Санкт-Петербургский государственный университет

Ариткулов Даниил Наилевич

Выпускная квалификационная работа

Создание IDE для нового языка программирования на основе расширенного JSON

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5162.2020 «Технологии программирования»*

Научный руководитель:
доцент, кандидат ф.-м. наук Лебединский Д.М,

Рецензент:
старший преподаватель кафедры информатики, к.ф.-м.н. Н. Ю. Ловягин

Санкт-Петербург
2024

Saint Petersburg State University

Aritkulov Daniil

Bachelor's Thesis

Development of IDE for a new programming language based on extended JSON

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5162.2020 «Programming Technologies»*

Scientific supervisor:
C.Sc., docent D.M. Lebedinskii

Reviewer:
C.Sc., senior lecturer N.Y. Lovyagin

Saint Petersburg
2024

Оглавление

Введение

В современном информационном мире, где скорость развития технологий непрерывно увеличивается, процесс создания программного обеспечения становится все более сложным и разнообразным. Появление новых языков программирования не только отражает этот непрерывный технологический прогресс, но и является ответом на постоянно изменяющиеся требования рынка и потребности разработчиков. В таком контексте создание инструментов, способствующих более удобному и эффективному программированию, становится необходимостью. А производство интегрированной среды разработки (IDE)[4] для нового языка программирования играет ключевую роль в его успешном внедрении и распространении среди специалистов.

JSON (JavaScript Object Notation)[6], известный своей простотой, гибкостью, универсальностью и широким распространением, уже хорошо зарекомендовал себя как удобный формат обмена данными в различных технологических областях. Его внутренняя структура, характеризующаяся облегченным синтаксисом и легко читаемым видом, естественным образом подходит для множества приложений, от веб-разработки до сериализации данных и т.д. Однако, его возможности могут быть значительно расширены. Помимо своей традиционной роли формата обмена данными, JSON таит в себе неиспользованный потенциал, который может служить основой для языка программирования, что представляет огромный интерес для исследования и разработки. Это стремление мотивировано признанием врожденных возможностей JSON выражать сложные алгоритмы в форме структурированного списка команд, тем самым устраняя разрыв между представлением данных и исполняемым кодом. Мы рассматриваем JSON как язык программирования.

В данной работе особое внимание уделяется двум аспектам: во-первых, концептуализировать и разработать новый язык программирования, который использует JSON в качестве списка команд для алгоритмического выражения. Во-вторых, спроектировать и разработать интуитивно понятную и многофункциональную интегрированную среду разработ-

ки, специально предназначенную для этого языка, предоставляющую разработчикам удобный интерфейс и инструменты для эффективного создания, визуализации и выполнения кода.

В последующих разделах будут описаны основные принципы нового языка программирования, процесс разработки архитектуры IDE, обеспечивающей гибкость и расширяемость для поддержки будущих изменений и добавления новых функций, определены основные компоненты, их взаимосвязи и способы взаимодействия.

1. Постановка задачи

Целью данной работы является создание и реализация интегрированной среды разработки для нового языка программирования на основе JSON, обеспечивающей удобное создание и управление проектами. Для её достижения были поставлены следующие задачи:

1. Исследовать особенности нового языка программирования.
2. Спроектировать архитектуру IDE, учитывая требования нового языка программирования.
3. Реализовать редактор кода.
4. Реализовать интерпретатор для нового языка программирования.
5. Интегрировать основные компоненты.
6. Протестировать интерпретатор и его взаимодействие с редактором кода.

2. Обзор

2.1. Обзор существующих языков программирования

2.1.1. JSOL[5]

JSOL (JavaScript Object Language) — это интерпретируемый язык, реализованный на языке программирования Python. Он позволяет создавать программы с использованием строк, чисел, списков, словарей, замыканий и первоклассных функций. Основной особенностью JSOL является то, что программы на этом языке записываются в формате JSON.

2.1.2. PSOL[5]

PSOL — это тот же JSOL, но с более чистым синтаксисом. Из-за этого различия программы на PSOL не могут быть представлены в формате JSON. Однако создатель этих языков предоставляет функцию, которая позволяет преобразовать программу, написанную на PSOL, в запись пригодную для JSOL.

2.2. Обзор основных положений нового языка программирования

2.2.1. Основные свойства и особенности

JSON — это облегченный формат обмена данными, обычно используется для передачи информации между клиентом и сервером в веб-разработке, а также для ее хранения в различных приложениях и системах. Его популярность обусловлена его простотой и совместимостью с большинством языков программирования. Он состоит из пар ключ-значение, организованных в объекты и массивы, обеспечивая гибкую и удобочитаемую структуру для представления сложных иерархий данных.

В качестве основы для нового языка программирования используется JSON, где каждый элемент интерпретируется как отдельная команда, написанная на русском языке, что дает ряд преимуществ. Во-первых, простой синтаксис формата позволяет лаконично и интуитивно выражать инструкции. Каждый элемент массива JSON может представлять отдельное действие или инструкцию, повышая модульность и ясность кодовой базы. Это делает язык максимально доступным и понятным для широкого круга пользователей, в том числе для тех, кто только начинает свой путь в программировании. Также, поскольку формат является текстовым, его легко расширять и изменять без необходимости внесения существенных модификаций в организацию языка. Во-вторых, использование русских слов для команд добавляет уровень доступности и знакомства для русскоязычных разработчиков. Используя свою родную лексику, разработчики могут легче понимать и выражать логику программы, сокращая возможные недопонимания, что способствует более быстрой разработке кода.

Независимая от платформы природа формата обеспечивает совместимость и взаимодействие в различных средах и системах. Программы, написанные на этом языке, могут легко интегрироваться с существующими API (Application Programming Interface)[13] и библиотеками на основе JSON, что позволяет разработчикам использовать обширную экосистему инструментов и ресурсов.

2.2.2. Список команд и принципы их работы

Как было упомянуто ранее, программа представляет собой массив JSON, где каждый элемент интерпретируется как отдельная команда. Все инструкции выполняются последовательно.

Список команд и их описание:

- «ввод» — ввести значение и положить его во внутренний стек.
- «вывод» — взять значение из стека и вывести его.
- «взять» — имеет два варианта:

1. если команда — это структура, и в ней есть поле с ключом «что», содержимое этого поля кладется в стек.
2. если нет, то из стека берется ссылка, и вместо нее кладется то, на что она ссылается.

   Ссылка — это другой объект, у которого есть поле с ключом «режим». Его содержимое показывает, как надо трактовать содержимое других полей. Возможны следующие варианты:

   а) режим «прямой», тогда у ссылки есть поле с ключом «адрес», в котором находится указатель на то, на что ссылается наша ссылка.

   б) режим «регистр», тогда у ссылки есть поле «имя», в котором содержится строка — имя регистра. В этом случае ссылка ссылается на поле с таким именем в наборе регистров.

- «положить» — берет из стека ссылку и значение и записывает значение по указанной ссылке.
- число — положить его в стек данных.
- объект — взять у него значение атрибута «имя» и выполнить как команду (весь объект команды передается функции, выполняющей эту команду, как единственный параметр, по ссылке).
- строка — выполнить команду с таким именем, в том числе: арифметическая, логическая, операция сравнения — взять из стека нужное количество чисел, вычислить выражение и положить результат в стек.
- «метка» — такая команда должна быть объектом с дополнительным полем с ключом «метка», в котором записана строка — имя метки. Как команда, ничего не делает и служит только для того, чтобы пометить определенное место в последовательности команд.
- «переход» — такая команда должна быть объектом с дополнительным полем с ключом «куда», в котором записана строка — имя

метки. Как команда, вызывает передачу управления на метку с указанным именем.

- «если» — такая команда должна быть объектом с дополнительными полями с ключами «да» и «нет», в которых записаны строки — имена меток. Как команда, берет значение из стека и вызывает передачу управления на метку с именем под ключом «да», если взятое из стека значение отличается от 0, и «нет» — в противном случае. Поле с ключом «нет» может отсутствовать, в этом случае, если снятое из стека значение равно 0, управление, как обычно, переходит на следующую по порядку команду.

В приведенном ниже примере демонстрируется программа, созданная на новом языке программирования. Она выполняет чтение двух чисел, сохраняет введенные данные и выводит результат их произведения.

Листинг 1: Пример программы

```
[
    "ввод",
    {"имя": "взять", "что": {"режим":"регистр", "имя":"а"}},
    "положить",
    "ввод",
    {"имя": "взять", "что": {"режим":"регистр", "имя":"б"}},
    "положить",
    {"имя": "взять", "что": {"режим":"регистр", "имя":"а"}},
    "взять",
    {"имя": "взять", "что": {"режим":"регистр", "имя":"б"}},
    "взять",
    "*",
    "вывод"
]
```

На данный момент существующие языки программирования, основанные на JSON, имеют большую функциональность. Однако данная работа является частью начинания по созданию нового решения, обладающего гибкостью и масштабируемостью, позволяющими добавлять

новые концепции. Изначально опираясь на JSON как минимальную основу, этот язык предоставляет возможности для значительного расширения, позволяя программистам описывать свои программы в различных стилях. В перспективе язык можно будет модифицировать так, что программы будут выглядеть совершенно иначе. Таким образом, закладывается фундамент для создания мощного и универсального инструмента.

2.3. Используемые инструменты

В качестве основного инструмента был выбран статически типизированный язык программирования C++, который известен своей высокой производительностью, портативностью и универсальностью. Более того, для него доступны библиотеки, которые отлично подходят для наших нужд.

2.3.1. Nlohmann JSON[8]

Nlohmann JSON — популярная библиотека C++ для работы с данными JSON. Она предоставляет удобные классы и функции для анализа, создания, манипулирования и сериализации/десериализаци данных JSON в приложениях C++. Библиотека известна своей простотой, удобством использования и высокой производительностью.

Некоторые ключевые особенности:

- Библиотека предлагает простой API, который позволяет разработчикам работать с данными JSON, используя знакомый синтаксис C++. Это упрощает интеграцию функций JSON в существующую кодовую базу C++.
- Разработана для совместимости со стандартной библиотекой шаблонов (STL)[12], что обеспечивает бесшовную интеграцию с другими структурами данных и алгоритмами C++.
- Проста в интеграции, так как состоит из одного заголовка, который можно включить непосредственно в исходный код C++. Это

позволяет легко интегрировать ее в проекты без необходимости отдельных этапов компиляции или связывания.

- Обеспечивает высокую производительность, с упором на минимизацию непроизводительных затрат памяти и максимизацию скорости синтаксического анализа и сериализации. Это достигается за счет эффективных структур данных и алгоритмов.

2.3.2. ncursesw[11][7][3]

ncursesw — это библиотека предназначенная для языков программирования C и C++, обеспечивающая создание текстовых пользовательских интерфейсов (TUI) в терминальной среде.

Основные возможности:

- Позволяет разработчикам разделить окно терминала на несколько виртуальных, каждое из которых может содержать собственный контент и управляться независимо, что помогает создавать сложные макеты и конструкции интерфейса.

- Доступна во многих Unix-подобных операционных системах.

- Предоставляет функции для обработки событий пользовательского ввода, таких как события клавиатуры и мыши, что позволяет разработчикам создавать интерактивные приложения TUI.

- Поддерживает форматирование текста, способствуя созданию визуально привлекательных интерфейсов.

- Буква «w» (wide) в названии означает «широкие символы», что указывает на то, что библиотека поддерживает многобайтовые и расширенные кодировки символов, что делает ее пригодным для локализации и поддержки наборов символов, отличных от ASCII[1], включая UTF-8[9].

2.3.3. Catch2 [2]

Catch2 — это фреймворк для тестирования кода на языке C++, который упрощает процесс написания и выполнения тестов. Он легко интегрируется в проекты, требуя минимальной настройки, и поддерживает как простые, так и сложные тестовые сценарии.

В общем, применение этих инструментов способствует достижению нашей цели: Nlohmann упрощает обработку алгоритма, представленного в формате JSON, в то время как ncursesw обеспечивает создание текстовых пользовательских интерфейсов, включающих в себя различные элементы, такие как окна, меню, кнопки и текстовые поля, что позволяет реализовать редактор кода. Catch2, в свою очередь, предоставляет возможности для удобного и эффективного тестирования кода, что играет ключевую роль в процессе разработки высококачественного программного обеспечения.

3. Реализация

Процесс создания интегрированной среды разработки был разбит на две основные части: разработка интерпретатора для нового языка программирования и реализация редактора кода, предназначенного для него.

3.1. Интерпретатор

В данном разделе представлена организация и описание реализации данного модуля.

3.1.1. Архитектура интерпретатора

Изображение на рис. 1 иллюстрирует основные компоненты: Parser, Context и CommandProcessor. Далее приведены подробности их функциональности и задач.

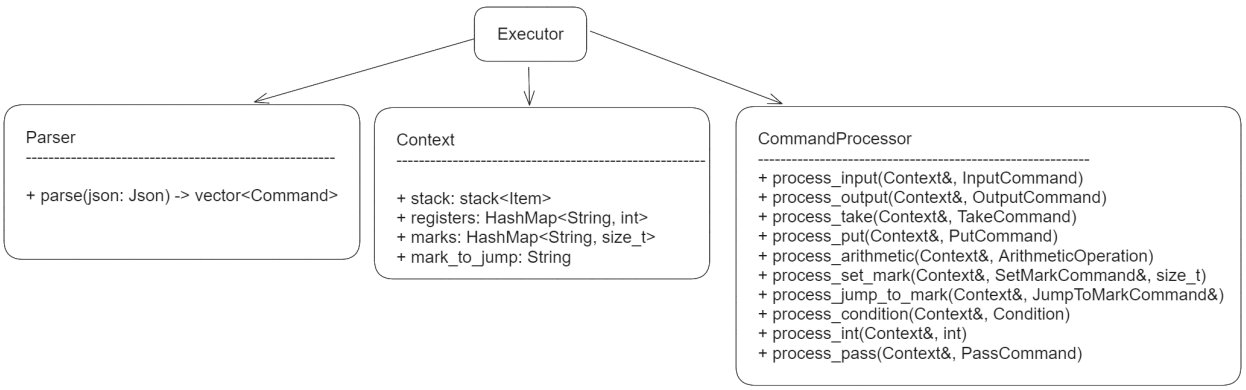

Рис. 1: Архитектура интерпретатора

3.1.2. Parser

Выполняет преобразование программы, созданной с использованием формата JSON, в разнообразные языковые конструкции или входные данные, необходимые для дальнейшего анализа или обработки. Включает в себя перевод команд и инструкций из представления в формате JSON в форму, понятную компилятору. Кроме того, Parser может выполнять и другие манипуляции с информацией, такие как фильтрация,

преобразование типов, или проверку корректности структуры программы перед ее дальнейшим преобразованием.

3.1.3. Context

Данный компонент содержит обширную информацию о текущем состоянии программы. В его основном наборе данных включены сведения о состоянии стека, где хранятся временные данные, а также информация о регистрах, которые являются важными для выполнения различных операций и обеспечения правильной работы программы. Более того, также хранит информацию о точках перехода в программе, которая включает в себя адреса инструкций, на которые может быть выполнен переход в процессе выполнения программы. Context играет ключевую роль в обеспечении корректного исполнения команд и контроля за ходом работы программы.

Структура стека реализована с использованием <vector>, который способен хранить числа, объекты типа «Ссылка», а также условия, необходимые для выполнения операций сравнения.

В C++ <vector> является контейнером STL, предоставляющим динамический массив элементов одного типа. Он обеспечивает автоматическое управление памятью и динамическое изменение размера массива в процессе выполнения программы, предоставляет доступ к элементам по индексу, поддерживает вставку и удаление элементов в конце массива, а также обеспечивает эффективный доступ к элементам благодаря линейной памяти.

3.1.4. CommandProcessor

Содержит набор обработчиков, каждый из которых предназначен для выполнения определенной команды, доступной в языке программирования. Обработчики представляют собой функции, которые принимают входные данные, анализируют их и выполняют соответствующие действия в соответствии с инструкцией. Такой подход обеспечивает эффективную и структурированную обработку всех доступных команд

в рамках языка программирования.

Такой подход к архитектуре обеспечивает возможность легкого расширения функциональности системы. Когда появляется новая команда, достаточно создать для неё отдельный класс и реализовать соответствующий обработчик. Это позволяет легко адаптироваться к изменениям и добавлению новых возможностей без необходимости внесения значительных изменений в структуру или логику работы программы.

3.1.5. Сценарий работы

Процесс начинается с чтения данных из файла, указанного в качестве главного параметра, который содержит алгоритм. После этого используется библиотека Nlohmann и происходит десериализация JSON для создания итерируемого объекта, который последовательно просматривается. В ходе просмотра каждый элемент добавляется в <vector>. Затем идет итерация по этому массиву, и для каждой команды запускается соответствующий обработчик. Работа программы завершается по окончании итерации. Однако неправильное использование меток перехода может привести к зацикливанию программы в бесконечном цикле.

3.2. Редактор кода

Редактор кода реализован с использованием текстового пользовательского интерфейса, что обеспечивает ряд преимуществ:

- Универсальность: TUI-редакторы могут работать на различных операционных системах и терминальных средах без необходимости адаптации под каждую из них. Это делает их удобными для использования на различных платформах.
- Эффективность использования ресурсов: такие редакторы обычно работают легче и быстрее, чем графические интерфейсы, так как не требуют использования графической подсистемы операционной системы.

- Простота в настройке и управлении: предоставляют простой и интуитивно понятный интерфейс, который легко настраивать под свои нужды и управлять с помощью клавиатуры.

- Совместимость с консольными приложениями: могут взаимодействовать с другими консольными приложениями и командами операционной системы, что делает их полезными инструментами для автоматизации задач и работы в консольной среде.

В целом, использование TUI для редактора кода обеспечивает производительность и удобство использования, что делает его хорошим выбором.

В основе реализации лежат ключевые классы, такие как Editor, ToolBar и IDE.

3.2.1. Editor

Основными объектами данного класса являются buffer_ (тип WINDOW) и contents_ (тип wstring). В поле contents_ хранится текст, который затем обрабатывается с использованием функций библиотеки и отображается в нужном виде и формате внутри buffer_.

WINDOW — это одна из особенностей ncursesw. Представляет собой структуру данных, которая определяет окно на экране терминала. Она содержит информацию о размерах окна, его положении на экране, а также о его содержимом. Каждое окно имеет свой собственный набор символов и атрибутов, которые определяют его видимость и внешний вид. С помощью функций библиотеки можно создавать, изменять, перемещать и удалять окна, а также управлять их содержимым, включая текст и цвет. Окна могут быть организованы в стек или располагаться рядом друг с другом, что позволяет создавать различные макеты пользовательского интерфейса в текстовой терминальной среде.

Поскольку программа представляет собой набор команд на русском языке, было решено использовать тип данных, поддерживающий Unicode[10] символы, чтобы обеспечить корректную работу с русским текстом. В качестве такого типа данных был выбран wstring, который представляет

собой строковый тип данных в стандартной библиотеке C++, поддерживающий широкий набор символов. Использование wstring позволяет программе корректно обрабатывать и отображать текст на русском языке, что сделало его оптимальным выбором для работы с записью алгоритма в данном проекте.

Этот класс является основным модулем, ответственным за обработку и отображение текстовой информации, с которой пользователь взаимодействует. В его функционал входят методы для управления отображением текста, его форматированием, а также перемещением курсора и изменением размеров рабочего окна. Он также обеспечивает возможность ввода текста пользователем с клавиатуры и обработку введенных символов. Таким образом, данный класс представляет собой ключевой компонент взаимодействия пользователя с текстовым интерфейсом.

3.2.2. ToolBar

ToolBar представляет собой важную часть редактора кода, предоставляя пользователю набор основных функций через текстовые метки, размещенные в области экрана, управляемой полем buffer_ типа WINDOW. Эти функции включают в себя «Выход» (для завершения работы программы), «Закрыть» (для закрытия текущего файла), «Создать» (для создания нового файла), «Открыть» (для открытия существующего файла) и «Сохранить как» (для сохранения файла в указанном месте). При нажатии определенных клавиш значение поля highlight_ (тип uint8_t) изменяется, что позволяет программе понять выбранную пользователем функцию.

3.2.3. IDE

Это основной класс, представляющий собой центральное звено всей программы, играет ключевую роль в её жизненном цикле, начиная от запуска и заканчивая завершением. В нем реализована вся бизнес-логика функционирования редактора кода, включая инициализацию необходимых объектов, координацию их взаимодействия, обеспечение

их корректной работы, а также процессы загрузки и сохранения файлов. Этот класс отвечает за отображение, настройку, очистку и обновление всех окон на экране терминала. Главное окно, содержащее инструкции по использованию приложения, также находится под его управлением.

На рис. 2 продемонстрирован интерфейс редактора кода.

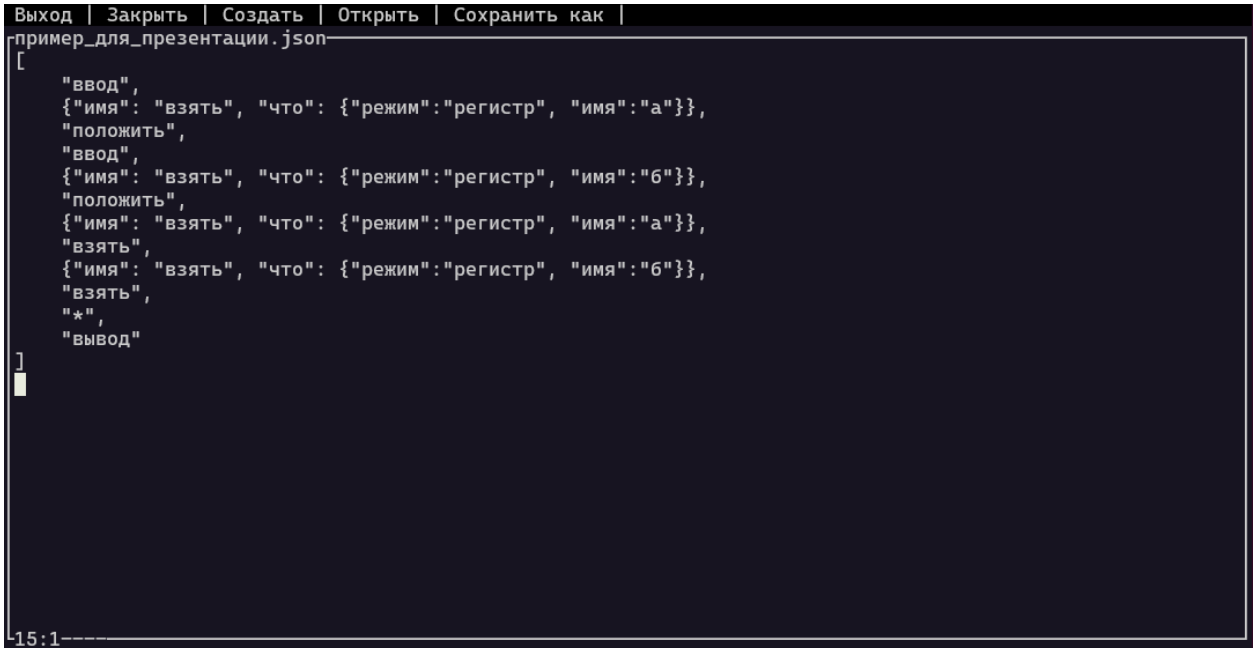

Рис. 2: Интерфейс редактора кода

3.3. Интеграция

После завершения разработки двух основных компонентов — интерпретатора и редактора кода, был произведен процесс интеграции, целью которой было обеспечить совместную работу обеих частей, а именно, возможность запуска интерпретатора непосредственно из редактора кода, что позволит пользователю выполнять программы, написанные на новом языке программирования, и просматривать их результаты в одном рабочем окружении.

На рис. 3 показано взаимодействие компонент при работе пользователя с программой.

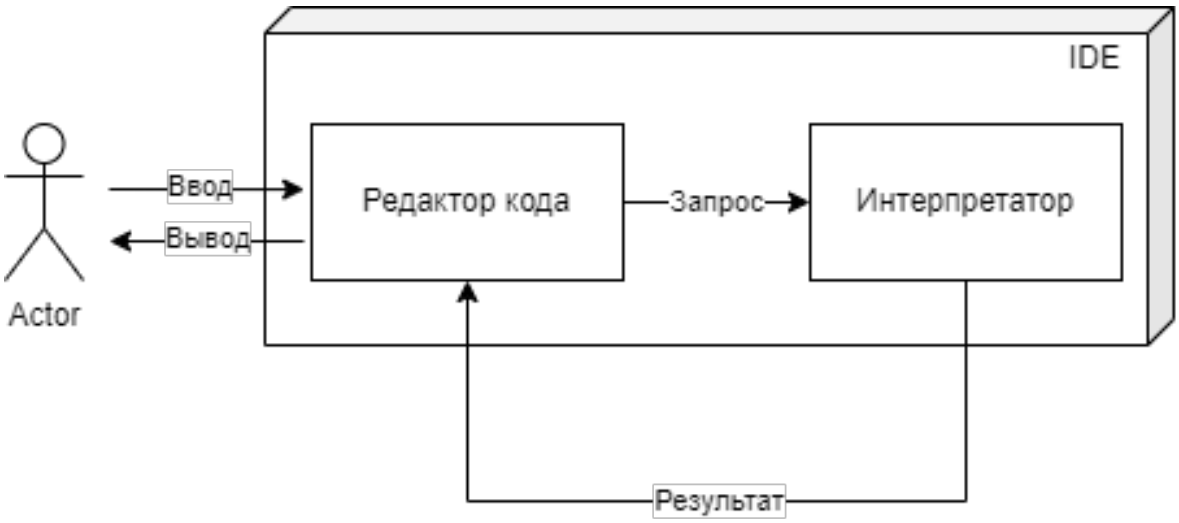

Рис. 3: Архитектура IDE

3.3.1. Output

Этот класс играет ключевую роль в отображении результатов работы программы после ее запуска. По своей структуре и функциональности он схож с классом Editor (редактора кода). Здесь основным полем является contents_, которое содержит информацию, выводимую на экран. После запуска программы результат ее работы обновляется и сохраняется в этом поле в виде текста типа wstring, который затем предоставляется пользователю по необходимости. Основное различие с классом Editor заключается в том, что здесь отсутствует функция редактирования текста. Пользователь может только вводить данные в окно buffer_ типа WINDOW, если это требуется программой.

3.3.2. Запуск

Для запуска программы пользователю предоставлена новая функция «Запуск» в ToolBar. При выборе этой функции начинает работать интерпретатор, который взаимодействует с текстом, находящимся в окне редактора на текущий момент. В результате процесса интеграции интерпретатор уже не выводит информацию в консоль, как это было ранее, а передает ее в объект класса Output, в котором она обрабатывается и выводится в нужном формате.

Пример работы программы после ее запуска изображен на рис. 4.

Рис. 4: Результат работы программы

3.4. Репозиторий

По ссылке можно перейти в репозиторий проекта на GitHub, где настроен пайплайн для непрерывной интеграции (CI). Этот пайплайн включает в себя автоматизированную сборку проекта и запуск тестов, что позволяет нам быстро выявлять и исправлять ошибки.

В репозитории также присутствует файл ReadMe, который содержит подробные инструкции по сборке и запуску программы. Кроме того, этот файл содержит информацию о синтаксисе языка программирования, примеры кода и описание сценариев использования программы.

4. Тестирование

Тестирование является важной и необходимой частью разработки программного обеспечения, так как оно позволяет выявить ошибки, которые могут быть неочевидны на первый взгляд. В ходе тестирования была проведена всесторонняя проверка работы интерпретатора, а также его взаимодействия с редактором кода. Эти тесты охватывали различные сценарии и случаи использования, что позволило выявить потенциальные проблемы и отклонения в поведении программы.

Благодаря тестированию удалось обнаружить множество скрытых ошибок и багов, что значительно улучшило качество и стабильность системы. Тесты служат не только для выявления текущих проблем, но и являются основой для будущих улучшений. Они позволяют быстро и эффективно обнаружить любые нарушения в работе кода при внесении новых изменений и добавлении новых функций. Это делает процесс разработки более надежным и предсказуемым, обеспечивая высокое качество конечного продукта.

Тестовые сценарии:

- Арифметические и логические операции. В рамках тестирования создается экземпляр класса Output, в который впоследствии записывается результат выполнения программы. Для каждого теста задается путь к JSON-файлу, содержащему запись соответствующей программы. Процесс проверки включает сравнение ожидаемого результата с полученным значением.

  Эти тесты не только проверяют корректность выполнения арифметических и логических операций интерпретатором, но также оценивают его взаимодействие с редактором кода. Это достигается за счет сравнения как значения, так и его передачи в буфер экземпляра Output. Таким образом, одновременно проверяются корректность вычислений и правильность передачи данных, что позволяет убедиться в корректной работе всей системы в целом.

- Деление на ноль. Одним из особых случаев в арифметических опе-

рациях является деление на ноль, которое требует отдельного рассмотрения. Для проверки этого сценария в интерпретатор передается программа, в которой выполняется операция деления на ноль. Затем проводится проверка того, что интерпретатор правильно обрабатывает эту ситуацию и оповещает об ошибке. Ожидается, что интерпретатор передаст сообщение «Деление на ноль» для вывода, тем самым информируя о возникновении ошибки. Этот тест гарантирует, что система корректно обрабатывает критические исключения и предоставляет пользователю соответствующую информацию.

- Несоответствие записи программы. Поскольку язык программирования имеет свой уникальный синтаксис и определенные возможности, запись программы должна соответствовать установленным требованиям и правилам языка. В рамках этих тестов проводится проверка типов данных и команд, используемых в программе, а также их согласованность со структурой языка. Ожидается, что при обнаружении несоответствия интерпретатор выдаст сообщение об ошибке, указывая на нарушение правил языка. Эти тесты обеспечивают гарантии того, что программы, написанные на данном языке, будут соответствовать его спецификациям и корректно интерпретироваться системой.

- Несоответствие формату JSON. Поскольку программы на данном языке представляются в виде файлов формата JSON, важно, чтобы интерпретатор корректно обрабатывал только те записи, которые соответствуют этому формату. Если программа не соответствует JSON-формату, интерпретатор должен сообщить об ошибке. В рамках этих тестов проводится проверка на соответствие требованиям формата JSON. Ожидается, что при обнаружении несоответствия интерпретатор выдаст сообщение об ошибке, указывая на нарушение формата. Эти тесты обеспечивают правильную обработку данных и гарантируют, что все входные программы корректно структурированы.

Все тесты проходят успешно, что подтверждает корректность работы интерпретатора, его взаимодействие с редактором кода и правильную обработку крайних случаев.

Заключение

В ходе работы были выполнены следующие задачи:

- Исследованы особенности нового языка программирования.
- Спроектирована архитектура IDE.
- Реализован редактор кода.
- Реализован интерпретатор для нового языка программирования.
- Произведен процесс интеграции основных компонент.
- Протестирован интерпретатор и его взаимодействие с редактором кода.

Список литературы

[1] ASCII. — URL: https://www.ascii-code.com/ (дата обращения: 2023-10-10).

[2] Catch 2. — URL: https://catch2-temp.readthedocs.io/en/latest/ (дата обращения: 2024-04-19).

[3] How to ncursesw. — URL: https://superjamie.github.io/2022/08/06/ncursesw (дата обращения: 2023-11-13).

[4] IDE и редакторы кода для разработчиков. — URL: https://habr.com/ru/companies/serverspace/articles/693374/ (дата обращения: 2023-10-25).

[5] JSOL. — URL: http://www.jsol.org (дата обращения: 2024-05-20).

[6] JSON. — URL: https://www.json.org/json-en.html (дата обращения: 2023-09-22).

[7] NCURSES Programming HOWTO. — URL: https://tldp.org/HOWTO/NCURSES-Programming-HOWTO/index.html (дата обращения: 2023-10-11).

[8] Nlohmann JSON. — URL: https://json.nlohmann.me/ (дата обращения: 2023-09-28).

[9] UTF-8. — URL: https://www.unicode.org/faq/utf_bom.html#UTF8 (дата обращения: 2023-10-10).

[10] The Unicode Standard: A Technical Introduction. — URL: https://www.unicode.org/standard/principles.html (дата обращения: 2023-10-10).

[11] ncursesw. — URL: https://invisible-island.net/ncurses/man/ncurses.3x.html (дата обращения: 2023-12-19).

[12] Справочник по стандартной библиотеке C++ (STL). — URL: https://learn.microsoft.com/ru-ru/cpp/standard-library/cpp-standard-library-reference?view=msvc-170 (дата обращения: 2023-10-10).

[13] Что такое API. — URL: https://habr.com/ru/articles/464261/ (дата обращения: 2023-10-25).