# COP701 Assignment:1

2023MCS2015

Generated by Doxygen 1.12.0

**1 LaTeX to Markdown Converter** 1
1.1 Overview . . . 1
1.2 Features . . . 1
1.3 Project Structure . . . 1
1.4 Dependencies . . . 1
1.5 Installation . . . 2
1.6 Make . . . 2
1.7 Usage . . . 2
1.8 Example Latex Code . . . 2

**2 Class Index** 3
2.1 Class List . . . 3

**3 File Index** 5
3.1 File List . . . 5

**4 Class Documentation** 7
4.1 ASTManager Class Reference . . . 7
4.1.1 Detailed Description . . . 7
4.1.2 Member Function Documentation . . . 7
4.1.2.1 newNode() . . . 7
4.1.2.2 print() . . . 8
4.2 ASTNode Class Reference . . . 8
4.2.1 Detailed Description . . . 8
4.2.2 Constructor & Destructor Documentation . . . 9
4.2.2.1 ASTNode() . . . 9
4.2.2.2 ∼ASTNode() . . . 9
4.2.3 Member Function Documentation . . . 9
4.2.3.1 addChild() . . . 9
4.3 converter Class Reference . . . 9
4.3.1 Detailed Description . . . 10
4.3.2 Constructor & Destructor Documentation . . . 11
4.3.2.1 converter() . . . 11
4.3.3 Member Function Documentation . . . 12
4.3.3.1 getMapping() . . . 12
4.3.3.2 printMarkdown() . . . 12
4.3.3.3 traversal() . . . 12
4.3.3.4 traverseChildren() . . . 13
4.3.3.5 traverseDate() . . . 13
4.3.3.6 traverseFigure() . . . 13
4.3.3.7 traverseFont() . . . 13
4.3.3.8 traverseHref() . . . 13
4.3.3.9 traverseList() . . . 14

4.3.3.10 traverseParagraph() . . . . . 14
4.3.3.11 traverseReference() . . . . . 14
4.3.3.12 traverseSection() . . . . . 14
4.3.3.13 traverseSubSection() . . . . . 14
4.3.3.14 traverseSubsubSection() . . . . . 15
4.3.3.15 traverseTable() . . . . . 15
4.3.3.16 traverseTitle() . . . . . 15
4.3.3.17 traverseVerbatim() . . . . . 15

**5 File Documentation** **17**

5.1 ast.h . . . . . 17
5.2 converter.h . . . . . 18

**Index** **21**

# Chapter 1

# LaTeX to Markdown Converter

## 1.1 Overview

This project is a C++ application that converts LaTeX documents into Markdown format. It uses Flex and Bison for lexical analysis and parsing. The parser processes various LaTeX constructs, such as sections, lists, tables, and more, and converts them into equivalent Markdown representations.

## 1.2 Features

- Convert LaTeX sections and subsections to Markdown headers.
- Handle ordered and unordered lists.
- Support for tables, figures, and verbatim text.
- Conversion of LaTeX formatting (bold, italic) to Markdown.
- Output Markdown to a file.

## 1.3 Project Structure

- `main.cpp`: The main entry point of the application.
- `ast.h` / `ast.cpp`: Defines and implements the Abstract Syntax Tree (AST) for LaTeX documents.
- `converter.h` / `converter.cpp`: Contains the logic for converting AST nodes into Markdown format.
- `parser.y` / `lexer.l`: Defines the Flex and Bison rules for lexical analysis and parsing LaTeX.
- `README.md`: This file, providing an overview and documentation of the project.

## 1.4 Dependencies

- Flex (Fast Lexical Analyzer Generator)
- Bison (GNU Parser Generator)
- C++11 or later (for certain features)

## 1.5 Installation

1. **Install Flex and Bison:**
   - On Debian-based systems: `sudo apt-get install flex bison`
   - On Red Hat-based systems: `sudo yum install flex bison`
2. **Clone the repository:**

```
git clone <repository_url>
cd <repository_directory>
```

## 1.6 Make

```
make
```

## 1.7 Usage

```
./compiler input.tex output.md
```

## 1.8 Example Latex Code

```
\section{Introduction}
This is a sample document.

\begin{itemize}
    \item Item 1
    \item Item 2
\end{itemize}

\begin{tabular}{|c|c|}
\hline
Header 1 & Header 2 \\
\hline
Data 1 & Data 2 \\
\hline
\end{tabular}
```

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:


ASTManager
: ASTManager class manages the AST, including creating and printing nodes . . . . . 7

ASTNode
: ASTNode class represents a node in the AST . . . . . 8

converter
: Converter class for traversing AST nodes and converting them to a Markdown-like format . . . 9

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:

ast.h . . . . . 17
converter.h . . . . . 18

# Chapter 4

# Class Documentation

## 4.1 ASTManager Class Reference

ASTManager class manages the AST, including creating and printing nodes.

`#include <ast.h>`

### Public Member Functions

- **ASTManager** ()

  *Constructors and Destructor.*
- ∼**ASTManager** ()

  *Default constructor.*
- ASTNode ∗ newNode (NodeType type)

  *Destructor.*
- ASTNode ∗ **newNode** ()

  *Create a new node with a specified type.*
- ASTNode ∗ **newNode** (const string &data)

  *Create a default node.*
- void print (ASTNode ∗root, int tabs=0) const

  *Create a new node with data.*

### 4.1.1 Detailed Description

ASTManager class manages the AST, including creating and printing nodes.

### 4.1.2 Member Function Documentation

#### 4.1.2.1 newNode()

```
ASTNode * ASTManager::newNode (
            NodeType type)
```

Destructor.

Node creation methods

#### 4.1.2.2 print()

```
void ASTManager::print (
            ASTNode * root,
            int tabs = 0) const
```

Create a new node with data.

Prints the AST starting from the root node

The documentation for this class was generated from the following files:

- ast.h
- ast.cpp

## 4.2 ASTNode Class Reference

ASTNode class represents a node in the AST.

```
#include <ast.h>
```

### Public Member Functions

- ASTNode ()

  *Child nodes.*
- **ASTNode** (NodeType type, const string &data="", const string &attributes="")

  *Default constructor.*
- ~ASTNode ()

  *Parameterized constructor.*
- void addChild (ASTNode *child)

  *Destructor to clean up children nodes.*
- void **print** (int tabs=0) const

  *Prints the node and its children with indentation based on the depth in the tree.*

### Public Attributes

- NodeType **node_type**
- string **data**

  *Type of the node (e.g., SECTION_H, ITEM_H)*
- string **attributes**

  *Data associated with the node (e.g., text content)*
- vector< ASTNode * > **children**

  *Additional attributes (e.g., label, reference)*

### 4.2.1 Detailed Description

ASTNode class represents a node in the AST.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 ASTNode()

```
ASTNode::ASTNode ()
```

Child nodes.

Constructors

#### 4.2.2.2 ~ASTNode()

```
ASTNode::~ASTNode ()
```

Parameterized constructor.

Destructor

### 4.2.3 Member Function Documentation

#### 4.2.3.1 addChild()

```
void ASTNode::addChild (
            ASTNode * child)
```

Destructor to clean up children nodes.

Adds a child node to the current node

The documentation for this class was generated from the following files:

- ast.h
- ast.cpp

## 4.3 converter Class Reference

Converter class for traversing AST nodes and converting them to a Markdown-like format.

```
#include <converter.h>
```

**Public Member Functions**

- converter ()

  *Mapping of node types to their string representations.*
- std::string traversal (ASTNode ∗root)

  *Traversal method for converting the entire AST starting from the root node.*
- std::string traverseSection (ASTNode ∗root, int type)

  *Traversal methods for different node types, based on their type.*
- std::string traverseSubSection (ASTNode ∗root, int type)

  *Handles SECTION nodes.*
- std::string traverseSubsubSection (ASTNode ∗root, int type)

  *Handles SUBSECTION nodes.*
- std::string traverseList (ASTNode ∗root, int type)

  *Handles SUBSUBSECTION nodes.*
- std::string traverseVerbatim (ASTNode ∗root, int type)

  *Handles LIST nodes (e.g., itemize, enumerate)*
- std::string traverseFont (ASTNode ∗root, int type)

  *Handles VERBATIM nodes (e.g., code blocks)*
- std::string traverseDate (ASTNode ∗root, int type)

  *Handles font formatting nodes (e.g., bold, italic)*
- std::string traverseTitle (ASTNode ∗root, int type)

  *Handles DATE nodes.*
- std::string traverseChildren (ASTNode ∗root)

  *Handles TITLE nodes.*
- std::string getMapping (int type)

  *Handles traversal of child nodes.*
- std::string traverseReference (ASTNode ∗root, int type)

  *Traversal methods for additional node types.*
- std::string **traverseLabel** (ASTNode ∗root, int type)

  *Handles REFERENCE nodes.*
- std::string traverseFigure (ASTNode ∗root, int type)

  *Handles LABEL nodes.*
- std::string traverseParagraph (ASTNode ∗root, int type)

  *Handles FIGURE nodes.*
- std::string **traverseString** (ASTNode ∗root, int type)

  *Handles PARAGRAPH nodes.*
- std::string traverseHref (ASTNode ∗root, int type)

  *Handles STRING nodes.*
- std::string traverseTable (ASTNode ∗root, int type)

  *Handles HREF (hyperlink) nodes.*
- void printMarkdown (const std::string &s, const std::string &filename)

  *Handles TABLE nodes (e.g., tabular environments)*

### 4.3.1 Detailed Description

Converter class for traversing AST nodes and converting them to a Markdown-like format.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 converter()

```
converter::converter ()
```

Mapping of node types to their string representations.

Constructor initializes the mapping of node types to their Markdown representations.

Constructor Section (Markdown heading level 2)

Subsection (Markdown heading level 3)

Subsubsection (Markdown heading level 4)

Unordered list (Markdown list item)

Ordered list (Markdown list item with number)

List item (Markdown list item) - for ordered lists

Bold text (Markdown bold)

Italic text (Markdown italic)

Underline text (HTML underline tag)

Paragraph (not directly used in Markdown)

Anchor label (not used in Markdown)

Anchor reference (not used in Markdown)

Table (not used directly in Markdown)

Image placeholder (Markdown image)

Image (Markdown image)

Figure caption (Markdown caption for images)

Generic string content

Date (Markdown date representation)

Title (Markdown heading level 1)

Verbatim text (Markdown code block)

Horizontal rule (Markdown horizontal rule)

Hyperlink (Markdown link placeholder)

### 4.3.3 Member Function Documentation

#### 4.3.3.1 getMapping()

```
std::string converter::getMapping (
            int type)
```

Handles traversal of child nodes.

Retrieves the string representation for a given node type from the mapping.

Retrieves the string representation for a given node type from the mapping

#### 4.3.3.2 printMarkdown()

```
void converter::printMarkdown (
            const std::string & s,
            const std::string & filename)
```

Handles TABLE nodes (e.g., tabular environments)

Writes the converted Markdown content to a specified file.

Outputs the converted Markdown content to a specified file

#### 4.3.3.3 traversal()

```
std::string converter::traversal (
            ASTNode * root)
```

Traversal method for converting the entire AST starting from the root node.

Converts the entire AST starting from the root node. Return empty string if root is null

Directly return item data

Handle section nodes

Handle subsection nodes

Handle subsubsection nodes

Handle list nodes

Handle verbatim nodes

Handle font formatting nodes

Handle title nodes

Handle date nodes

Handle figure nodes

Handle reference nodes

Handle horizontal rules

Handle paragraph nodes

Handle hyperlink nodes

Handle table nodes

Handle unknown or other node types

#### 4.3.3.4 traverseChildren()

```
std::string converter::traverseChildren (
            ASTNode * root)
```

Handles TITLE nodes.

Traverses and processes all child nodes.

#### 4.3.3.5 traverseDate()

```
std::string converter::traverseDate (
            ASTNode * root,
            int type)
```

Handles font formatting nodes (e.g., bold, italic)

Converts DATE nodes to Markdown format.

#### 4.3.3.6 traverseFigure()

```
std::string converter::traverseFigure (
            ASTNode * root,
            int type)
```

Handles LABEL nodes.

Converts FIGURE nodes to Markdown format.

#### 4.3.3.7 traverseFont()

```
std::string converter::traverseFont (
            ASTNode * root,
            int type)
```

Handles VERBATIM nodes (e.g., code blocks)

Converts font formatting nodes (e.g., bold, italic) to Markdown format.

#### 4.3.3.8 traverseHref()

```
std::string converter::traverseHref (
            ASTNode * root,
            int type)
```

Handles STRING nodes.

Converts HREF nodes (hyperlinks) to Markdown format.

#### 4.3.3.9 traverseList()

```
std::string converter::traverseList (
            ASTNode * root,
            int type)
```

Handles SUBSUBSECTION nodes.

Converts LIST nodes (either ITEMIZE or ENUMERATE) to Markdown format. Create indentation based on nesting level

Handle unordered list

Handle ordered list

#### 4.3.3.10 traverseParagraph()

```
std::string converter::traverseParagraph (
            ASTNode * root,
            int type)
```

Handles FIGURE nodes.

Converts PARAGRAPH nodes to Markdown format.

#### 4.3.3.11 traverseReference()

```
std::string converter::traverseReference (
            ASTNode * root,
            int type)
```

Traversal methods for additional node types.

Converts REFERENCE nodes to Markdown format.

#### 4.3.3.12 traverseSection()

```
std::string converter::traverseSection (
            ASTNode * root,
            int type)
```

Traversal methods for different node types, based on their type.

Converts a SECTION node to Markdown format.

#### 4.3.3.13 traverseSubSection()

```
std::string converter::traverseSubSection (
            ASTNode * root,
            int type)
```

Handles SECTION nodes.

Converts a SUBSECTION node to Markdown format.

#### 4.3.3.14 traverseSubsubSection()

```
std::string converter::traverseSubsubSection (
            ASTNode * root,
            int type)
```

Handles SUBSECTION nodes.

Converts a SUBSUBSECTION node to Markdown format.

#### 4.3.3.15 traverseTable()

```
std::string converter::traverseTable (
            ASTNode * root,
            int type)
```

Handles HREF (hyperlink) nodes.

Converts TABLE nodes to Markdown format. Iterate through all rows in the table

Iterate through all cells in the row

Extract cell data and add it to the row string

Add cell data to the row with formatting

Trim the last pipe character and add a newline after the row

Remove trailing " | "

Iterate through all cells in the row

Extract cell data and add it to the row string

Add cell data to the row with formatting

Trim the last pipe character and add a newline after the row

Adding formatting (e.g., column separators)

Add a separator line after the header (first row)

Create a separator line of dashes

Insert after the first row

#### 4.3.3.16 traverseTitle()

```
std::string converter::traverseTitle (
            ASTNode * root,
            int type)
```

Handles DATE nodes.

Converts TITLE nodes to Markdown format.

#### 4.3.3.17 traverseVerbatim()

```
std::string converter::traverseVerbatim (
            ASTNode * root,
            int type)
```

Handles LIST nodes (e.g., itemize, enumerate)

Converts VERBATIM nodes (code blocks) to Markdown format.

The documentation for this class was generated from the following files:

- converter.h
- converter.cpp

# Chapter 5

# File Documentation

## 5.1 ast.h

```
00001 #ifndef _AST_H
00002 #define _AST_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <vector>
00007 #include <stack>
00008 #include <map>
00009 #include <algorithm>
00010
00011 using namespace std;
00012
00014 enum NodeType {
00015     AST_H,
00016     DOCUMENT_H,
00017     SECTION_H,
00018     SUBSECTION_H,
00019     SUBSUBSECTION_H,
00020     TEXTBF_H,
00021     TEXTIT_H,
00022     UNDERLINE_H,
00023     STRING_H,
00024     ENUMERATE_H,
00025     ITEMIZE_H,
00026     ITEM_H,
00027     PAR_H,
00028     TABULAR_H,
00029     ROW_H,
00030     CELL_H,
00031     FIGURE_H,
00032     CAPTION_H,
00033     INCLUDE_GRAPHICS_H,
00034     LABEL_H,
00035     REF_H,
00036     HLINE_H,
00037     SQRT_H,
00038     TITLE_H,
00039     DATE_H,
00040     VERBATIM_H,
00041     HRULE_H,
00042     HREF_H,
00043     TEXT_H,
00044     CODE_H
00045 };
00046
00048 inline string nodeTypeToString(NodeType type) {
00049     switch (type) {
00050         case AST_H: return "AST_H";
00051         case SECTION_H: return "SECTION_H";
00052         case SUBSECTION_H: return "SUBSECTION_H";
00053         case ITEMIZE_H: return "ITEMIZE_H";
00054         case ENUMERATE_H: return "ENUMERATE_H";
00055         case ITEM_H: return "ITEM_H";
00056         case TEXTBF_H: return "TEXTBF_H";
00057         case TEXTIT_H: return "TEXTIT_H";
00058         case UNDERLINE_H: return "UNDERLINE_H";
00059         case PAR_H: return "PAR_H";
00060         case LABEL_H: return "LABEL_H";
```

```
00061         case REF_H: return "REF_H";
00062         case TABULAR_H: return "TABULAR_H";
00063         case FIGURE_H: return "FIGURE_H";
00064         case INCLUDE_GRAPHICS_H: return "INCLUDE_GRAPHICS_H";
00065         case CAPTION_H: return "CAPTION_H";
00066         case STRING_H: return "STRING_H";
00067         case DOCUMENT_H: return "DOCUMENT_H";
00068         case ROW_H: return "ROW_H";
00069         case CELL_H: return "CELL_H";
00070         case SQRT_H: return "SQRT_H";
00071         case HLINE_H: return "HLINE_H";
00072         case SUBSUBSECTION_H: return "SUBSUBSECTION_H";
00073         case TITLE_H: return "TITLE_H";
00074         case DATE_H: return "DATE_H";
00075         case VERBATIM_H: return "VERBATIM_H";
00076         case HRULE_H: return "HRULE_H";
00077         case HREF_H: return "HREF_H";
00078         case TEXT_H: return "TEXT_H";
00079         case CODE_H: return "CODE_H";
00080         default: return "UNKNOWN_NODE_TYPE";
00081     }
00082 }
00083
00085 class ASTNode {
00086 public:
00087     NodeType node_type;
00088     string data;
00089     string attributes;
00090     vector<ASTNode *> children;
00091
00093     ASTNode();
00094     ASTNode(NodeType type, const string &data = "", const string &attributes = "");
00095
00097     ~ASTNode();
00098
00100     void addChild(ASTNode *child);
00101
00103     void print(int tabs = 0) const;
00104 };
00105
00107 class ASTManager {
00108 public:
00110     ASTManager();
00111     ~ASTManager();
00112
00114     ASTNode* newNode(NodeType type);
00115     ASTNode* newNode();
00116     ASTNode* newNode(const string& data);
00117
00119     void print(ASTNode* root, int tabs = 0) const;
00120 };
00121
00123 extern ASTManager astManager;
00124
00125 #endif
```

## 5.2 converter.h

```
00001 #ifndef CONVERTER_H
00002 #define CONVERTER_H
00003
00004 #include "ast.h"
00005 #include <string>
00006 #include <map>
00007
00009 class converter {
00010 private:
00011     std::map<int, std::string> myMapping;
00012
00013 public:
00015     converter();
00016
00018     std::string traversal(ASTNode* root);
00019
00021     std::string traverseSection(ASTNode* root, int type);
00022     std::string traverseSubSection(ASTNode* root, int type);
00023     std::string traverseSubsubSection(ASTNode* root, int type);
00024     std::string traverseList(ASTNode* root, int type);
00025     std::string traverseVerbatim(ASTNode* root, int type);
00026     std::string traverseFont(ASTNode* root, int type);
00027     std::string traverseDate(ASTNode* root, int type);
00028     std::string traverseTitle(ASTNode* root, int type);
00029     std::string traverseChildren(ASTNode* root);
```

```
00030
00032     std::string getMapping(int type);
00033
00035     std::string traverseReference(ASTNode* root, int type);
00036     std::string traverseLabel(ASTNode* root, int type);
00037     std::string traverseFigure(ASTNode* root, int type);
00038     std::string traverseParagraph(ASTNode* root, int type);
00039     std::string traverseString(ASTNode* root, int type);
00040     std::string traverseHref(ASTNode* root, int type);
00041     std::string traverseTable(ASTNode* root, int type);
00042
00044     void printMarkdown(const std::string& s, const std::string& filename);
00045 };
00046
00047 #endif
```

# Index


~ASTNode
    ASTNode, 9

addChild
    ASTNode, 9
ASTManager, 7
    newNode, 7
    print, 7
ASTNode, 8
    ~ASTNode, 9
    addChild, 9
    ASTNode, 9

converter, 9
    converter, 11
    getMapping, 12
    printMarkdown, 12
    traversal, 12
    traverseChildren, 12
    traverseDate, 13
    traverseFigure, 13
    traverseFont, 13
    traverseHref, 13
    traverseList, 13
    traverseParagraph, 14
    traverseReference, 14
    traverseSection, 14
    traverseSubSection, 14
    traverseSubsubSection, 14
    traverseTable, 15
    traverseTitle, 15
    traverseVerbatim, 15

getMapping
    converter, 12

LaTeX to Markdown Converter, 1

newNode
    ASTManager, 7

print
    ASTManager, 7
printMarkdown
    converter, 12

traversal
    converter, 12
traverseChildren
    converter, 12
traverseDate
    converter, 13
traverseFigure
    converter, 13
traverseFont
    converter, 13
traverseHref
    converter, 13
traverseList
    converter, 13
traverseParagraph
    converter, 14
traverseReference
    converter, 14
traverseSection
    converter, 14
traverseSubSection
    converter, 14
traverseSubsubSection
    converter, 14
traverseTable
    converter, 15
traverseTitle
    converter, 15
traverseVerbatim
    converter, 15