LaTeX to Markdown Converter

The goal of this project is to develop a LaTeX to Markdown converter from scratch, without relying on external libraries. The converter will take a LaTeX file as input and produce an equivalent Markdown file as output.

The converter should support the following features:

- **Sections and Subsections**: Convert LaTeX sectioning commands into corresponding Markdown headers.
- **Italics and Bold**: Translate LaTeX formatting for italics and bold text into Markdown syntax.
- **Horizontal Lines**: Convert LaTeX horizontal rules into Markdown horizontal lines.
- **Paragraphs**: Handle LaTeX paragraph breaks and translate them into Markdown paragraph separations.
- **Code Blocks**: Convert LaTeX verbatim environments into Markdown code blocks.
- **Hyperlinks**: Translate LaTeX hyperlinks into Markdown link syntax.
- **Images**: Convert LaTeX image inclusions into Markdown image syntax.
- **Lists**: Support both ordered and unordered lists in LaTeX, converting them into the corresponding Markdown list formats.
- **Tables**: Translate LaTeX tables into Markdown table syntax.

Design

In main.cpp, I start by setting up the main part of my program. I initialize the lexer and parser, then read the LaTeX file that I get from the command line. The file content goes to the parser, which converts it into an abstract syntax tree (AST). After that, I create the Markdown output from the AST and take care of any errors that come up.

For parser.y, I'm defining how my LaTeX input should be understood using Bison/Yacc. I write down the rules for LaTeX syntax and what should happen when these rules match, which usually means building and handling the AST. I also define the tokens the parser should recognize, and I include error handling to manage any syntax mistakes.

In lexer.l, I handle breaking down the LaTeX text into tokens. I write regular expressions to find things like commands, braces, and text. I also define what to do when a token is found so that the parser can process it properly.

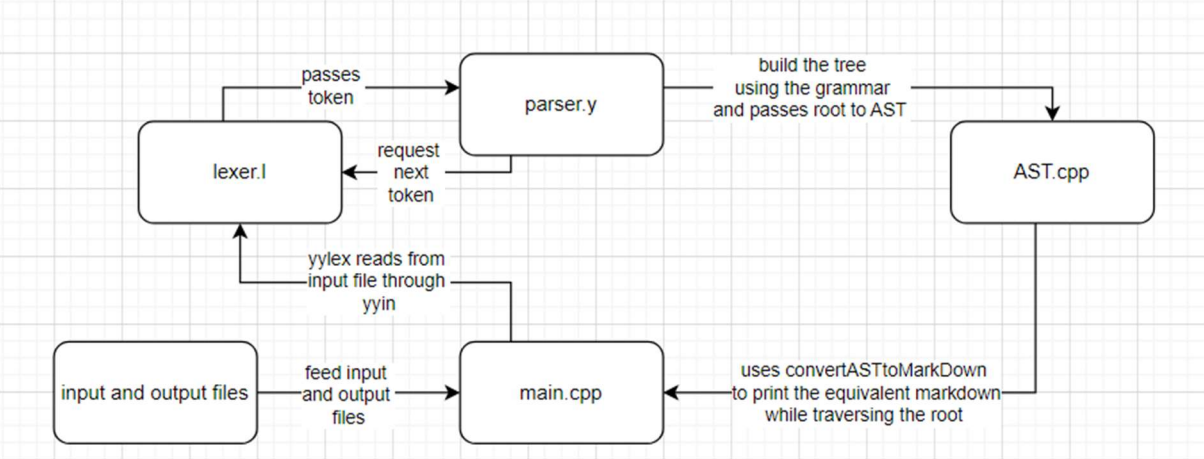
The AST.h file is where I define the structure of the abstract syntax tree. I set up the main class for AST nodes and create other classes for different LaTeX elements like headings and lists. This file includes the functions and data needed to work with the AST.

Finally, AST.cpp is where I write the code for the AST classes I declared in AST.h. I implement the functions and set up how each node type will interact with the rest of the program, such as

generating Markdown or navigating through the tree. This file ensures that the AST operations are done correctly and efficiently.

Working

In the program, main.cpp starts by setting up the lexer and parser, then reads the LaTeX file provided as a command-line argument. The lexer processes this file, breaking it down into tokens based on patterns defined in lexer.l. These tokens are sent to the parser, which uses the rules in parser.y to build an abstract syntax tree (AST) representing the structure of the LaTeX document. The AST is then used to generate the Markdown output. This involves translating the LaTeX elements into their Markdown equivalents. Any errors in the LaTeX input are managed by the parser. The AST's structure and operations are defined in AST.h and implemented in AST.cpp, which handles how the tree is built and manipulated. Overall, the program converts LaTeX files into Markdown by processing, parsing, and translating the document content.

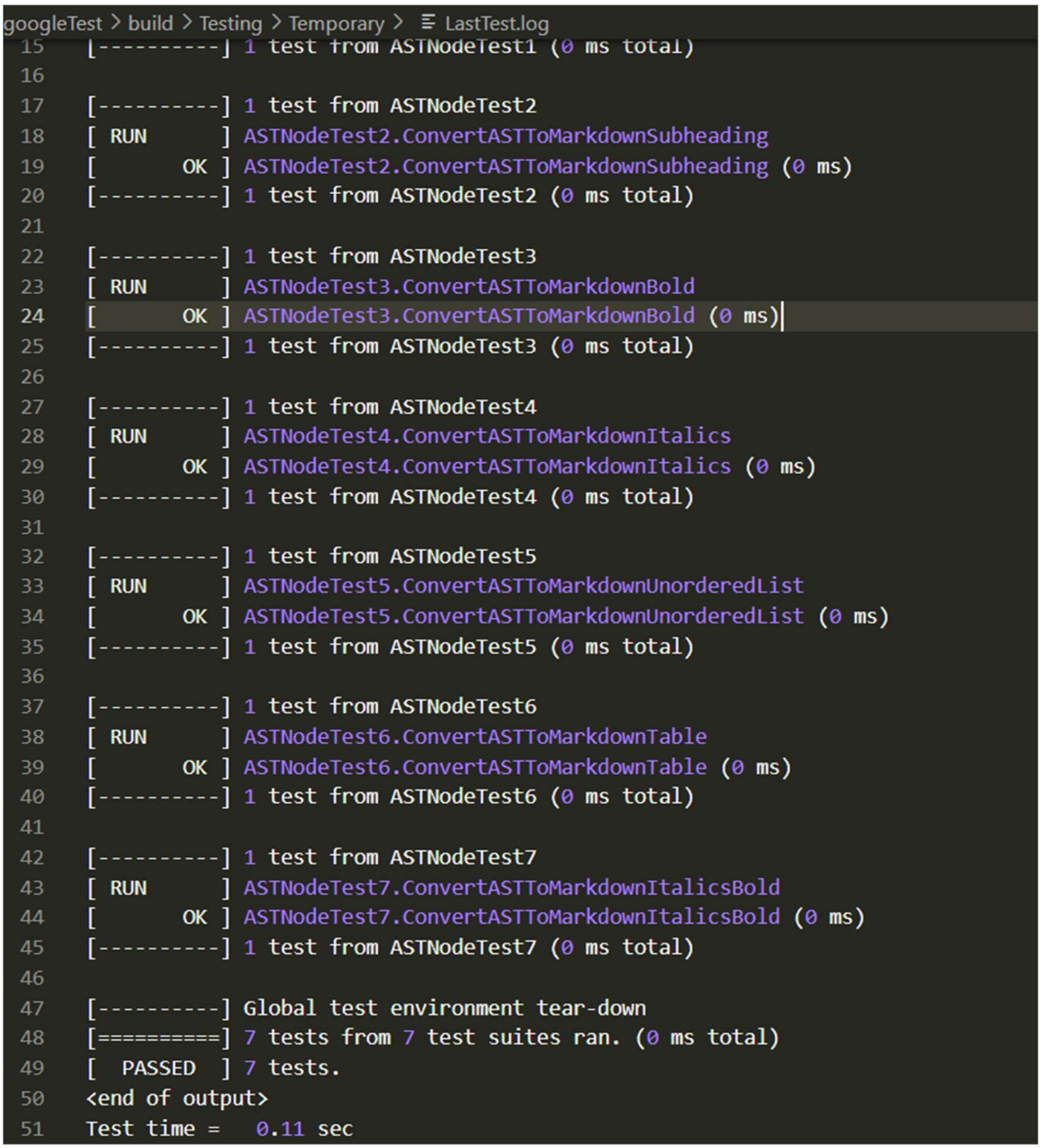


Pic -1 (The process)

Testing

In unit testing for the LaTeX to Markdown converter, I first set up Google Test in my project to handle the testing framework. I then created specific test cases to cover various components of the program. I tested the conversion process from AST to Markdown to ensure that the output was correct and met the expected Markdown format. Each test case was designed to validate a specific functionality and ensure that all parts of the program worked as intended. Finally, I built and ran the test executable to check if all tests passed, addressing any issues that were discovered during the testing process.

I also tried to test the main function nut for some reason it was giving me error while opening the input file and even if when I was able to open the input file still the program does not gets halts and stuck at showing that test cases are running, since I was not able to solve that error hence I didn't pushed on the GitHub.



Pic-2 (Image of the test Cases Performed)

Build and Usage

Build Instructions

The cmake tool was used to make the build process , To build and run the project, follow these steps:

First, use Git to copy the project files from the internet to the computer. we do this by running:

Clone the repository:

git clone https://github.com/yourusername/latexToMarkdownConverter.git

```
1. cd latexToMarkdownConverter
```

Create a build directory and run CMake:

Create a folder where the build files will be placed. Then, move into that folder. Use CMake to create the necessary files for building the project:

```
1. mkdir build
2. cd build
3. cmake ..
```

Compile the project:

Compile the code into an executable file by running.

```
1. make
2. cd ..
```

Run the converter:

```
1. ./run.sh <input_file.tex> <output_file.md>
```

The project folder includes:

- **/src Folder:** Contains the source code files, including main.cpp, AST.cpp, AST.h, lexer.l, and parser.y.
- **/build Folder:** Where the compiled files will be stored during the build process.
- **run.sh Script:** A script to run the converter with your files.
- **/images :** consist of images used in the input files.
- **/googleTest:** it consist of unit test cases on which this program was tested.