Latex2Markdown

Generated by Doxygen 1.9.1

Chapter 1

Main Page

1.1 LaTeX-to-Markdown Converter

1.1.1 Overview

LaTeX-to-Markdown is a converter that converts LaTeX code into Markdown code. This project uses Flex and Bison to tokenize and parse LaTeX code, generating an abstract syntax tree (AST) that is parsed to get the required Markdown code.

1.1.2 Features Implemented

- Sections subsections and subsubsection
- Italics and bold (Nested bold and Italic)
- Horizontal line
- Paragraph
- Code blocks
- Hyperlink
- Images
- Ordered List
- Unordered List
- Tables
- Strike through
- Inline Math
- Display Math

1.1.3 Workflow

1. **Tokenization**: The LaTeX code is first tokenized using Flex. Different tokens are generated based on the input Latex code. These tokens are passed to the parser for further processing.
2. **Parsing**: The tokens which are passed by lexer are parsed using Bison. While parsing the tokens the AST is created subsequently. This AST contains all the neccessary information about the type of command and its value.
3. **Markdown Conversion**: The AST which is generated during the parsing of the tokens is parsed from top to bottom to generate the equivalent markdown code. The AST is printed in the seperate file.

1.1.4 Dependencies

- C++ (version GCC-6.3.0-1 or greater)
- Flex (flex 2.6.4 or greater)
- Bison (bison (GNU Bison) 3.8.2 or greater)
- Gtest
- Cmake

1.1.4.1 Build Instructions

1. **Clone the repository:**
```
git clone https://github.com/Shreyash0907/Latex-to-markdown.git
cd latex-to-markdown
```
2. **Build the project:**
```
mkdir build
cd build
cmake ..
cd ../
```
3. **Execute run.sh:**
```
./run.sh <input_file.tex> <output_file.md>
```

1.1.4.2 Gtest Instruction

1. **Open Directory:**
```
cd sampleTests
mkdir build
cd build
cmake ..
make
```
2. **Execute Gtest:**
```
./unitTests
```

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 Node Class Reference

Represents a structure of the node in the AST.

`#include <Node.hpp>`

Public Member Functions

- symbol getType ()

  Used to access the private vaiable value.
- void setValue (std::string ∗val)

  Used to set the value of the private variable val.
- std::string ∗ getValue ()

  Used as getter funtion for value variable.
- Node (symbol val)

  Constructor for the Node.
- void convert2Markdown ()

  To convert the Node's value to markdown.
- void printAST (Node ∗node, int depth)

  Prints the Abstract syntax tree depth wise.

Public Attributes

- std::vector< Node ∗ > productions

  Stores the children productions for a partiular node.
- std::vector< std::string > tstruct

  Stores the structure of the table columns.
- int depth

  In case of nested lists, the depth of the particular list is stored.
- int rownum

  Denotes the row number of the table. used when converting to markdown.

4.1.1 Detailed Description

Represents a structure of the node in the AST.

It has different variables and member function in them.

Definition at line 80 of file Node.hpp.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Node()

```
Node::Node (
            symbol val )  [inline]
```

Constructor for the Node.

Parameters

val	Value with which the Node is initialize.

Definition at line 120 of file Node.hpp.

```
: type(val) {};
```

4.1.3 Member Function Documentation

4.1.3.1 convert2Markdown()

```
void Node::convert2Markdown ( )
```

To convert the Node's value to markdown.

Prints the Abstract syntax tree depth wise. It converts the node value to markdown language. It takes the type of the node as the value and based on that the conversion is done.

Returns

Function doesn't return any value.

Definition at line 188 of file Node.cpp.

```
                              {

    switch (this->type)
    {

    case Start:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string(*(this->productions[0]->value)));
        break;

    case Program:
        if(static_cast<int>(this->productions.size()) == 2){
            this->productions[0]->convert2Markdown();
            this->productions[1]->convert2Markdown();
            this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
        }
        break;

    case Text:
        this->setValue(this->value);
        break;

    case Space:
        this->setValue(this->value);
        break;

    case Newline:
        this->setValue(this->value);
        break;

    case Lcurb:
        this->setValue(new std::string("{"));
        break;

    case Rcurb:
        this->setValue(new std::string("}"));
        break;

    case Lsqrb:
        this->setValue(new std::string("["));
        break;

    case Rsqrb:
        this->setValue(new std::string("]"));
        break;

    case Pipe:
        this->setValue(new std::string("|"));
        break;

    case Aper:
        this->setValue(new std::string("&"));
        break;

    case Bslash:
        this->setValue(new std::string("\\"));
        break;

    case Paragraph:
        this->setValue(new std::string("\n"));
        break;

    case Graphic:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string("![Image]("+ *(this->productions[0]->value) + ")" ));
        break;

    case Hrule:
        this->setValue(new std::string("\n----"));
        break;

    case Href:
        this->productions[0]->convert2Markdown();
        this->productions[1]->convert2Markdown();
        this->setValue(new std::string("[" + *(this->productions[1]->value) + "]" + "(" +
*(this->productions[0]->value) + ")" ));
        break;

    case Subsubsection:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string("### " + *(this->productions[0]->value)));
        break;

    case Subsection:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string("## " + *(this->productions[0]->value)));
        break;

    case Section:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string("# " + *(this->productions[0]->value)));
        break;

```

```
  case Bold:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("**" + *(this->productions[0]->value) + "**"));
      break;

  case Sout:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("~~" + *(this->productions[0]->value) + "~~"));
      break;

  case Imath:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("$" + *(this->productions[0]->value) + "$"));
      break;

  case Dmath:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("$$" + *(this->productions[0]->value) + "$$"));
      break;

  case Italic:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("*" + *(this->productions[0]->value) + "*"));
      break;

  case Url:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Ostatement:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Operationlist:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Blocks:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Empty:
      break;

  case List:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Codecontent:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
      break;

  case Code:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string("“`" + *(this->productions[0]->value) + "“`"));
      break;

  case Symbols:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Lsentences:
      if(static_cast<int>(this->productions.size()) == 2){
          this->productions[0]->convert2Markdown();
          this->productions[1]->convert2Markdown();
          this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
      }else{
          this->productions[0]->convert2Markdown();
          this->setValue(new std::string(*(this->productions[0]->value)));
      }
      break;

  case Ospaces:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[1]->value) +
*(this->productions[0]->value)));
```

```
      break;

  case Ospace:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Sentences:
      if(static_cast<int>(this->productions.size()) == 2){
          this->productions[0]->convert2Markdown();
          this->productions[1]->convert2Markdown();
          this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
      }else{
          this->productions[0]->convert2Markdown();
          this->setValue(new std::string(*(this->productions[0]->value)));
      }
      break;

  case Sentence:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Gdata:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Gsentences:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[1]->value) +
*(this->productions[0]->value)));
      break;

  case Gsentence:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Operations:
      this->productions[0]->convert2Markdown();
      this->setValue(new std::string(*(this->productions[0]->value)));
      break;

  case Oitem:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();
      if(this->productions[0]->getType() == Lsentences){
          for(int i = 1 ; i < this->depth ; i++){
              this->setValue(new std::string(*(this->value) + "    "));
          }
          this->setValue(new std::string(*(this->value) + "1." + *(this->productions[0]->value) +
*(this->productions[1]->value)));
      }else{
          this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
      }
      break;

  case Unoitem:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();
      if(this->productions[0]->getType() == Lsentences){
          for(int i = 1 ; i < this->depth ; i++){
              this->setValue(new std::string(*(this->value) + "    "));
          }
          this->setValue(new std::string(*(this->value) + "-" + *(this->productions[0]->value) +
*(this->productions[1]->value)));
      }else{
          this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
      }
      break;

  case Tcontent:
      this->productions[0]->convert2Markdown();
      this->productions[1]->convert2Markdown();

      if(this->productions[1]->getType() == Tlines){
          this->setValue(new std::string(*(this->productions[0]->value) + "| " +
*(this->productions[1]->value) + "|\n"));
          if(this->rownum == 1){
              std::string* temp = new std::string("");
              for(int i = this->tstruct.size()-1 ; i >= 0; i--){
                  if(this->tstruct[i] == "l"){
                        temp = new std::string(*temp + "| :--- ");
                    }else if(this->tstruct[i] == "r"){
                        temp = new std::string(*temp + "| ---:  ");
                    }else{
                        temp = new std::string(*temp + "| :---:  ");
                    }

                }
                temp = new std::string(*temp + "|\n");
                this->setValue(new std::string(*(this->value) + *temp));
            }
        }else{
             this->setValue(new std::string(*(this->productions[0]->value) ));
        }
        break;

    case Tlines:
        if(static_cast<int>(this->productions.size()) == 2){
            this->productions[0]->convert2Markdown();
            this->productions[1]->convert2Markdown();
            this->setValue(new std::string(*(this->productions[0]->value) +
   *(this->productions[1]->value)));

        }else{
            this->productions[0]->convert2Markdown();
            this->setValue(new std::string(*(this->productions[0]->value)));
        }
        break;

    case Tline:
        this->productions[0]->convert2Markdown();
        this->productions[1]->convert2Markdown();
        if(this->productions[0]->getType() == Text){
            this->setValue(new std::string(*(this->productions[0]->value) +
   *(this->productions[1]->value)));
        }else{
            this->setValue(new std::string(" | " + *(this->productions[1]->value)));
        }
        break;


    case Hline:
        this->setValue(new std::string(""));
        break;

    case Table:
        this->productions[0]->convert2Markdown();
        this->setValue(new std::string(*(this->productions[0]->value)));
        break;

    default:
        break;
    }
}
```

4.1.3.2 getType()

symbol Node::getType ( ) [inline]

Used to access the private vaiable value.

Returns

Returns the type of the Node.

Definition at line 103 of file Node.hpp.

```
                        {
            return type;
        }
```

4.1.3.3 getValue()

```
std::string* Node::getValue ( )  [inline]
```

Used as getter funtion for value variable.

Returns

Returns the Node's value

Definition at line 115 of file Node.hpp.

```
                             {
            return value;
        }
```

4.1.3.4 printAST()

```
void Node::printAST (
             Node * node,
             int depth )
```

Prints the Abstract syntax tree depth wise.

Parameters

node	The node which type will be printed.
depth	The depth at which the current node is present.

Definition at line 504 of file Node.cpp.

```
                                         {
    if(node->productions.size() == 2){
        for(int i = 0 ; i < depth; i++){
            std::cout«" ";
        }
        std::cout«getEnumValue(node->getType())«"\n";
        printAST(node->productions[0], depth + 1);
        printAST(node->productions[1], depth + 1);
    }else if(node->productions.size() == 1){
        for(int i = 0 ; i < depth; i++){
            std::cout«" ";
        }
        std::cout«getEnumValue(node->getType())«"\n";
        printAST(node->productions[0], depth + 1);
    }
}
```

4.1.3.5 setValue()

```
void Node::setValue (
             std::string * val )  [inline]
```

Used to set the value of the private variable val.

Parameters

val	value to be store as the Node's value

Definition at line 109 of file Node.hpp.

```
109                             {
110             value = val;
111         }
```

4.1.4 Member Data Documentation

4.1.4.1 depth

`int Node::depth`

In case of nested lists, the depth of the particular list is stored.

Definition at line 96 of file Node.hpp.

4.1.4.2 productions

`std::vector< Node* > Node::productions`

Stores the children productions for a partiular node.

Definition at line 90 of file Node.hpp.

4.1.4.3 rownum

`int Node::rownum`

Denotes the row number of the table. used when converting to markdown.

Definition at line 99 of file Node.hpp.

4.1.4.4 tstruct

`std::vector<std::string> Node::tstruct`

Stores the structure of the table columns.

Definition at line 93 of file Node.hpp.

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

- /mnt/d/Mtech/COP701/Latex-to-markdown/Node.hpp
- /mnt/d/Mtech/COP701/Latex-to-markdown/Node.cpp

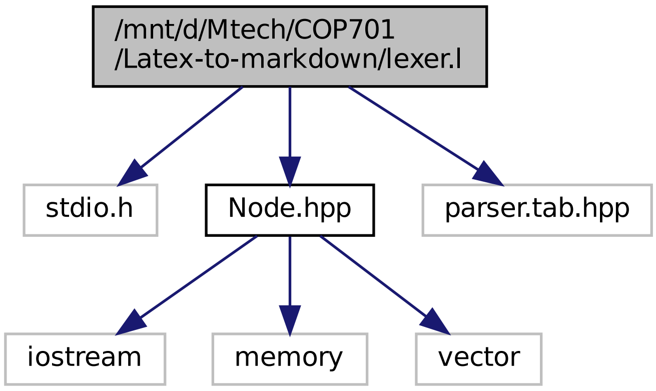
Chapter 5

File Documentation

5.1 /mnt/d/Mtech/COP701/Latex-to-markdown/lexer.l File Reference

Flex file for generation of tokens.

```
#include <stdio.h>
#include "Node.hpp"
#include "parser.tab.hpp"
```
Include dependency graph for lexer.l:



Functions

- <=> _ WORD (({ALPHABET}|{NUMERIC})+({ALPHABET}|{NUMERIC}) ∗) LCURB \

  Token for word.
- int yywrap ()

Variables

- APER &LSQRB[RSQRB \] PIPE DOLLAR NEWLINE n TAB t ITALIC textit BOLD textbf PARAGRAPH par SECTION section SUBSECTION subsection SUBSUBSECTION subsubsection HREF href HRULE hrule HLINE hline GRAPHIC includegraphics BEGIN begin END end DOCUMENT document TABLE tabular UNOLIST itemize OLIST enumerate CBLOCK verbatim ITEM item SOUT sout {BSLASH}{ITALIC} {return ITALIC; }

    Token for Amparsand.

5.1.1 Detailed Description

Flex file for generation of tokens.

This file contains the lexer rules and actions for tokenizing the latex code.

5.1.2 Function Documentation

5.1.2.1 WORD()

```
<=> _ WORD (
               ({ALPHABET}|{NUMERIC})+({ALPHABET}|{NUMERIC}) *  )
```

Token for word.

Token for text.

Token for left curly bracket. Token for right curly bracket.

Definition at line 49 of file lexer.l.

```
                         {ALPHABET}|{NUMERIC})+({ALPHABET}|{NUMERIC})*)

TEXT                   (({WORD}|{PUNCTUATION})+)


LCURB                  \{

RCURB                  \}
```

5.1.2.2 yywrap()

```
int yywrap ( )
```

Definition at line 258 of file lexer.l.

```
               {
    return 1;
}
```

5.1.3 Variable Documentation

5.1.3.1 sout

```
APER& LSQRB [RSQRB \] PIPE DOLLAR NEWLINE n TAB t ITALIC textit BOLD textbf PARAGRAPH par
SECTION section SUBSECTION subsection SUBSUBSECTION subsubsection HREF href HRULE hrule HLINE
hline GRAPHIC includegraphics BEGIN begin END end DOCUMENT document TABLE tabular UNOLIST
itemize OLIST enumerate CBLOCK verbatim ITEM item SOUT sout {BSLASH}{ITALIC} {return ITALIC;
}
```

Token for Amparsand.

Token for left square bracket.

Token for pipe.

Token for dollar.

Token for newlline.

Token for tab.

Token for italic.

Token for bold.

Token for paragraph.

Token for section.

Token for subsection.

Token for subsubsection.

Token for href.

Token for hrule.

Token for hline.

Token for graphic.

Token for operations.

Token for begin.

Token for end.

Token for document.

Token for tabular.

Token for itemize.

Token for enumerate.

Token for code block.

Token for item.

Token for strike through.

Token for block type.

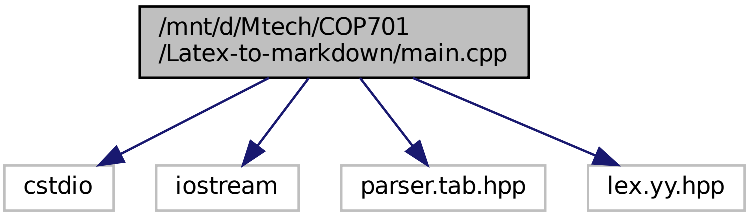
Definition at line 199 of file lexer.l.

5.2 /mnt/d/Mtech/COP701/Latex-to-markdown/main.cpp File Reference

Main function in the project.

```
#include <cstdio>
#include <iostream>
#include "parser.tab.hpp"
#include "lex.yy.hpp"
```

Include dependency graph for main.cpp:



Functions

- int yylex (void)

  Function used to create the token from input file.
- int yyparse (void)

  Function used to parse the input token and creates the AST.
- Node ∗ getRoot ()

  Function returns the root Node of the AST after the completion of parsing.
- int main (int argc, char ∗argv[ ])

  Main function in the project.

Variables

- int yydebug

  Used to debug the parsing process.

5.2.1 Detailed Description

Main function in the project.

Author

Shreyash Chikte

Version

0.1

Date

2024-08-24

Copyright

Copyright (c) 2024

5.2.2 Function Documentation

5.2.2.1 getRoot()

```
Node * getRoot ( )
```

Function returns the root Node of the AST after the completion of parsing.

Get the Root object of the AST.

Returns

Object of Node.

Node∗

Function returns the root Node of the AST after the completion of parsing.

Returns

Node∗

Definition at line 830 of file parser.y.

```
                {
        return root;
}
```

5.2.2.2 main()

```
int main (
             int argc,
             char * argv[] )
```

Main function in the project.

Parameters

argc	Denotes the count of the arguments.
argv	Stores the arguments in the array.

Returns

Returns integer value.

This denotes the input File name

Denotes the Output file name

Definition at line 34 of file main.cpp.

```
                                    {
       // yydebug = 1;
       if (argc < 2) {
           std::cerr « "Usage:  " « argv[0] « " <filename>" « std::endl;
           return 1;
       }

       const char* inputFile = argv[1];
       const char* outputFile = argv[2];

       FILE *inputPtr, *outputPtr;

       inputPtr = fopen(inputFile, "r");
       if (inputPtr == NULL) {
           cout«"Error:  Unable to open latex File\n.";
           return 0;
       }

       outputPtr = fopen(outputFile, "w");

       if (outputPtr == NULL) {
           cout«"Error:  Unable to open Markdown File\n.";
           return 0;
       }

       extern FILE *yyin;

       yyin = inputPtr;


       yyparse();

       Node* temp = getRoot();
       temp->convert2Markdown();

       fprintf(outputPtr,"%s\n", temp->getValue()->c_str());

       temp->printAST(temp, 0);

       fclose(inputPtr);
       fclose(outputPtr);
       return 0;
   }
```

5.2.2.3 yylex()

```
int yylex (
              void  )
```

Function used to create the token from input file.

5.2.2.4 yyparse()

```
int yyparse (
            void  )
```

Function used to parse the input token and creates the AST.

5.2.3 Variable Documentation

5.2.3.1 yydebug

```
int yydebug  [extern]
```

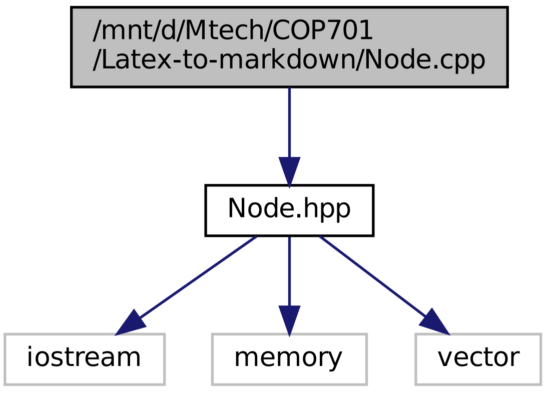
Used to debug the parsing process.

5.3 /mnt/d/Mtech/COP701/Latex-to-markdown/Node.cpp File Reference

Funcion implementation for class Node.

```
#include "Node.hpp"
```

Include dependency graph for Node.cpp:



Functions

- std::string getEnumValue (symbol type)

  Prints the Abstract syntax tree depth wise.

5.3.1 Detailed Description

Funcion implementation for class Node.

Author

Shreyash Chikte (shreyashsc9@gmail.com)

Version

0.1

Date

2024-08-24

Copyright

Copyright (c) 2024

5.3.2 Function Documentation

5.3.2.1 getEnumValue()

```
std::string getEnumValue (
            symbol type )
```

Prints the Abstract syntax tree depth wise.

Parameters

type	The type of the node is passed.

Returns

Function doesn't return any value.

Definition at line 20 of file Node.cpp.

```
20                                         {
21     switch(type){
22 
23         case Operations :
24             return "Operation_Node";
25             break;
26         case Unoitem:
27             return "Unoitem_Node";
28             break;
29         case Gsentence:
30             return "Gsentence_Node";
31             break;
32         case Gsentences:
```

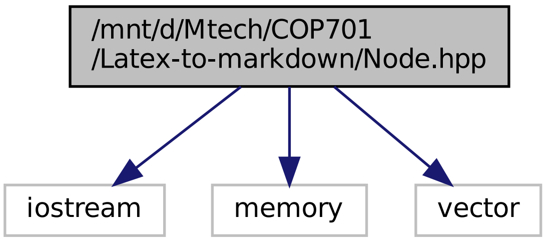
```
            return "Gsentences_Node";
            break;
        case Oitem:
            return "Oitem_Node";
            break;
        case Gdata:
            return "Gdata_Node";
            break;
        case Sentence:
            return "Sentence_Node";
            break;
        case Sentences:
            return "Sentences_Node";
            break;
        case Ospace:
            return "Ospace_Node";
            break;
        case Ospaces:
            return "Ospaces_Node";
            break;
        case Tcontent:
            return "Tcontent_Node";
            break;
        case Tstructure:
            return "Tstructure_Node";
            break;
        case Tline:
            return "Tline_Node";
            break;
        case Tlines:
            return "Tlines_Node";
            break;
        case Lsentences:
            return "Lsentences_Node";
            break;
        case Symbols:
            return "Symbold_Node";
            break;
        case List:
            return "List_Node";
            break;
        case Codecontent:
            return "Codecontent_Node";
            break;
        case Start:
            return "Start_Node";
            break;
        case Program:
            return "Program_Node";
            break;
        case Blocks:
            return "Blocks_Node";
            break;
        case Operationlist:
            return "Operationlist_Node";
            break;
        case Ostatement:
            return "Ostatement_Node";
            break;
        case Table:
            return "Table_Node";
            break;
        case Thead:
            return "Thead";
            break;
        case Url:
            return "Url_Node";
            break;
        case Text:
            return "Text_Node";
            break;
        case Space:
            return "Space_Node";
            break;
        case Newline:
            return "Newline_Node";
            break;
        case Italic:
            return "Italic_Node";
            break;
        case Bold:
            return "Bold_Node";
            break;
        case Section:
            return "Section_Node";
            break;
        case Subsection:
            return "Subsection_Node";
            break;
        case Subsubsection:
            return "Subsubsection_Node";
            break;
        case Href:
            return "Href_Node";
            break;
        case Hrule:
            return "Hrule_Node";
            break;
        case Graphic:
            return "Graphic_Node";
            break;
        case Paragraph:
            return "Paragraph_Node";
            break;
        case Lcurb:
            return "Left curly bracket Node";
            break;
        case Rcurb:
            return "Right curly bracket Node";
            break;
        case Lsqrb:
            return "Left square bracket Node";
            break;
        case Rsqrb:
            return "Right square bracket Node";
            break;
        case Aper:
            return "Ampersand_Node";
            break;
        case Pipe:
            return "Pipe_Node";
            break;
        case Bslash:
            return "Back Slash Node";
            break;
        case Code:
            return "Code_Node";
            break;
        case Empty:
            return "Empty_Node";
            break;
        case Hline:
            return "Hline_Node";
            break;
        case Dmath:
            return "Display Math Node";
            break;
        case Sout:
            return "Strike Through Node";
            break;
        case Imath:
            return "Inline math Node";
            break;
        default:
            return "Unexpected_Node";
            break;
    }
}
```

5.4 /mnt/d/Mtech/COP701/Latex-to-markdown/Node.hpp File Reference

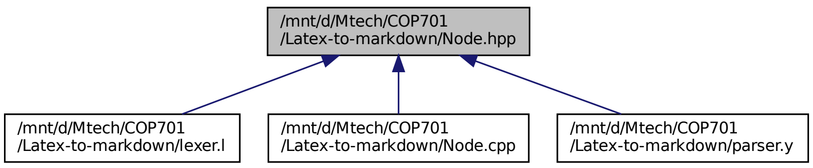
Class defination for the structure of the Node in AST.

```
#include <iostream>
#include <memory>
#include <vector>
```

Include dependency graph for Node.hpp:



This graph shows which files directly or indirectly include this file:


/mnt/d/Mtech/COP701
/Latex-to-markdown/Node.hpp
/mnt/d/Mtech/COP701
/Latex-to-markdown/lexer.l
/mnt/d/Mtech/COP701
/Latex-to-markdown/Node.cpp
/mnt/d/Mtech/COP701
/Latex-to-markdown/parser.y

Classes

- class Node

  Represents a structure of the node in the AST.

Enumerations

- enum symbol {
  Operations , Unoitem , Gsentence , Gsentences ,
  Oitem , Gdata , Sentence , Sentences ,
  Ospace , Ospaces , Tcontent , Tstructure ,
  Tline , Tlines , Lsentences , Symbols ,
  List , Codecontent , Start , Program ,
  Blocks , Operationlist , Ostatement , Table ,
  Thead , Url , Text , Space ,
  Newline , Italic , Bold , Section ,
  Subsection , Subsubsection , Href , Hrule ,
  Graphic , Paragraph , Lcurb , Rcurb ,
  Lsqrb , Rsqrb , Aper , Pipe ,
  Bslash , Code , Empty , Hline ,
  Dmath , Sout , Imath }

  Denotes the type of the node in AST.

5.4.1 Detailed Description

Class defination for the structure of the Node in AST.

Author

Shreyash Chikte (`shreyashsc9@gmail.com`)

Defines the enum for the type of node in AST

Version

0.1

Date

2024-08-24

Copyright

Copyright (c) 2024

5.4.2 Enumeration Type Documentation

5.4.2.1 symbol

```
enum symbol
```

Denotes the type of the node in AST.

Enumerator

Operations	
Unoitem	
Gsentence	
Gsentences	
Oitem	
Gdata	
Sentence	
Sentences	
Ospace	
Ospaces	
Tcontent	
Tstructure	
Tline	
Tlines	
Lsentences	
Symbols	

Enumerator

List	
Codecontent	
Start	
Program	
Blocks	
Operationlist	
Ostatement	
Table	
Thead	
Url	
Text	
Space	
Newline	
Italic	
Bold	
Section	
Subsection	
Subsubsection	
Href	
Hrule	
Graphic	
Paragraph	
Lcurb	
Rcurb	
Lsqrb	
Rsqrb	
Aper	
Pipe	
Bslash	
Code	
Empty	
Hline	
Dmath	
Sout	
Imath	

Definition at line 21 of file Node.hpp.

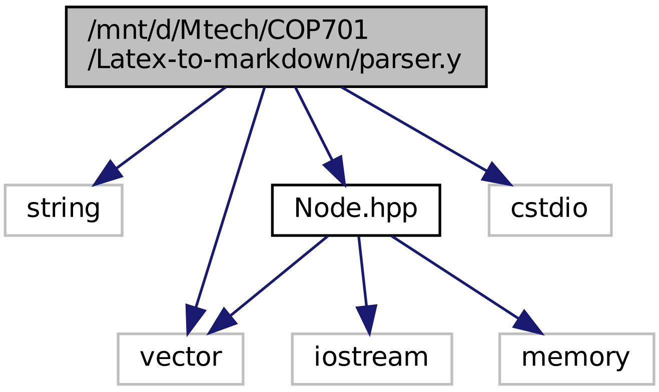
```
           {
    Operations,
    Unoitem,
    Gsentence,
    Gsentences,
    Oitem,
    Gdata,
    Sentence,
    Sentences,
    Ospace,
    Ospaces,
    Tcontent,
    Tstructure,
    Tline,
    Tlines,
    Lsentences,
    Symbols,
    List,
    Codecontent,
    Start,
    Program,
    Blocks,
    Operationlist,
    Ostatement,
    Table,
    Thead,
    Url,
    Text,
    Space,
    Newline,
    Italic,
    Bold,
    Section,
    Subsection,
    Subsubsection,
    Href,
    Hrule,
    Graphic,
    Paragraph,
    Lcurb,
    Rcurb,
    Lsqrb,
    Rsqrb,
    Aper,
    Pipe,
    Bslash,
    Code,
    Empty,
    Hline,
    Dmath,
    Sout,
    Imath,
};
```

5.5 /mnt/d/Mtech/COP701/Latex-to-markdown/parser.y File Reference

Bison file for parsing Latex code.

```
#include <string>
#include <vector>
#include "Node.hpp"
#include <cstdio>
```

Include dependency graph for parser.y:



Functions

- int yylex (void)
- Node ∗ getRoot ()

  Get the Root object of the AST.
- root setValue (new std::string("")) = new Node(Tcontent)
- root productions push_back ($3)
- tstr clear ()
- productions push_back (temp)
- setValue (new std::string("operations"))
- temp setValue (new std::string("|"))
- temp setValue (new std::string(∗$1))
- tstr push_back (st)
- temp setValue (new std::string("&"))
- temp setValue (new std::string("\n"))

Variables

- debug
- Node ∗ root
- int cnt = 0
- int tcol = 0
- int trow = 0
- std::vector< std::string > tstr
- code requires
- union {
  std::string ∗ str
  Node ∗ node
  } start

  Union to define the data types of the tokens.
- BDOC NEWLINE program EDOC
- program __pad0__

  Parses simple operations and block operations.
- operationList program
- blocks __pad1__

  parses the blocks operations in code
- BTABLE table ETABLE
- BCBLOCK codecontent ECBLOCK
- Node ∗ temp = new Node(Code)
- DMOPEN sentences DMEND
- table __pad2__

  parses the Table component in the code
- thead __pad3__

  parses heading component of the Table
- list __pad4__

  parses the list component
- BOLIST depthI oitem EOLIST depthD NEWLINE
- depthD __pad5__

  Increases the depth of list in case of nested lists.
- depthI __pad6__

  Decreases the depth of list in case of nested lists.
- operationList __pad7__

Parses the simple operations.
- operations __pad8__

  Parses various operations.
- ITALIC ostatement
- HRULE
- PARAGRAPH
- IMATH sentences IMATH
- url __pad9__

  parses the url in href component
- ostatement __pad10__

  parses the nested simple operations
- unoitem __pad11__

  parses List items of unordered list
- ospaces ITEM lsentences unoitem
- depth = cnt
- oitem __pad12__

  parses List items of ordered list
- ospaces ITEM lsentences oitem
- tstructure __pad13__

  parses table structure
- PIPE tstructure
- std::string st = ∗(temp->getValue())
- tcontent __pad14__

  parses data content of the table
- tcontent tlines BSLASH BSLASH ospaces
- rownum = trow
- tstruct = tstr
- tlines __pad15__

  parses multiple table rows
- tline tlines
- tline __pad16__

  parses individual table rows
- gdata __pad17__

  parses graphics data
- codecontent __pad18__

  parses code present inside the verbatim block
- sentences codecontent
- symbols __pad19__

  parses different type of symbols in verbatim blocks
- RCURB
- LSQRB
- RSQRB
- APER
- PIPE
- BSLASH
- startingtext __pad20__

  parses included libraries
- sentence sentences
- ospace __pad21__

  parses optional spaces
- ospaces __pad22__

  parses multiple optional spaces

- ospaces ospace
- gsentences __pad23__

  parses multiple graphics modifying data
- gsentence __pad24__

  parses grpahics modifying data
- sentence __pad25__

  parses text and optional spaces
- lsentences __pad26__

  parses List items texts
- SPACE lsentences
- optspace __pad27__

  parses optional spaces for some specific cases
- SPACE

5.5.1 Detailed Description

Bison file for parsing Latex code.

This file contains the grammer rules and semantics for creating the AST.

5.5.2 Function Documentation

5.5.2.1 clear()

```
tstr clear ( )
```

5.5.2.2 getRoot()

```
Node* getRoot ( )
```

Get the Root object of the AST.

Function returns the root Node of the AST after the completion of parsing.

Returns

Node∗

Definition at line 830 of file parser.y.

```
830                {
831         return root;
832 }
```

5.5.2.3 push_back() [1/3]

```
productions push_back (
             $3 )
```

5.5.2.4 push_back() [2/3]

```
tstr push_back (
             st  )
```

5.5.2.5 push_back() [3/3]

```
productions push_back (
             temp  )
```

5.5.2.6 setValue() [1/6]

```
setValue (
             new  std::string"" ) = new Node(Tcontent)
```

5.5.2.7 setValue() [2/6]

```
temp setValue (
             new  std::string"&" )
```

5.5.2.8 setValue() [3/6]

```
temp setValue (
             new  std::string"\n" )
```

5.5.2.9 setValue() [4/6]

```
setValue (
             new  std::string"operations" )
```

5.5.2.10 setValue() [5/6]

```
temp setValue (
             new  std::string"|" )
```

5.5.2.11 setValue() [6/6]

```
temp setValue (
             new  std::string* $1 )
```

5.5.2.12 yylex()

```
int yylex (
             void  )
```

5.5.3 Variable Documentation

5.5.3.1 __pad0__

```
program __pad0__
```

Parses simple operations and block operations.

Definition at line 87 of file parser.y.

5.5.3.2 __pad10__

```
ostatement __pad10__
```

parses the nested simple operations

Definition at line 359 of file parser.y.

5.5.3.3 __pad11__

`unoitem __pad11__`

parses List items of unordered list

Definition at line 370 of file parser.y.

5.5.3.4 __pad12__

`oitem __pad12__`

parses List items of ordered list

Definition at line 395 of file parser.y.

5.5.3.5 __pad13__

`tstructure __pad13__`

parses table structure

Definition at line 420 of file parser.y.

5.5.3.6 __pad14__

`tcontent __pad14__`

parses data content of the table

Definition at line 455 of file parser.y.

5.5.3.7 __pad15__

`tlines __pad15__`

parses multiple table rows

Definition at line 485 of file parser.y.

5.5.3.8 __pad16__

```
tline __pad16__
```

parses individual table rows

Definition at line 504 of file parser.y.

5.5.3.9 __pad17__

```
gdata __pad17__
```

parses graphics data

Definition at line 533 of file parser.y.

5.5.3.10 __pad18__

```
codecontent __pad18__
```

parses code present inside the verbatim block

Definition at line 545 of file parser.y.

5.5.3.11 __pad19__

```
symbols __pad19__
```

parses different type of symbols in verbatim blocks

Definition at line 571 of file parser.y.

5.5.3.12 __pad1__

```
blocks __pad1__
```

parses the blocks operations in code

Definition at line 111 of file parser.y.

5.5.3.13 __pad20__

```
startingtext __pad20__
```

parses included libraries

Definition at line 653 of file parser.y.

5.5.3.14 __pad21__

```
ospace __pad21__
```

parses optional spaces

Definition at line 681 of file parser.y.

5.5.3.15 __pad22__

```
ospaces __pad22__
```

parses multiple optional spaces

Definition at line 707 of file parser.y.

5.5.3.16 __pad23__

```
gsentences __pad23__
```

parses multiple graphics modifying data

Definition at line 724 of file parser.y.

5.5.3.17 __pad24__

```
gsentence __pad24__
```

parses grpahics modifying data

Definition at line 738 of file parser.y.

5.5.3.18 __pad25__

```
sentence __pad25__
```

parses text and optional spaces

Definition at line 750 of file parser.y.

5.5.3.19 __pad26__

```
lsentences __pad26__
```

parses List items texts

Definition at line 774 of file parser.y.

5.5.3.20 __pad27__

```
optspace __pad27__
```

parses optional spaces for some specific cases

Definition at line 814 of file parser.y.

5.5.3.21 __pad2__

```
table __pad2__
```

parses the Table component in the code

Definition at line 153 of file parser.y.

5.5.3.22 __pad3__

```
thead __pad3__
```

parses heading component of the Table

Definition at line 164 of file parser.y.

5.5.3.23 __pad4__

```
list __pad4__
```

parses the list component

Definition at line 175 of file parser.y.

5.5.3.24 __pad5__

```
depthD __pad5__
```

Increases the depth of list in case of nested lists.

Definition at line 190 of file parser.y.

5.5.3.25 __pad6__

```
depthI __pad6__
```

Decreases the depth of list in case of nested lists.

Definition at line 192 of file parser.y.

5.5.3.26 __pad7__

```
operationList __pad7__
```

Parses the simple operations.

Definition at line 198 of file parser.y.

5.5.3.27 __pad8__

```
operations __pad8__
```

Parses various operations.

Definition at line 210 of file parser.y.

5.5.3.28 __pad9__

```
url __pad9__
```

parses the url in href component

Definition at line 347 of file parser.y.

5.5.3.29 APER

```
APER
```

Initial value:
```
{
                                                    $$ = new Node(Symbols)
```

Definition at line 617 of file parser.y.

5.5.3.30 BSLASH

```
BSLASH
```

Initial value:
```
{
                                                    $$ = new Node(Symbols)
```

Definition at line 639 of file parser.y.

5.5.3.31 cnt

```
int cnt = 0
```

Definition at line 16 of file parser.y.

5.5.3.32 codecontent

```
symbols codecontent
```

Initial value:
```
{
                                                    $$ = new Node(Codecontent)
```

Definition at line 549 of file parser.y.

5.5.3.33 debug

```
debug
```

Initial value:
```
{
# 12 "/mnt/d/Mtech/COP701/Latex-to-markdown/parser.y" 2
    extern int yylineno
```

Definition at line 8 of file parser.y.

5.5.3.34 depth

```
depth = cnt
```

Definition at line 376 of file parser.y.

5.5.3.35 DMEND

```
DMOPEN sentences DMEND
```

Initial value:
```
{
                                                            $$ = new Node(Blocks)
```

Definition at line 137 of file parser.y.

5.5.3.36 ECBLOCK

```
BCBLOCK codecontent ECBLOCK
```

Initial value:
```
{
                                                            $$ = new Node(Blocks)
```

Definition at line 126 of file parser.y.

5.5.3.37 EDOC

```
BDOC NEWLINE program EDOC
```

Initial value:
```
{
                                                            root = new Node(Start)
```

Definition at line 76 of file parser.y.

5.5.3.38 ETABLE

```
BTABLE table ETABLE
```

Initial value:

```
{
                                                    $$ = new Node(Blocks)
```

Definition at line 117 of file parser.y.

5.5.3.39 HRULE

```
HRULE
```

Initial value:

```
{
                                                    $$ = new Node(Operations)
```

Definition at line 289 of file parser.y.

5.5.3.40 IMATH

```
IMATH sentences IMATH
```

Initial value:

```
{
                                                    $$ = new Node(Operations)
```

Definition at line 331 of file parser.y.

5.5.3.41 lsentences

```
TEXT lsentences
```

Initial value:

```
{
                                                    $$ = new Node(Lsentences)
```

Definition at line 786 of file parser.y.

5.5.3.42 LSQRB

```
LSQRB
```

Initial value:

```
{
                                                    $$ = new Node(Symbols)
```

Definition at line 594 of file parser.y.

5.5.3.43 NEWLINE

```
NEWLINE
```

Initial value:

```
{
                                                                        $$ = new Node(List)
```

Definition at line 182 of file parser.y.

5.5.3.44 node

```
Node* node
```

Definition at line 38 of file parser.y.

5.5.3.45 oitem

```
list oitem
```

Initial value:

```
{
                                                                        $$ = new Node(Oitem)
```

Definition at line 399 of file parser.y.

5.5.3.46 ospace

```
ospace
```

Initial value:

```
{
                                                                        $$ = new Node(Ospaces)
```

Definition at line 711 of file parser.y.

5.5.3.47 ospaces

```
APER ospaces
```

Initial value:

```
{
                                                                        trow++
```

Definition at line 459 of file parser.y.

5.5.3.48 ostatement

```
SOUT ostatement
```

Initial value:

```
{
                                                        $$ = new Node(Operations)
```

Definition at line 217 of file parser.y.

5.5.3.49 PARAGRAPH

```
PARAGRAPH
```

Initial value:

```
{
                                                        $$ = new Node(Operations)
```

Definition at line 311 of file parser.y.

5.5.3.50 PIPE

```
PIPE
```

Initial value:

```
{
                                                        $$ = new Node(Symbols)
```

Definition at line 628 of file parser.y.

5.5.3.51 program

```
blocks program
```

Initial value:

```
{
                                                        $$ = new Node(Program)
```

Definition at line 91 of file parser.y.

5.5.3.52 RCURB

```
RCURB
```

Initial value:

```
{
                                                        $$ = new Node(Symbols)
```

Definition at line 582 of file parser.y.

5.5.3.53 requires

```
code requires
```

Initial value:

```
{
# 27 "/mnt/d/Mtech/COP701/Latex-to-markdown/parser.y" 2
    using namespace std
```

Definition at line 23 of file parser.y.

5.5.3.54 root

```
Node* root
```

Definition at line 14 of file parser.y.

5.5.3.55 rownum

```
rownum = trow
```

Definition at line 464 of file parser.y.

5.5.3.56 RSQRB

```
RSQRB
```

Initial value:

```
{
                                                            $$ = new Node(Symbols)
```

Definition at line 606 of file parser.y.

5.5.3.57 sentences

```
sentence sentences
```

Initial value:

```
{
                                                            $$ = new Node(Sentences)
```

Definition at line 668 of file parser.y.

5.5.3.58 SPACE

```
SPACE
```

Initial value:

```
{
                                                    $$ = new Node(Empty)
```

Definition at line 818 of file parser.y.

5.5.3.59 st

```
std::string st = *(temp->getValue())
```

Definition at line 442 of file parser.y.

5.5.3.60

```
expect { ...  } start
```

Union to define the data types of the tokens.

type defines the non-terminals in the grammer

token defines the tokens generated by lexer

Start symbol of the grammar.

It parses the include libraries and other code before encountering the start of the required code.

5.5.3.61 str

```
std::string* str
```

Definition at line 37 of file parser.y.

5.5.3.62 tcol

```
tcol = 0
```

Definition at line 17 of file parser.y.

5.5.3.63 temp

```
Node * temp = new Node(Code)
```

Definition at line 130 of file parser.y.

5.5.3.64 tlines

```
tline tlines
```

Initial value:

```
{
                                                $$ = new Node(Tlines)
```

Definition at line 491 of file parser.y.

5.5.3.65 trow

```
trow = 0
```

Definition at line 18 of file parser.y.

5.5.3.66 tstr

```
std::vector<std::string> tstr
```

Definition at line 19 of file parser.y.

5.5.3.67 tstruct

```
tstruct = tstr
```

Definition at line 465 of file parser.y.

5.5.3.68 tstructure

```
TEXT tstructure
```

Initial value:

```
{
                                                $$ = new Node(Tstructure)
```

Definition at line 424 of file parser.y.

5.5.3.69 unoitem

```
list unoitem
```

Initial value:

```
{
                                                    $$ = new Node(Unoitem)
```

Definition at line 374 of file parser.y.

5.6 /mnt/d/Mtech/COP701/Latex-to-markdown/README.md File Reference

Index