

**Оглавление**

[**1. Введение 4**](#_Toc200443592)

[**1. Основная часть 5**](#_Toc200443593)

[**2.1 Анализ предметной области 5**](#_Toc200443594)

[**2.1.1 Выбор языка программирования и среды разработи 5**](#_Toc200443595)

[**2.1.2 Вывод 7**](#_Toc200443596)

[**2.2 Разработка программы 8**](#_Toc200443597)

[**2.2.1 Описание грамматики входного языка (LaTeX) 8**](#_Toc200443598)

[**2.2.2 Разработка лексического анализатора 11**](#_Toc200443599)

[**2.2.3 Построение синтаксического анализатора 14**](#_Toc200443600)

[**2.2.4 Формирование абстрактного синтаксического дерева (AST). 16**](#_Toc200443601)

[**2.2.5 Генерация XML на основе AST. 19**](#_Toc200443602)

[**2.3 Вспомогательные программы 22**](#_Toc200443603)

[**Заключение 24**](#_Toc200443604)

[**Список литературы: 25**](#_Toc200443605)

1. **Введение**

Эксплуатационная практика проходила в Федеральном исследовательском центре "Информатики и управления" Российской Академии Наук (ФИЦ ИУ РАН). Организация находится по адресу г. Москва ул. Вавилова, 40. Это научная организация выполняющая фундаментальные, поисковые и прикладные научные исследования и разработки в области вычислительной и прикладной математики, системного анализа и управления, теоретической информатики и информационных технологий.

**Цель практики:**

Цель практики заключается в том, что надо разработать программу на языке Python которая преобразовывает набор входных файлов в формате LaTeX(.tex) в выходной формат XML.

**Задачи:**

* Описание грамматики входного языка (LaTeX)
* Разработка лексического анализатора
* Построение синтаксического анализатора
* Формирование абстрактного синтаксического дерева (AST).
* Генерация XML на основе AST.
* Дополнение грамматики до атрибутной

1. **Основная часть**

## **2.1 Анализ предметной области**

### 2.1.1 Выбор языка программирования и среды разработи

Выбор языка программирования и среды разработки играет ключевую роль решении задачи преоброзования входных файлов в выходной. Для решения этой задачи я выбрал Python как язык программирования Visual Studio Code в качестве среды разработки   
**Выбор языка, почему Python?**

Выбор пал на Python потому у него понятный и простой синтаксис, что делает его идеальным для быстрой разработки такой программы. Есть еще причины почему именно Python:

Например потому у Python есть большое количество библиотек которые могут быть использованы для решения заданной задачи

* Готовые инструменты для реализации лексический и синтаксических анализаторов - Lex-Yacc, ANTLR, PLY
* Для обработки текста - re, string
* Работы с XML - xml.etree.ElementTree, lxml

Так же Python работает на всех популярных операционных системах (Windows, macOS, Linux) что позволяет использовать один и тот же код без изменений на разных платформах. Это особенно ценно, если проект будет развиваться и использоваться в различных средах. Python поддерживает объектно-ориентированное программирование, что позволяет удобно моделировать узлы синтаксического дерева и работать с ними

**Почему Visual Studio Code?**

Visual Studio Code (VS Code) - это бесплатный редактор кода, разработанный компанией Microsoft. Он является одним из самых популярных инструментов для разработки программного обеспечения и поддерживает множество языков программирования такие как JavaScript, TypeScript, Python, Java, C++, и другие. VS Code интегрируется с Git и другими системами контроля версий, что позволяет разработчикам легко управлять своими репозиториями и контролировать версии своего кода. Он также интегрируется с различными сервисами, такими как Azure, Docker, AWS и другие, что упрощает развертывание и управление приложениями. В целом это мощный и гибкий редактор кода, который предоставляет множество функций для удобства разработки программного обеспечения. Он легко расширяемый и интегрируется с различными сервисами, что делает его одним из наиболее популярных редакторов кода.

### 2.1.2 Вывод

Выбранный стек технологий (Python + VS Code) обеспечивает оптимальный баланс между скоростью разработки, качеством решения и удобством сопровождения кода для данной задачи.

## 2.2 Разработка программы

### 2.2.1 Описание грамматики входного языка (LaTeX)

LaTeX — система подготовки документов и язык разметки. Она позволяет автоматизировать многие задачи набора текста, включая работу с математическими формулами, таблицами, изображениями и ссылками. Используется для подготовки научных и технических документов. Грамматика формального языка определяет правила построения строк в этом языке. Она состоит из:

* + Нетерминалов: Это абстрактные символы, которые представляют собой категории или конструкции языка, подлежащие дальнейшему разложению. Например, в LaTeX нетерминалы могут включать такие понятия, как <документ>, <команда> или <формула>. Они задают структуру документа на высоком уровне.
  + Терминалов: Это конечные символы или последовательности, которые непосредственно появляются в тексте документа и не подлежат дальнейшей замене. Примеры терминалов в LaTeX: команды (например, \textbf, \section), специальные символы (например, {, }), а также обычный текст или числа.
  + Правил вывода: правил, которые определяют, как можно заменять не терминалы на терминалы или другие не терминалы. Например <документ> ::= \documentclass{<класс>} <преамбула> \begin{document} <тело> \end{document}
  + Начального символа: В грамматике LaTeX начальным символом является <документ>, так как любой корректный LaTeX-документ начинается с описания его общей структуры, заданной командой \documentclass.

**Пример**

document        = preamble content\* ;

preamble        = command\* ;

content         = section | figure | text | bibliography | comment ;

command         = "\\", command\_name, [ arguments ] ;

command\_name    = letter+ | special\_char ;

arguments       = optional\_argument | mandatory\_argument ;

optional\_argument = "[", text, "]" ;

mandatory\_argument = "{", text, "}" ;

**Вывод**

LaTeX представляет собой мощный и гибкий язык разметки, предназначенный для создания высококачественных документов с точным контролем над их структурой и оформлением

Ключевые преимущества LaTeX:

- Четкая структура – документ организуется логично, с разделением на преамбулу и основное содержимое.

- Удобство работы с формулами – LaTeX является стандартом для набора математических выражений в научных публикациях.

- Автоматизация оформления – система сама управляет нумерацией, ссылками, оглавлением и списками.

- Кроссплатформенность – документы, написанные на LaTeX, могут быть скомпилированы в различных операционных системах без потери качества.

Его использование значительно упрощает процесс верстки сложных документов, обеспечивая единообразие и соответствие международным стандартам.

### 2.2.2 Разработка лексического анализатора

Лексический анализатор — это часть компилятора, которая читает исходную программу и выделяет в её тексте лексемы входного языка. Этапы обработки исходного кода, в нашем случае:

* Разбивает текст на токены(лексемы) — минимальные значимые элементы языка (ключевые слова, скобки, команды, текст).
* Отбрасывает незначимые символы (пробелы, комментарии, лишние переносы строк).
* Подготавливает данные для синтаксического анализатора(парсера)
* Обнаружение ошибок: Выявляет некорректные символы или структуры на раннем этапе.

**Зачем он нужен?**

Используется для структурирования входных данных

LaTeX - документ — это обычный текст. Лексер преобразует его в последовательность токенов, с которыми удобнее работать. Парсер работает не с сырым текстом, а с готовыми токенами, что ускоряет анализ и уменьшает количество ошибок.

**Основные методы лексера**

- advance(): Перемещает указатель на следующий символ, обновляя строку и столбец.

- get\_braced\_content(): Извлекает содержимое в фигурных скобках {...}, учитывая вложенность.

-clean\_text\_content(): Очищает текст от LaTeX-специфичных конструкций (например, ~, \-).

- get\_text(): Извлекает текстовый токен до специальных символов (\, {, }, [, ], %, $).

- get\_formula\_inline(), get\_formula\_block(): Обрабатывают математические формулы в $...$ или окружениях (equation, align, gather).

- get\_author(): Обрабатывает команды авторов (\author, \aut, \index), поддерживая индексы и переводы имён.

-get\_next\_token(): Основной метод, возвращающий следующий токен на основе текущего символа.

**Как он работает?**

Читает исходный код затем построчно или побайтово считывает файл. Разбивает на токены. Используют регулярные выражения или конечные автоматы для выделения лексем. Классифицирует токены

Каждый токен получает тип и значение:

* COMMAND: \section, \textbf
* LBRACE/RBRACE: {, }
* TEXT: произвольный текст внутри {...}
* COMMENT: всё после % до конца строки

Лексер передаёт парсеру поток токенов, который анализируется согласно грамматике LaTeX.

**Вывод:**

Лексический анализатор — ключевой этап в преобразовании LaTeX → XML, обеспечивающий корректность и скорость обработки.

**Пример вывода:** Найденные токены:

Token(TokenType.COMMAND", "\newcommand", line=1, col=1)

Token(TokenType.GROUP", "{}", line=1, col=12)

Token(TokenType.COMMAND", "\bs", line=1, col=13)

Token(TokenType.BRACES", "}", line=1, col=16)

Token(TokenType.GROUP", "{\boldsymbol{\sigma}}", line=1, col=17)

Token(TokenType.COMMAND", "\newcommand", line=2, col=1)

Token(TokenType.GROUP", "{}", line=2, col=12)

Token(TokenType.COMMAND", "\bb", line=2, col=13

### **2.2.3 Построение синтаксического анализатора**

Синтаксический анализатор (парсер) — часть программы, преобразующей входные данные (как правило, текст) в структурированный формат, нужный для задач последующего их анализа и использования. Это компонент компилятора или интерпретатора, который разбивает данные на более мелкие элементы для облегчения перевода на другой язык. Если подробнее то программа проверяет структуру входных данных на соответствие формальной грамматике

* Преобразует последовательность токенов (полученных от лексера) в абстрактное синтаксическое дерево (AST)
* Выявляет синтаксические ошибки(незакрытые скобки, неправильные команды)
* Если лексер работает с отдельными символами/лексемами, то парсер анализирует их взаимосвязь и иерархию.

**Зачем нужен парсер в LaTeX-конвертере?**

Парсер определяет, как команды, окружения и текст связаны между собой а также создает древовидное представление документа для последующей обработки. Подготавливает данные к преобразованию в XMl - AST — промежуточное представление для генерации XML.

**Основные методы парсера**

advance(): Перемещает указатель на следующий токен.

expect(): Проверяет, соответствует ли текущий токен ожидаемому типу, и вызывает ошибку при несоответствии.

parse(): Создаёт корневой узел Document и рекурсивно вызывает

parse\_statement для обработки всех токенов.

parse\_statement(): Обрабатывает отдельные токены, создавая узлы AST в зависимости от типа токена. Поддерживает текст, команды, секции, формулы, списки и т.д.

**Как он работает?**

* Получает поток токены от лексера
* Анализирует структуру согласно грамматике LaTeX
* Использует один из подходов:
* Рекурсивный спуск (подходит для LaTeX)
* Таблично-управляемый (LR-анализаторы)
* Комбинаторный (использует функциональный подход)
* Строит AST

**Вывод:** Без синтаксического анализатора невозможно корректное преобразование LaTeX в XML или другие форматы.

Что выводит:

└── Command (ID: 0f9e1eb2): \titel [Children: 1]

└── Group (ID: 74813781): {} [Children: 1]

└── Command (ID: 288a9fed): \tit [Children: 2]

└── Group (ID: a111f704): {} [Children: 1]

└── Command (ID: 5739303e): \aut [Children: 2]

└── Group (ID: 047dd2d7): {} [Children: 1]

└── Command (ID: 264dcc7d): \autkol [Children: 2]

└── Group (ID: 0d7d6c6c): {} [Children: 1]

## **2.2.4 Формирование абстрактного синтаксического дерева (AST).**

Абстрактное синтаксическое дерево (Abstract Syntax Tree, AST) — это древовидная структура данных, которая: Представляет иерархию и смысл исходного кода (в нашем случае — LaTeX-документа). В контексте LaTeX-документов AST служит "скелетом" документа, сохраняя его логическую структуру и предоставляя удобный формат для дальнейшего анализа, преобразования или генерации вывода.

AST – Состоит из:

* Узлов - Узлы AST отражают логические элементы LaTeX: Команды (\section, \textbf) Текстовые блоки (абзацы, строки). Окружения (itemize, equation). Математических выражений ($...$, \frac). - узел AST может выглядеть как Command(name="textbf", args=[Text("текст")]).
* Ребр - Рёбра (связи между узлами) показывают: Вложенность (например, содержимое окружения itemize → список узлов \item). Аргументы команд (например, \section{Введение} → аргумент Введение связан с узлом \section). Иерархию документа (преамбула → тело, разделы → подразделы).
* Абстрактность - означает отбрасывание несущественного синтаксиса. AST отбрасывает: Пробелы (кроме значимых, например, в \verb|текст с пробелами|). Комментарии (% ...).

**Зачем нужен AST?**

* Упрощение анализа: AST позволяет легко анализировать структуру документа, например, извлекать все заголовки или авторов.
* Трансформация: AST можно модифицировать для изменения документа (например, переформатирование или перевод).
* Генерация вывода: AST используется для генерации других форматов (PDF, HTML) или проверки корректности.

**Пример AST**

Абстрактное синтаксическое дерево (AST):

└── Document (ID: cea880d6): [empty] [Children: 68]

└── Definition (ID: 994ea263): \def\stat{} [Children: 0]

└── Text (ID: 104f4c57): zatsar [Children: 0]

└── Definition (ID: 56901023): \def\tit{} [Children: 0]

└── Text (ID: c558a530): СИСТЕМА СИТУАЦИОННОГО УПРАВЛЕНИЯ КАК МУЛЬТИСЕРВИСН... [Children: 0]

└── Definition (ID: af66b628): \def\titkol{} [Children: 0]

### **2.2.5 Генерация XML на основе AST.**

XML (eXtensible Markup Language) - это язык разметки, позволяющий описывать данные в виде структурированного текста, удо бного для обработки программами. XML поддерживает произвольные теги и атрибуты, что делает его универсальным средством представления данных  
**Что такое генерация XML на основе AST?**

Это процесс преобразования абстрактного синтаксического дерева , построенного из исходного текста (например, LaTeX), в формат XML . При этом:

* Каждый тип узла AST (например, section, itemize, formula) становится XML-элементом.
* Дочерние элементы AST становятся вложенными тегами.
* Атрибуты или значения узла могут быть представлены как атрибуты XML или содержимое

**Основные методы**

generate\_id(): Генерирует уникальные идентификаторы для элементов (например, id1, id2).

escape\_xml(): Экранирует специальные символы для корректного XML.

process\_node(): Рекурсивно обрабатывает узлы AST, создавая XML-элементы.

split\_latex\_content(): Разбивает текст на части, выделяя LaTeX-команды и специальные символы.

generate\_xml(): Формирует итоговый XML с отступами и сохраняет его в файл, если указано.

Пример вывода XML:

<papers>

<paper>

<metadata>

<authors>

<author value="John Doe"/>

</authors>

<title>

<text value="\title{My Paper}"/>

</title>

</metadata>

<fulltext>

<section id="id1">

<text>

<value value="\section{Introduction}"/>

</text>

<text>

<value value="This is a text."/>

</text>

<formula id="id2" value="\begin{equation}x = y + 1\end{equation}"/>

</section>

</fulltext>

<literat <papers>

<paper>

<metadata>

<authors>

<author value="John Doe"/>

</authors>

<title>

<text value="\title{My Paper}"/>

</title>

</metadata>

<fulltext>

<section id="id1">

<text>

<value value="\section{Introduction}"/>

</text>

<text>

<value value="This is a text."/>

</text>

<formula id="id2" value="\begin{equation}x = y + 1\end{equation}"/>

</section>

</fulltext>

<literature/>

</paper>

</papers>ure/>

</paper>

</papers>

## **2.3 Вспомогательные программы**

**В самом начале программы чтоб получить нужные файлы нам нужна программа для этого**

Цель: Подготовить LaTeX-файлы из RAR-архивов для дальнейшей обработки основным конвейером, обеспечив их корректную кодировку (UTF-8)Вспомогательная программа для разархивирования RAR-файлов и преобразования кодировки LaTeX-файлов обеспечивают подготовку данных для основного конвейера обработки LaTeX-документов. Они эффективно решают задачи извлечения .tex-файлов, их перекодировки в UTF-8 и добавления необходимых LaTeX-команд. Использование параллельной обработки и логирования делает программу надёжной и удобной для работы с большими наборами данных.

**В самом конце программы создан блок кода который автоматизирует обработку LaTeX - файлов**Эта программа дополняет ранее описанные компоненты (лексер, парсер, генератор XML)

Цель: Автоматизировать процесс преобразования LaTeX-файлов (.tex) из указанной директории в XML-формат, используя ранее разработанные лексер, парсер и генератор XML, с подробным логированием и отчётом об ошибках  
Программа предполагает наличие ранее разработанных компонентов:

Лексер (tokenize\_latex\_from\_file из файла лексере).

Парсер (parse\_latex из парсере).

Класс Node (определён в парсере для представления узлов AST).

А точнее код обрабатывает все .tex-файлы во входной директории, выполняя токенизацию, парсинг и генерацию XML.

# **Заключение**

В рамках практики была разработана программа на Python для преобразования LaTeX-файлов в XML-формат. Выполнены следующие задачи: Описание грамматики: Формализовано подмножество LaTeX, охватывающее основные конструкции научных статей, в виде BNF.

Лексический анализатор: Реализован лексер, который разбивает LaTeX-текст на токены, поддерживая обработку команд, формул, авторов и других элементов.

Синтаксический анализатор: Построен парсер, создающий AST на основе токенов с поддержкой иерархических конструкций.

Генерация XML: Реализован генератор XML, преобразующий AST в структурированный XML-документ

### **Список литературы:**

1.Ахо А. В. Компиляторы: принципы, технологии и инструменты / А. В. Ахо, М. С. Лам, Р. Сети, Дж. Д. Ульман ; пер. с англ. — М. : Вильямс, 2008. — 1184 с.

2. Компилятор за выходные: лексер и парсер // Хабр. — 2024. — URL: https://habr.com/ru/articles/787800/ (дата обращения: 10.06.2025).

3. LaTeX // Википедия: свободная энциклопедия. — URL: https://ru.wikipedia.org/wiki/LaTeX (дата обращения: 10.06.2025).  
4. LaTeX // Викиучебник. — URL: https://ru.wikibooks.org/wiki/LaTeX (дата обращения: 10.06.2025).

5. Extensible Markup Language (XML) 1.0 (Fifth Edition) // World Wide Web Consortium (W3C). — 2008. — URL: https://www.w3.org/TR/xml/ (дата обращения: 10.06.2025).