем саму программу по редактированию и форматирования текста. Сама программа продемонстрирована на рисунке 1.3.

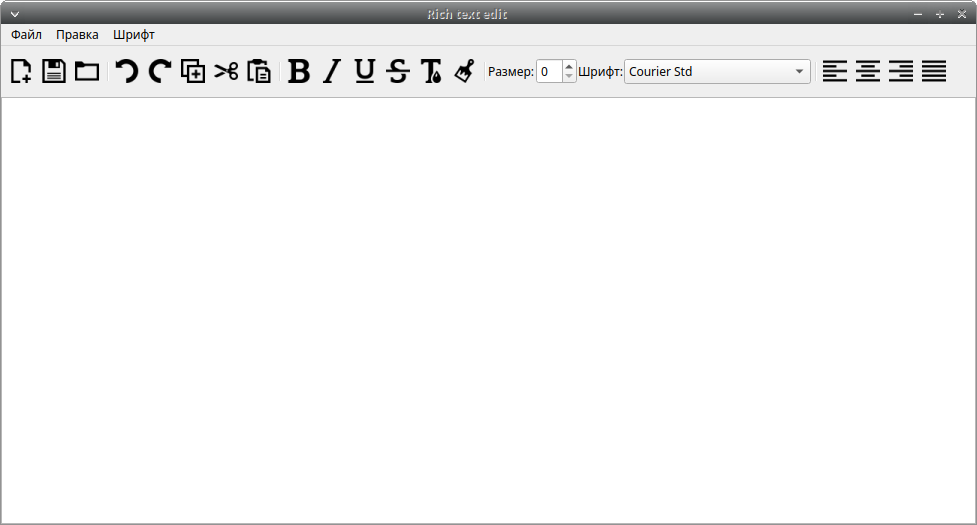


Рисунок 1.3 – Программа

*Открытие и сохранение*

Для открытия и сохранения файла программа использует перевод в HTML. Когда программа сохраняет написанный текст в файл, то программа переводит текст в формат HTML при помощи метода toHtml(). Поэтому сохраненный файл можно просматривать практически во всех Web-браузерах. Однако форматирование текста происходит на стороне языка CSS. При открытии файла все содержимое файла загружается в программу. Затем при помощи метода setHtml() вставляется в текст.

*Выбор цвета*

Для выбора цвет используется специальное диалоговое окно, предназначенное специально для выбора действия. Оно уже включено в состав Qt.

Для вызова диалогового окна необходимо класс QColorDialog(). При выборе получается строку номера цвета, подобное HTML. Для получение данной строки вызывается метод getColor(). Для установки цвета шрифта или выделения - метод QColor().

*Новый документ*

Чаще всего создание нового документа подразумевается создание пустого текста. Его можно вызывать установкой текста “”. Метод в Qt - setText(“”).

*Выбор шрифта*

Для выбора шрифта в программу встроен список шрифтов. Сейчас в программу включен возможность выбора следующих шрифтов:

* Courier Std,
* Hellentic Typewriter Regular,
* Helvetica,
* Arial,
* SansSerif,
* Helvetica,
* Times,
* Monospace

Метод установки шрифта setCurrentFont(). Для уточнение шрита - QFont().

*Выравнивание текста*

Существуют 4 типа выравнивания: по левому краю, посередине, по правому краю, по ширине. Реализация в PyQt соответственно: lambda : text.setAlignment(Qt.AlignLeft), lambda: text.setAlignment(Qt.AlignCenter), lambda: text.setAlignment(Qt.AlignRight), lambda: text.setAlignment(Qt.AlignJustify).

Таблица 2.5 ‑ Горячие клавиши действий

|  |  |
| --- | --- |
| Новый | ctrl+N |
| Открыть | ctrl+O |
| Сохранить | ctrl+S |
| Выход | ctrl+Q |
| Отменить | ctrl+Z |
| Возвращать | ctrl+R |
| Копировать | ctrl+C |
| Вырезать | ctrl+X |
| Вставить | ctrl+V |
| Жирный | ctrl+B |
| Выравнивание посередине | ctrl+E |
| Курсив | ctrl+I |
| Подчеркнутый | ctrl+U |
| Зачеркнутый | ctrl+Y |
| Цвет текста | ctrl+M |
| Цвет выделения | ctrl+H |
| Выравнивание по левому краю | ctrl+L |
| Выравнивание по правому краю | ctrl+T |
| Выравнивание по ширине | ctrl+J |

## [**2.5 Контрольный пример и результаты тестирования**](https://docs.google.com/document/d/1FH83jtT3N2cOL0k5kQRgojxAvWK-eW5N/edit#heading=h.vx1227)

### **2.5.1 Форматирование текста**

Рассмотрим список возможных форматирований. Продемонстрируем его на программе.

*Обычный текст*

Если не проводить форматированный текст, то текст будет обычным с размером 9, шрифт Arial, выравнивание по левому краю, черный текст.

*Жирный шрифт*

Жирный шрифт продемонстрирован на рисунке 3.1.

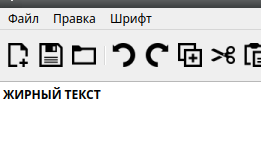


Рисунок 2.1 – Жирный текст

*Курсив*

Шрифт курсива продемонстрирован на рисунке 2.2.

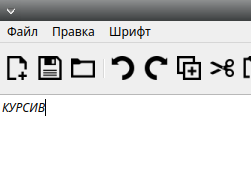


Рисунок 2.2 – Курсив

*Подчеркнутый текст*

Подчеркнутый текст продемонстрирован на рисунке 2.3.

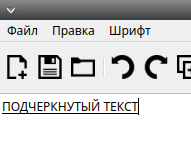


Рисунок 2.3 – Подчеркнутый текст

*Зачеркнутый текст*

Зачеркнутый текст продемонстрирован на рисунке 2.4.

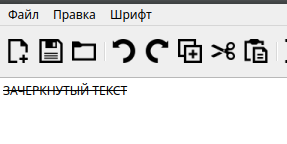


Рисунок 2.4 – Зачеркнутый текст

*Цвет текста*

Также можно изменить цвет текста. Например, синий (рисунок 2.5) или красный (рисунок 2.6).

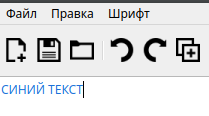


Рисунок 2.5 – Синий текст

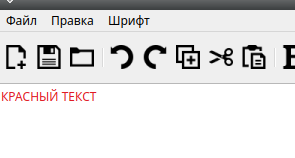


Рисунок 2.6 – Красный текст

*Выделение*

На рисунке 2.7 выделение желтого цвета. А на рисунке 2.8 выделение зеленого цвета.

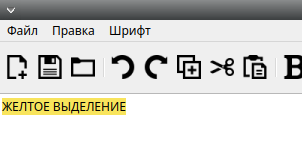


Рисунок 2.7 – Выделение желтого цвета

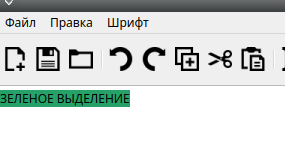


Рисунок 2.8 – Выделение зеленого цвета

*Диалоговое окно цвета*

Для выбора цвета устанавливается диалоговое окно цвета. На рисунке 2.9 продемонстрирован диалог выбора цвета. Также можно открыть пункт “Все цвета” (рисунок 2.10).

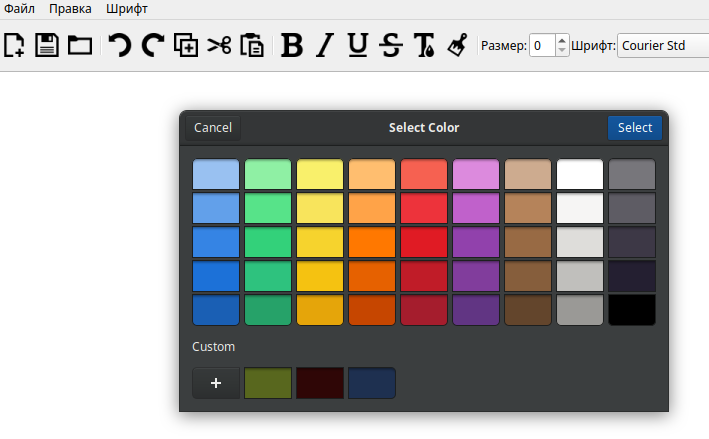


Рисунок 2.9 – Диалог выбора цвета в системе Ubuntu XFCE

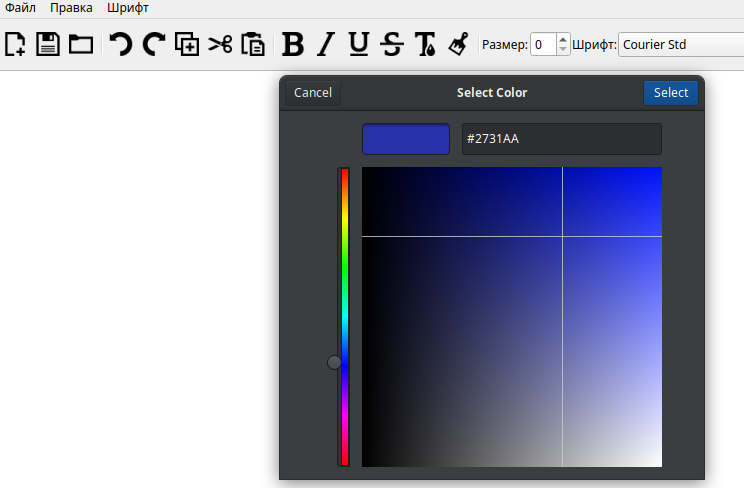


Рисунок 2.10 – Диалог выбора цвета в случае, если был открыт пункт “Все цвета” в системе Ubuntu XFCE

*Размер текста*

Также был изменен размер текста. Размер текста 21 - на рисунке 2.11. Размер текста 36 - на рисунке 2.12.

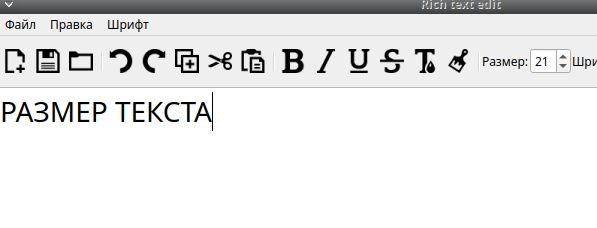


Рисунок 2.11 – Текст с размером 21

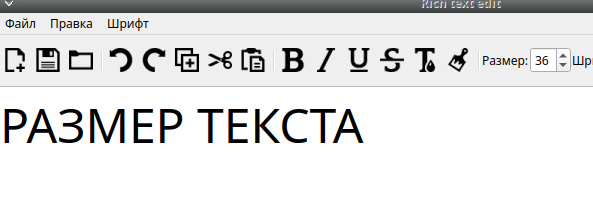


Рисунок 2.12 – Текст с размером 36

*Изменение шрифта*

Программой предусмотрено изменение шрифт текста. На рисунке 2.13 - шрифт Times. На рисунок 2.14 - шрифт Courier Std.



Рисунок 2.13 – Шрифт Times



Рисунок 2.14 – Шрифт Courier Std

*Выравнивание текста*

Есть возможность выравнивать текст по левому краю, посередине (2.15), по правому краю (2.16), по ширине (2.17).

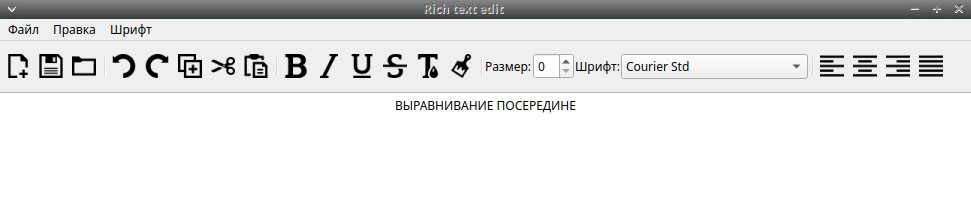


Рисунок 2.15 – Выравнивание текста посередине

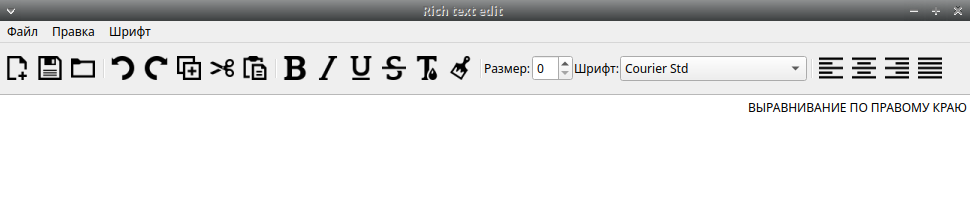


Рисунок 2.16 – Выравнивание текста по правому краю

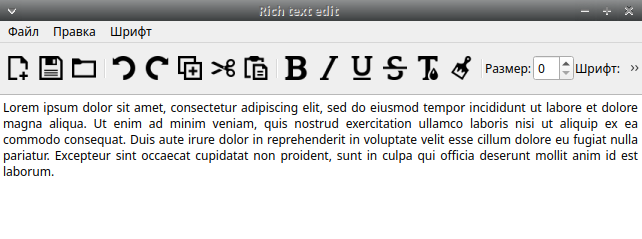


Рисунок 2.17 – Выравнивание текста по ширине

*Отмена и возврат*

Мы печатали текст в программу (рисунок 2.18). Далее мы возвратили к предыдущему варианту (рисунок 2.19). Позже мы возвращали текст обратно (2.20).

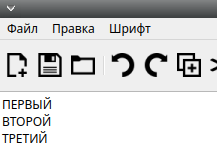


Рисунок 2.18 – Первоначальный текст

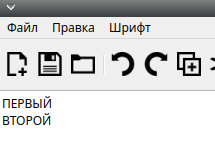


Рисунок 2.19 – Текст после кнопки “назад”

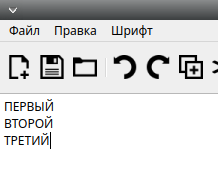


Рисунок 2.20 – Текст после возврата кнопкой “Назад”

### **2.5.2 Сохранение и загрузка текста**

Рассмотрим сохранение и загрузка файлов. Для начала заполним текст и большим числом форматирований (рисунок 2.21).

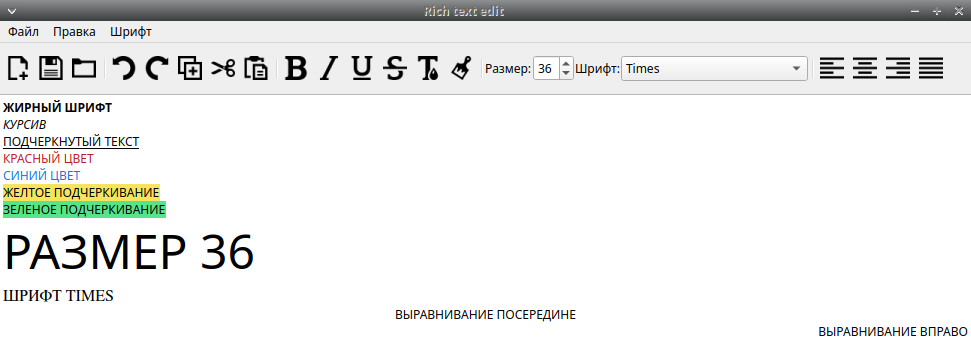


Рисунок 2.21 – Текст с большим количеством форматирований

Сохраняем данный текст в формат HTML.

## **2.6 Вывод по второй главе**

В ходе второй главы был разработан визуальный редактор. Были описаны все пути разработки.

После разработки текстового редактора с использованием Python и библиотеки Qt, я убедилась в правильности выбора языка программирования и инструментов. Python позволил мне легко и быстро создать основной функционал приложения, благодаря своему простому синтаксису и большому количеству доступных библиотек.

Библиотека Qt оказалась очень мощным инструментом для создания графического интерфейса. Я смогла легко настроить различные виджеты, стили оформления и инструменты для работы с текстом, что позволило создать удобный и привлекательный пользовательский интерфейс.

Кроссплатформенная поддержка Qt также оказалась очень важной, поскольку мое приложение будет работать на различных операционных системах без необходимости дополнительной оптимизации.

В целом, выбор Python и библиотеки Qt для разработки текстового редактора оказался очень удачным, и я уверена, что моё приложение будет удобным и доступным для широкого круга пользователей.

# [**ГЛАВА 3. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ**](https://docs.google.com/document/d/1FH83jtT3N2cOL0k5kQRgojxAvWK-eW5N/edit#heading=h.4f1mdlm)

## [**3.1 Данные для экспериментов**](https://docs.google.com/document/d/1FH83jtT3N2cOL0k5kQRgojxAvWK-eW5N/edit#heading=h.2u6wntf)

### **3.1.1 Описание - стропа**

**Строп** (канат, корд) — структура данных, которая позволяет эффективно хранить и обрабатывать длинные строки, например текст. Обеспечивает хранение длинной строки в виде дерева состоящего из небольших подстрок. Она удобна для хранения и обработки текстовых файлов и обеспечивает эффективное выполнение типичных для текстового редактора операций: вставки, удаления, обращения (случайного доступа).

Строп представляет собой двоичное дерево, где каждый лист (узел без потомков) содержит строку (подстроку текста) и длину (вес), а каждый нелистовой узел дерева содержит длину листьев своего левого дочернего поддерева. Узел с двумя детьми разбивает целую строку на две части, где первую часть хранит левое поддерево, вторую — правое, вес самого узла равен длине первой части. Для основных операций полагается, что строки не изменяются в неразрушающем случае, допуская копирование-при-записи. Листовые узлы обычно реализуются как строки постоянной длины с счётчиком ссылок, когда он обнуляется память высвобождается. Могут также использоваться продвинутые способы сборки мусора.

### **3.1.2 Тип информаций - стропа**

Рассмотрим работу структуру данных строп. Будем делать это на примере строки “СтруктураДанныхСтроп”. Нам необходимо разбить данную строку на части. В данном случае построим бинарное дерево. Готовое дерево можно увидеть на рисунке 3.22.

В узле A - общее количество символов - 20. В узлах B, C, D, E, F - значение левой части, где указаны общие количества символов. На листьях G, H, I, J, K, L – количества конечных символов.

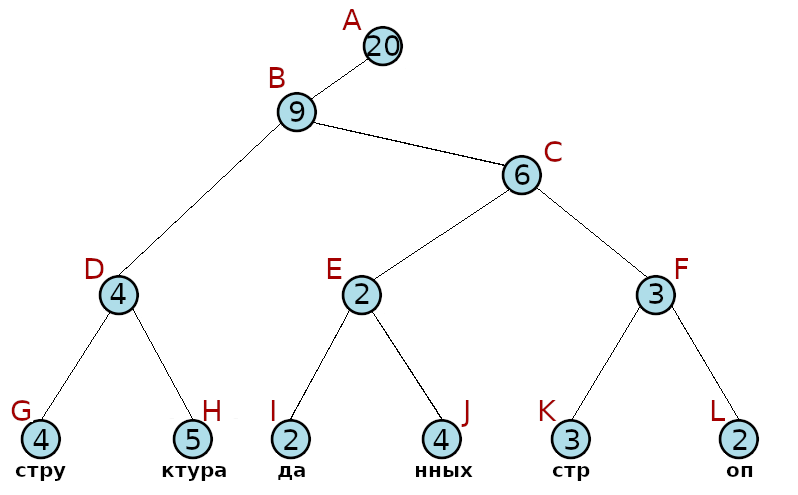


Рисунок 3.1 – Разбиение на стропу строки “структура данных строп”

Попробуем проводить операцию с подстрокой “данных”. Операция может быть любой. Это копирование, вырезать, вставить, жирный шрифт, курсив, подчеркнутый текст, зачеркнутый текст, цвет текста, выделение и т. д. Число операций неограничено.

Попробуем найти первый символ. Он располагается на 10 символе. Мы увидим, что число 10 больше, чем 9. Поэтому будем передвигаться по правому краю. Переходим в узел C. Поскольку мы передвинули по правому краю, то при дальнейших действиях 10 отнимем на 9. Получим 1. Продолжаем передвижение. Видим, что число 1 меньше, чем 6. Значит переходим по левой части. Переходим в узел E. Поскольку перешли налево, то отнять не нужно. Далее, поскольку число 1 меньше, чем 2, переходим налево. Переходим в узел I. По аналогичной с предыдущей причиной нам не нужно отнимать числа. Далее обнаруживаем, что это конечный путь. Мы получили число 1 в итоге. Поэтому получаем первый символ. Это символ “д”. Всю процедуру нахождения можно увидеть на рисунке 3.2.

В итоге можно увидеть, что понадобилось меньше времени, чтобы найти данный символ. Это способ более экономный.

Аналогично проделывая действие можно получить подстроку (рисунок 3.3), на которую необходимо проводить операцию. При помощи него можно проделывать любую операцию, которую нам необходимо (рисунок 3.4).

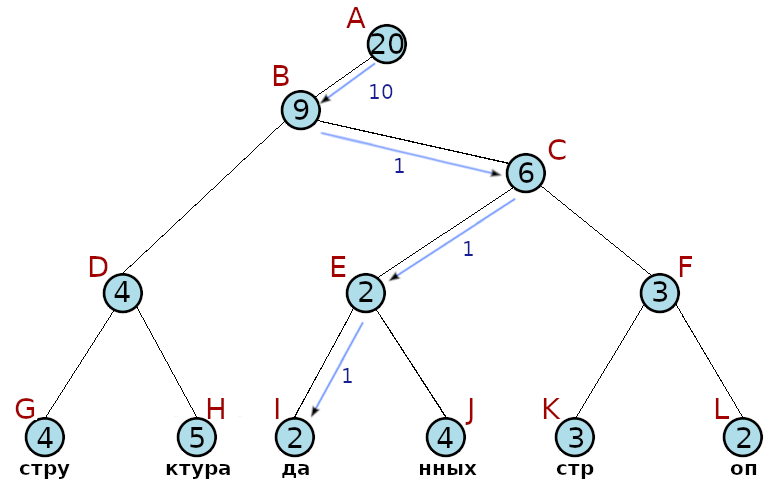


Рисунок 3.2 – Нахождение 10-го символа

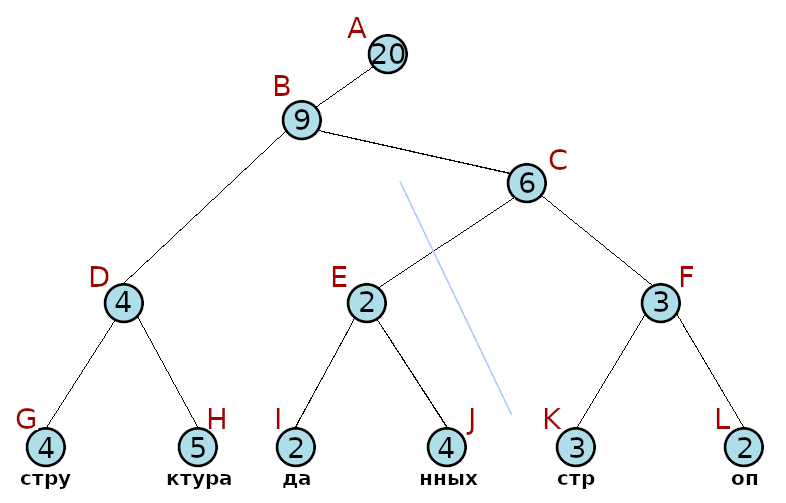


Рисунок 3.3 – Получение подстроки

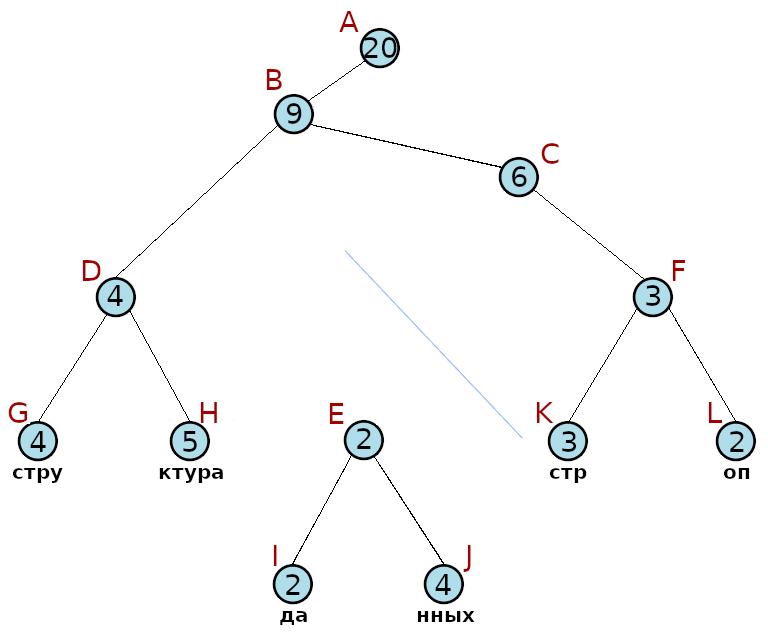


Рисунок 3.4 – Отделение подстроки (выполнение операции)

## **3.2 Вывод по третьей главе**

Исследование показало, что структура данных, известная как строп (канат, корд), предоставляет эффективное хранение и обработку длинных строк, таких как текстовые файлы. Она основана на двоичном дереве, где каждый узел содержит подстроку текста и длину, что обеспечивает эффективное выполнение операций вставки, удаления и обращения (случайного доступа). Важно отметить, что для основных операций предполагается, что строки не изменяются в неразрушающем случае, и могут допускать копирование при записи. Таким образом, строп представляет собой мощный инструмент для работы с текстовыми данными, который может быть использован в различных приложениях для обработки и хранения информации.

В результате исследования работы структуры данных строп на примере строки "СтруктураДанныхСтроп" мы разбили данную строку на части и построили бинарное дерево. В каждом узле дерева содержится информация о количестве символов в подстроке. Проводя операции с подстроками, мы использовали возможности этой структуры данных, такие как перемещение по краю и определение позиций символов.

Конкретно в нашем примере мы попытались найти первый символ подстроки "данных". Мы начали с корневого узла и перемещались по дереву в зависимости от сравнения счетчиков символов. В итоге мы нашли первый символ подстроки, который оказался символом "д".

Таким образом, использование структуры данных строп позволяет эффективно разбивать текстовые строки на части и выполнять различные операции над подстроками. Это может быть полезно в контексте обработки текстовых файлов или других задач, где требуется эффективное управление строковыми данными.

# **Заключение**

На протяжении написания курсовой работы были рассмотрены все тонкости созданного приложения. Были рассмотрены все аспекты по вопросу визуального редактора.

Средой программирования избран Visual Studio Code. Графическим интерфейсом был выбран PyQt5.

В конечном итоге можно считать, что цели, поставленные заранее, были полностью выполнены. Курсовая работа считается завершенной. А приложение WYSIWYG считается готовым к тестированию и публикации.

В результате исследования работы структуры данных строп на примере строки "СтруктураДанныхСтроп" мы разбили данную строку на части и построили бинарное дерево. В каждом узле дерева содержится информация о количестве символов в подстроке. Проводя операции с подстроками, мы использовали возможности этой структуры данных, такие как перемещение по краю и определение позиций символов.

Конкретно в нашем примере мы попытались найти первый символ подстроки "данных". Мы начали с корневого узла и перемещались по дереву в зависимости от сравнения счетчиков символов. В итоге мы нашли первый символ подстроки, который оказался символом "д".

Таким образом, использование структуры данных строп позволяет эффективно разбивать текстовые строки на части и выполнять различные операции над подстроками. Это может быть полезно в контексте обработки текстовых файлов или других задач, где требуется эффективное управление строковыми данными.

# **Список литературы**

PyQt. Создание оконных приложений на Python 3. Н.А. Прохоренко, 2016.

PyQt5 библиотека для создания интерфейсов на Python. Алан Харрис. 2018.

Qt Assistant – интерактивная документация с примерами, входит в стандартную поставку Qt (англ.)

Rapid GUI Programming with Python and Qt. Маартен Лернхаут. 2022.

Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с.

Ж. Бланшет, М. Саммерфилд «QT 4: программирование GUI на С++», КУДИЦ-Пресс, 2008

М. Шлее «Qt 5.3. Профессиональное программирование на C++», БХВ-Петербург, 2015

Прохоренок Н.А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.

Рейтц К., Шлюссер Т. Автостопом по Python. – СПб.: Питер, 2017. – 336 с.: ил. – (Серия «Бестселлеры O’Reilly»).

# **ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ**

from PyQt5.QtWidgets import \*

from PyQt5.QtWidgets import \*

from PyQt5.QtGui import \*

from PyQt5.QtPrintSupport import \*

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import QMainWindow, QAction, qApp, QApplication,QFileDialog,QHBoxLayout,QWidget,QApplication

import sys

def new():

    text.setText("")

def bold():

    if text.fontWeight != QFont.Bold:

        text.setFontWeight(QFont.Bold)

        return

    text.setFontWeight(QFont.Normal)

def italic():

    state = text.fontItalic()

    text.setFontItalic(not (state))

def underline():

    state = text.fontUnderline()

    text.setFontUnderline(not (state))

def strike():

    f = text.font()

    if (f.strikeOut()):

        f.setStrikeOut(False)

    else:

        f.setStrikeOut(True)

    text.setFont(f)

def color():

    newColorDialog = QColorDialog()

    newColor = newColorDialog.getColor()

    text.setTextColor(QColor(newColor.name()))

def highlight():

    newColorDialog = QColorDialog()

    newColor = newColorDialog.getColor()

    text.setTextBackgroundColor(QColor(newColor.name()))

def SetFontSize():

    text.setFontPointSize(fontSize.value())

def SetFontName():

    text.setCurrentFont(QFont(fontName.currentText()))

def SaveTextFile():

    document = text.document()

    file = QFile("file.html")

    if file.open(QFile.WriteOnly):

        OutputText = document.toHtml()

        file.write(OutputText.encode('utf-8'))

        file.close()

        return True

    else:

        return False

def SaveAsTextFile(self):

    fileName, \_ = QFileDialog.getSaveFileName(win,"Сохранить файл","","HTML (\*.html)")

    if fileName:

        document = text.document()

        file = QFile(f"{fileName}.html")

        if file.open(QFile.WriteOnly):

            OutputText = document.toHtml()

            file.write(OutputText.encode('utf-8'))

            file.close()

            return True

        else:

            return False

def LoadTextFile():

    fileName=QFileDialog.getOpenFileName(win,"Открыть файл","","HTML (\*.html)")

    if (fileName[0]):

        f = QFile(fileName[0])

        f.open(QFile.ReadOnly | QFile.Text)

        istream = QTextStream(f)

        text.setHtml(istream.readAll())

        f.close()

app = QApplication(sys.argv)

win = QWidget()

win.resize(975, 500)

layout = QVBoxLayout()

layout.setContentsMargins(0, 0, 0, 0)

text = QTextEdit()

#menu bar

MenuBar = QMenuBar()

FileMenu = QMenu("Файл")

MenuBar.addMenu(FileMenu)

NewMenu = QAction(QIcon("icons/new.png"), "Новый")

NewMenu.setShortcut("Ctrl+N")

NewMenu.triggered.connect(new)

FileMenu.addAction(NewMenu)

FileMenu.addSeparator()

OpenMenu = QAction(QIcon("icons/open.png"), "Открыть")

OpenMenu.setShortcut("Ctrl+O")

OpenMenu.triggered.connect(LoadTextFile)

FileMenu.addAction(OpenMenu)

SaveMenu = QAction(QIcon("icons/save.png"), "Сохранить")

SaveMenu.setShortcut("Ctrl+S")

SaveMenu.triggered.connect(SaveTextFile)

FileMenu.addAction(SaveMenu)

SaveAsMenu = QAction(QIcon("icons/save.png"), "Сохранить как")

SaveAsMenu.triggered.connect(SaveAsTextFile)

FileMenu.addAction(SaveAsMenu)

FileMenu.addSeparator()

QuitMenu = QAction("Выход")

QuitMenu.setShortcut("Ctrl+Q")

QuitMenu.triggered.connect(app.quit)

FileMenu.addAction(QuitMenu)

EditMenu = QMenu("Правка")

MenuBar.addMenu(EditMenu)

UndoMenu = QAction(QIcon("icons/undo.png"), "Отменить")

UndoMenu.setShortcut("Ctrl+Z")

UndoMenu.triggered.connect(text.undo)

EditMenu.addAction(UndoMenu)

RedoMenu = QAction(QIcon("icons/redo.png"), "Возвращать")

RedoMenu.setShortcut("Ctrl+R")

RedoMenu.triggered.connect(text.redo)

EditMenu.addAction(RedoMenu)

EditMenu.addSeparator()

CopyMenu = QAction(QIcon("icons/copy.png"), "Копировать")

CopyMenu.setShortcut("Ctrl+C")

CopyMenu.triggered.connect(text.copy)

EditMenu.addAction(CopyMenu)

CutMenu = QAction(QIcon("icons/cut.png"), "Вырезать")

CutMenu.setShortcut("Ctrl+X")

CutMenu.triggered.connect(text.cut)

EditMenu.addAction(CutMenu)

PasteMenu = QAction(QIcon("icons/paste.png"), "Вставить")

PasteMenu.setShortcut("Ctrl+V")

PasteMenu.triggered.connect(text.paste)

EditMenu.addAction(PasteMenu)

EditMenu.addSeparator()

FontMenu = QMenu("Шрифт")

MenuBar.addMenu(FontMenu)

BoldMenu = QAction(QIcon("icons/bold.png"), "Жирный")

BoldMenu.setShortcut("Ctrl+B")

BoldMenu.triggered.connect(bold)

FontMenu.addAction(BoldMenu)

ItalicMenu = QAction(QIcon("icons/italic.png"), "Курсив")

ItalicMenu.setShortcut("Ctrl+I")

ItalicMenu.triggered.connect(italic)

FontMenu.addAction(ItalicMenu)

UnderlineMenu = QAction(QIcon("icons/underline.png"), "Подчеркнутый")

UnderlineMenu.setShortcut("Ctrl+U")

UnderlineMenu.triggered.connect(underline)

FontMenu.addAction(UnderlineMenu)

StrikeMenu = QAction(QIcon("icons/strike.png"), "Зачеркнутый")

StrikeMenu.setShortcut("Ctrl+Y")

StrikeMenu.triggered.connect(strike)

FontMenu.addAction(StrikeMenu)

FontMenu.addSeparator()

TextColorMenu = QAction(QIcon("icons/font-color.png"), "Цвет текста")

TextColorMenu.setShortcut("Ctrl+M")

TextColorMenu.triggered.connect(color)

FontMenu.addAction(TextColorMenu)

HighLightColorMenu = QAction(QIcon("icons/highlight.png"), "Цвет выделения")

HighLightColorMenu.setShortcut("Ctrl+H")

HighLightColorMenu.triggered.connect(highlight)

FontMenu.addAction(HighLightColorMenu)

FontMenu.addSeparator()

LeftAlignMenu = QAction(QIcon("icons/align-left.png"), "Выравнивание по левому краю")

LeftAlignMenu.setShortcut("Ctrl+L")

LeftAlignMenu.triggered.connect(lambda : text.setAlignment(Qt.AlignLeft))

FontMenu.addAction(LeftAlignMenu)

CenterAlignMenu = QAction(QIcon("icons/align-center.png"), "Выравнивание посередине")

CenterAlignMenu.setShortcut("Ctrl+E")

CenterAlignMenu.triggered.connect(lambda : text.setAlignment(Qt.AlignCenter))

FontMenu.addAction(CenterAlignMenu)

RightAlignMenu = QAction(QIcon("icons/align-right.png"), "Выравнивание по правому краю")

RightAlignMenu.setShortcut("Ctrl+T")

RightAlignMenu.triggered.connect(lambda : text.setAlignment(Qt.AlignRight))

FontMenu.addAction(RightAlignMenu)

JustifyAlignMenu = QAction(QIcon("icons/align-justify.png"), "Выравнивание по ширине")

JustifyAlignMenu.setShortcut("Ctrl+J")

JustifyAlignMenu.triggered.connect(lambda : text.setAlignment(Qt.AlignJustify))

FontMenu.addAction(JustifyAlignMenu)

layout.addWidget(MenuBar)

#menu bar end

toolbar = QToolBar()

win.setLayout(layout)

win.setWindowTitle("Rich text edit")

layout.addWidget(toolbar)

NewAction = QAction(QIcon("icons/new.png"), "Новый")

NewAction.triggered.connect(new)

toolbar.addAction(NewAction)

SaveAction = QAction(QIcon("icons/save.png"), "Сохранить")

SaveAction.triggered.connect(SaveTextFile)

toolbar.addAction(SaveAction)

LoadAction = QAction(QIcon("icons/open.png"), "Открыть")

LoadAction.triggered.connect(LoadTextFile)

toolbar.addAction(LoadAction)

toolbar.addSeparator()

UndoAction = QAction(QIcon("icons/undo.png"), "Отменить")

UndoAction.triggered.connect(text.undo)

toolbar.addAction(UndoAction)

RedoAction = QAction(QIcon("icons/redo.png"), "Возвращать")

RedoAction.triggered.connect(text.redo)

toolbar.addAction(RedoAction)

CopyAction = QAction(QIcon("icons/copy.png"), "Копировать")

CopyAction.triggered.connect(text.copy)

toolbar.addAction(CopyAction)

CutAction = QAction(QIcon("icons/cut.png"), "Вырезать")

CutAction.triggered.connect(text.cut)

toolbar.addAction(CutAction)

PasteAction = QAction(QIcon("icons/paste.png"), "Вставить")

PasteAction.triggered.connect(text.paste)

toolbar.addAction(PasteAction)

toolbar.addSeparator()

BoldAction = QAction(QIcon("icons/bold.png"), "Жирный")

BoldAction.triggered.connect(bold)

toolbar.addAction(BoldAction)

ItalicAction = QAction(QIcon("icons/italic.png"), "Курсив")

ItalicAction.triggered.connect(italic)

toolbar.addAction(ItalicAction)

UnderlineAction = QAction(QIcon("icons/underline.png"), "Подчеркнутый")

UnderlineAction.triggered.connect(underline)

toolbar.addAction(UnderlineAction)

StrikeAction = QAction(QIcon("icons/strike.png"), "Зачеркнутый")

StrikeAction.triggered.connect(strike)

toolbar.addAction(StrikeAction)

ColorAction = QAction(QIcon("icons/font-color.png"), "Цвет текста")

ColorAction.triggered.connect(color)

toolbar.addAction(ColorAction)

HighlightAction = QAction(QIcon("icons/highlight.png"), "Выделение")

HighlightAction.triggered.connect(highlight)

toolbar.addAction(HighlightAction)

toolbar.addSeparator()

SizeLabel = QLabel("Размер:")

toolbar.addWidget(SizeLabel)

fontSize = QSpinBox()

fontSize.valueChanged.connect(SetFontSize)

toolbar.addWidget(fontSize)

FontLabel = QLabel("Шрифт:")

toolbar.addWidget(FontLabel)

fontName = QComboBox()

fontName.addItems(["Courier Std", "Hellentic Typewriter Regular", "Helvetica", "Arial", "SansSerif", "Helvetica", "Times", "Monospace"])

fontName.activated.connect(SetFontName)

toolbar.addWidget(fontName)

toolbar.addSeparator()

AlignLeftAction = QAction(QIcon("icons/align-left.png"), "Выравнивание по левому краю")

AlignLeftAction.triggered.connect(lambda : text.setAlignment(Qt.AlignLeft))

toolbar.addAction(AlignLeftAction)

AlignCenterAction = QAction(QIcon("icons/align-center.png"), "Вырванивание посередине")

AlignCenterAction.triggered.connect(lambda : text.setAlignment(Qt.AlignCenter))

toolbar.addAction(AlignCenterAction)

AlignRightAction = QAction(QIcon("icons/align-right.png"), "Выравнивание по правому краю")

AlignRightAction.triggered.connect(lambda : text.setAlignment(Qt.AlignRight))

toolbar.addAction(AlignRightAction)

AlignJustifyAction = QAction(QIcon("icons/align-justify.png"), "Выравнивание по ширине")

AlignJustifyAction.triggered.connect(lambda : text.setAlignment(Qt.AlignJustify))

toolbar.addAction(AlignJustifyAction)

layout.addWidget(text)

win.show()

sys.exit(app.exec\_())