

Latex2Markdown

Generated by Doxygen 1.9.1

<b>1 Main Page</b>	<b>1</b>
1.1 LaTeX-to-Markdown Converter	1
1.1.1 Overview	1
1.1.2 Features Implemented	1
1.1.3 Workflow	2
1.1.4 Dependencies	2
1.1.4.1 Build Instructions	2
1.1.4.2 Gtest Instruction	2
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Node Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Node()	8
4.1.3 Member Function Documentation	8
4.1.3.1 convert2Markdown()	8
4.1.3.2 getType()	12
4.1.3.3 getValue()	13
4.1.3.4 printAST()	13
4.1.3.5 setValue()	13
4.1.4 Member Data Documentation	14
4.1.4.1 depth	14
4.1.4.2 productions	14
4.1.4.3 rownum	14
4.1.4.4 tstruct	14
<b>5 File Documentation</b>	<b>15</b>
5.1 /mnt/d/Mtech/COP701/Latex-to-markdown/lexer.l File Reference	15
5.1.1 Detailed Description	16
5.1.2 Function Documentation	16
5.1.2.1 WORD()	16
5.1.2.2 yywrap()	16
5.1.3 Variable Documentation	16
5.1.3.1 sout	17
5.2 /mnt/d/Mtech/COP701/Latex-to-markdown/main.cpp File Reference	18
5.2.1 Detailed Description	18
5.2.2 Function Documentation	19
5.2.2.1 getRoot()	19

5.2.2.2 main()	19
5.2.2.3 yylex()	20
5.2.2.4 yyparse()	21
5.2.3 Variable Documentation	21
5.2.3.1 yydebug	21
5.3 /mnt/d/Mtech/COP701/Latex-to-markdown/Node.cpp File Reference	21
5.3.1 Detailed Description	22
5.3.2 Function Documentation	22
5.3.2.1 getEnumValue()	22
5.4 /mnt/d/Mtech/COP701/Latex-to-markdown/Node.hpp File Reference	24
5.4.1 Detailed Description	26
5.4.2 Enumeration Type Documentation	26
5.4.2.1 symbol	26
5.5 /mnt/d/Mtech/COP701/Latex-to-markdown/parser.y File Reference	28
5.5.1 Detailed Description	31
5.5.2 Function Documentation	31
5.5.2.1 clear()	31
5.5.2.2 getRoot()	31
5.5.2.3 push_back() [1/3]	32
5.5.2.4 push_back() [2/3]	32
5.5.2.5 push_back() [3/3]	32
5.5.2.6 setValue() [1/6]	32
5.5.2.7 setValue() [2/6]	32
5.5.2.8 setValue() [3/6]	32
5.5.2.9 setValue() [4/6]	32
5.5.2.10 setValue() [5/6]	33
5.5.2.11 setValue() [6/6]	33
5.5.2.12 yylex()	33
5.5.3 Variable Documentation	33
5.5.3.1 __pad0__	33
5.5.3.2 __pad10__	33
5.5.3.3 __pad11__	34
5.5.3.4 __pad12__	34
5.5.3.5 __pad13__	34
5.5.3.6 __pad14__	34
5.5.3.7 __pad15__	34
5.5.3.8 __pad16__	35
5.5.3.9 __pad17__	35
5.5.3.10 __pad18__	35
5.5.3.11 __pad19__	35
5.5.3.12 __pad1__	35
5.5.3.13 __pad20__	36

5.5.3.14 __pad21__ . . . . .	36
5.5.3.15 __pad22__ . . . . .	36
5.5.3.16 __pad23__ . . . . .	36
5.5.3.17 __pad24__ . . . . .	36
5.5.3.18 __pad25__ . . . . .	37
5.5.3.19 __pad26__ . . . . .	37
5.5.3.20 __pad27__ . . . . .	37
5.5.3.21 __pad2__ . . . . .	37
5.5.3.22 __pad3__ . . . . .	37
5.5.3.23 __pad4__ . . . . .	38
5.5.3.24 __pad5__ . . . . .	38
5.5.3.25 __pad6__ . . . . .	38
5.5.3.26 __pad7__ . . . . .	38
5.5.3.27 __pad8__ . . . . .	38
5.5.3.28 __pad9__ . . . . .	39
5.5.3.29 APER . . . . .	39
5.5.3.30 BSLASH . . . . .	39
5.5.3.31 cnt . . . . .	39
5.5.3.32 codecontent . . . . .	39
5.5.3.33 debug . . . . .	40
5.5.3.34 depth . . . . .	40
5.5.3.35 DMEND . . . . .	40
5.5.3.36 ECBLOCK . . . . .	40
5.5.3.37 EDOC . . . . .	40
5.5.3.38 ETABLE . . . . .	41
5.5.3.39 HRULE . . . . .	41
5.5.3.40 IMATH . . . . .	41
5.5.3.41 lsentences . . . . .	41
5.5.3.42 LSQRB . . . . .	41
5.5.3.43 NEWLINE . . . . .	42
5.5.3.44 node . . . . .	42
5.5.3.45 oitem . . . . .	42
5.5.3.46 ospace . . . . .	42
5.5.3.47 ospaces . . . . .	42
5.5.3.48 ostatement . . . . .	43
5.5.3.49 PARAGRAPH . . . . .	43
5.5.3.50 PIPE . . . . .	43
5.5.3.51 program . . . . .	43
5.5.3.52 RCURB . . . . .	43
5.5.3.53 requires . . . . .	44
5.5.3.54 root . . . . .	44
5.5.3.55 rownum . . . . .	44

---

5.5.3.56 RSQRB . . . . .	44
5.5.3.57 sentences . . . . .	44
5.5.3.58 SPACE . . . . .	45
5.5.3.59 st . . . . .	45
5.5.3.60 . . . . .	45
5.5.3.61 str . . . . .	45
5.5.3.62 tcol . . . . .	45
5.5.3.63 temp . . . . .	46
5.5.3.64 tlines . . . . .	46
5.5.3.65 trow . . . . .	46
5.5.3.66 tstr . . . . .	46
5.5.3.67 tstruct . . . . .	46
5.5.3.68 tstructure . . . . .	46
5.5.3.69 unitem . . . . .	47
5.6 /mnt/d/Mtech/COP701/Latex-to-markdown/README.md File Reference . . . . .	47
<b>Index</b>	<b>49</b>

# 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:

<a href="#">Node</a>	Represents a structure of the node in the AST . . . . .	<a href="#">7</a>
----------------------	---	-------------------





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

/mnt/d/Mtech/COP701/Latex-to-markdown/ <a href="#">lexer.l</a>	
Flex file for generation of tokens . . . . .	15
/mnt/d/Mtech/COP701/Latex-to-markdown/ <a href="#">main.cpp</a>	
Main function in the project . . . . .	18
/mnt/d/Mtech/COP701/Latex-to-markdown/ <a href="#">Node.cpp</a>	
Funcion implementation for class <a href="#">Node</a> . . . . .	21
/mnt/d/Mtech/COP701/Latex-to-markdown/ <a href="#">Node.hpp</a>	
Class defination for the structure of the <a href="#">Node</a> in AST . . . . .	24
/mnt/d/Mtech/COP701/Latex-to-markdown/ <a href="#">parser.y</a>	
Bison file for parsing Latex code . . . . .	28



## 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 variable 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

<i>val</i>	Value with which the <a href="#">Node</a> is initialize.
------------	--

Definition at line 120 of file Node.hpp.

```
120 :   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.

```
188 {
189
190     switch (this->type)
191     {
192
193     case Start:
194         this->productions[0]->convert2Markdown();
```

```

195         this->setValue(new std::string(*(this->productions[0]->value)));
196         break;
197
198     case Program:
199         if(static_cast<int>(this->productions.size()) == 2){
200             this->productions[0]->convert2Markdown();
201             this->productions[1]->convert2Markdown();
202             this->setValue(new std::string(*(this->productions[0]->value) +
*(this->productions[1]->value)));
203         }
204         break;
205
206     case Text:
207         this->setValue(this->value);
208         break;
209
210     case Space:
211         this->setValue(this->value);
212         break;
213
214     case Newline:
215         this->setValue(this->value);
216         break;
217
218     case Lcurb:
219         this->setValue(new std::string("("));
220         break;
221
222     case Rcurb:
223         this->setValue(new std::string(")"));
224         break;
225
226     case Lsqrb:
227         this->setValue(new std::string("["));
228         break;
229
230     case Rsqrb:
231         this->setValue(new std::string("]"));
232         break;
233
234     case Pipe:
235         this->setValue(new std::string("|"));
236         break;
237
238     case Aper:
239         this->setValue(new std::string("&"));
240         break;
241
242     case Bslash:
243         this->setValue(new std::string("\\"));
244         break;
245
246     case Paragraph:
247         this->setValue(new std::string("\n"));
248         break;
249
250     case Graphic:
251         this->productions[0]->convert2Markdown();
252         this->setValue(new std::string("![Image] (" + *(this->productions[0]->value) + ") "));
253         break;
254
255     case Hrule:
256         this->setValue(new std::string("\n---"));
257         break;
258
259     case Href:
260         this->productions[0]->convert2Markdown();
261         this->productions[1]->convert2Markdown();
262         this->setValue(new std::string("(" + *(this->productions[1]->value) + ")" + "(" +
*(this->productions[0]->value) + ") "));
263         break;
264
265     case Subsubsection:
266         this->productions[0]->convert2Markdown();
267         this->setValue(new std::string("### " + *(this->productions[0]->value)));
268         break;
269
270     case Subsection:
271         this->productions[0]->convert2Markdown();
272         this->setValue(new std::string("## " + *(this->productions[0]->value)));
273         break;
274
275     case Section:
276         this->productions[0]->convert2Markdown();
277         this->setValue(new std::string("# " + *(this->productions[0]->value)));
278         break;
279

```

```

280     case Bold:
281         this->productions[0]->convert2Markdown();
282         this->setValue(new std::string("*" + *(this->productions[0]->value) + "*"));
283         break;
284
285     case Sout:
286         this->productions[0]->convert2Markdown();
287         this->setValue(new std::string("~" + *(this->productions[0]->value) + "~"));
288         break;
289
290     case Imath:
291         this->productions[0]->convert2Markdown();
292         this->setValue(new std::string("$" + *(this->productions[0]->value) + "$"));
293         break;
294
295     case Dmath:
296         this->productions[0]->convert2Markdown();
297         this->setValue(new std::string("$" + *(this->productions[0]->value) + "$"));
298         break;
299
300     case Italic:
301         this->productions[0]->convert2Markdown();
302         this->setValue(new std::string("*" + *(this->productions[0]->value) + "*"));
303         break;
304
305     case Url:
306         this->productions[0]->convert2Markdown();
307         this->setValue(new std::string(*(this->productions[0]->value)));
308         break;
309
310     case Ostatement:
311         this->productions[0]->convert2Markdown();
312         this->setValue(new std::string(*(this->productions[0]->value)));
313         break;
314
315     case Operationlist:
316         this->productions[0]->convert2Markdown();
317         this->setValue(new std::string(*(this->productions[0]->value)));
318         break;
319
320     case Blocks:
321         this->productions[0]->convert2Markdown();
322         this->setValue(new std::string(*(this->productions[0]->value)));
323         break;
324
325     case Empty:
326         break;
327
328     case List:
329         this->productions[0]->convert2Markdown();
330         this->setValue(new std::string(*(this->productions[0]->value)));
331         break;
332
333     case Codecontent:
334         this->productions[0]->convert2Markdown();
335         this->productions[1]->convert2Markdown();
336         this->setValue(new std::string(*(this->productions[0]->value) +
337 *(this->productions[1]->value)));
338         break;
339
340     case Code:
341         this->productions[0]->convert2Markdown();
342         this->setValue(new std::string("`" + *(this->productions[0]->value) + "`"));
343         break;
344
345     case Symbols:
346         this->productions[0]->convert2Markdown();
347         this->setValue(new std::string(*(this->productions[0]->value)));
348         break;
349
350     case Lsentences:
351         if(static_cast<int>(this->productions.size()) == 2){
352             this->productions[0]->convert2Markdown();
353             this->productions[1]->convert2Markdown();
354             this->setValue(new std::string(*(this->productions[0]->value) +
355 *(this->productions[1]->value)));
356         }else{
357             this->productions[0]->convert2Markdown();
358             this->setValue(new std::string(*(this->productions[0]->value)));
359         }
360         break;
361
362     case Ospaces:
363         this->productions[0]->convert2Markdown();
364         this->productions[1]->convert2Markdown();
365         this->setValue(new std::string(*(this->productions[1]->value) +
366 *(this->productions[0]->value)));

```

```

364         break;
365
366     case Ospace:
367         this->productions[0]->convert2Markdown();
368         this->setValue(new std::string(*(this->productions[0]->value)));
369         break;
370
371     case Sentences:
372         if(static_cast<int>(this->productions.size()) == 2){
373             this->productions[0]->convert2Markdown();
374             this->productions[1]->convert2Markdown();
375             this->setValue(new std::string(*(this->productions[0]->value) +
376 *(this->productions[1]->value)));
377         }else{
378             this->productions[0]->convert2Markdown();
379             this->setValue(new std::string(*(this->productions[0]->value)));
380         }
381         break;
382
383     case Sentence:
384         this->productions[0]->convert2Markdown();
385         this->setValue(new std::string(*(this->productions[0]->value)));
386         break;
387
388     case Gdata:
389         this->productions[0]->convert2Markdown();
390         this->setValue(new std::string(*(this->productions[0]->value)));
391         break;
392
393     case Gsentences:
394         this->productions[0]->convert2Markdown();
395         this->productions[1]->convert2Markdown();
396         this->setValue(new std::string(*(this->productions[1]->value) +
397 *(this->productions[0]->value)));
398         break;
399
400     case Gsentence:
401         this->productions[0]->convert2Markdown();
402         this->setValue(new std::string(*(this->productions[0]->value)));
403         break;
404
405     case Operations:
406         this->productions[0]->convert2Markdown();
407         this->setValue(new std::string(*(this->productions[0]->value)));
408         break;
409
410     case Oitem:
411         this->productions[0]->convert2Markdown();
412         this->productions[1]->convert2Markdown();
413         if(this->productions[0]->getType() == Lsentences){
414             for(int i = 1 ; i < this->depth ; i++){
415                 this->setValue(new std::string(*(this->value) + " "));
416             }
417             this->setValue(new std::string(*(this->value) + "1." + *(this->productions[0]->value) +
418 *(this->productions[1]->value)));
419         }else{
420             this->setValue(new std::string(*(this->productions[0]->value) +
421 *(this->productions[1]->value)));
422         }
423         break;
424
425     case Unoittem:
426         this->productions[0]->convert2Markdown();
427         this->productions[1]->convert2Markdown();
428         if(this->productions[0]->getType() == Lsentences){
429             for(int i = 1 ; i < this->depth ; i++){
430                 this->setValue(new std::string(*(this->value) + " "));
431             }
432             this->setValue(new std::string(*(this->value) + "-" + *(this->productions[0]->value) +
433 *(this->productions[1]->value)));
434         }else{
435             this->setValue(new std::string(*(this->productions[0]->value) +
436 *(this->productions[1]->value)));
437         }
438         break;
439
440     case Tcontent:
441         this->productions[0]->convert2Markdown();
442         this->productions[1]->convert2Markdown();
443
444         if(this->productions[1]->getType() == Tlines){
445             this->setValue(new std::string(*(this->productions[0]->value) + "| " +
446 *(this->productions[1]->value) + "\n"));
447             if(this->rownum == 1){
448                 std::string* temp = new std::string("");
449                 for(int i = this->tstruct.size()-1 ; i >= 0; i--){
450                     if(this->tstruct[i] == "1"){

```



```

444         temp = new std::string(*temp + "| :--- ");
445     }else if(this->tstruct[i] == "x"){
446         temp = new std::string(*temp + "| ---: ");
447     }else{
448         temp = new std::string(*temp + "| :---: ");
449     }
450 }
451 }
452 temp = new std::string(*temp + "\n");
453 this->setValue(new std::string(*(this->value) + *temp));
454 }
455 }else{
456     this->setValue(new std::string(*(this->productions[0]->value) ));
457 }
458 break;
459
460 case Tlines:
461     if(static_cast<int>(this->productions.size()) == 2){
462         this->productions[0]->convert2Markdown();
463         this->productions[1]->convert2Markdown();
464         this->setValue(new std::string(*(this->productions[0]->value) +
465 *(this->productions[1]->value)));
466     }else{
467         this->productions[0]->convert2Markdown();
468         this->setValue(new std::string(*(this->productions[0]->value)));
469     }
470     break;
471
472 case Tline:
473     this->productions[0]->convert2Markdown();
474     this->productions[1]->convert2Markdown();
475     if(this->productions[0]->getType() == Text){
476         this->setValue(new std::string(*(this->productions[0]->value) +
477 *(this->productions[1]->value)));
478     }else{
479         this->setValue(new std::string(" | " + *(this->productions[1]->value)));
480     }
481     break;
482
483 case Hline:
484     this->setValue(new std::string(""));
485     break;
486
487 case Table:
488     this->productions[0]->convert2Markdown();
489     this->setValue(new std::string(*(this->productions[0]->value)));
490     break;
491
492 default:
493     break;
494 }
495 }

```

#### 4.1.3.2 getType()

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

Used to access the private variable value.

#### Returns

Returns the type of the [Node](#).

Definition at line 103 of file Node.hpp.

```

103     {
104         return type;
105     }

```

#### 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.

```
115     {
116         return value;
117     }
```

#### 4.1.3.4 printAST()

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

Prints the Abstract syntax tree depth wise.

##### Parameters

<i>node</i>	The node which type will be printed.
<i>depth</i>	The depth at which the current node is present.

Definition at line 504 of file Node.cpp.

```
504     {
505         if(node->productions.size() == 2){
506             for(int i = 0 ; i < depth; i++){
507                 std::cout<<" ";
508             }
509             std::cout<<getEnumValue(node->getType())<<"\n";
510             printAST(node->productions[0], depth + 1);
511             printAST(node->productions[1], depth + 1);
512         }else if(node->productions.size() == 1){
513             for(int i = 0 ; i < depth; i++){
514                 std::cout<<" ";
515             }
516             std::cout<<getEnumValue(node->getType())<<"\n";
517             printAST(node->productions[0], depth + 1);
518         }
519     }
```

#### 4.1.3.5 setValue()

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

Used to set the value of the private variable val.

### Parameters

<code>val</code>	value to be store as the <a href="#">Node's</a> 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](#)

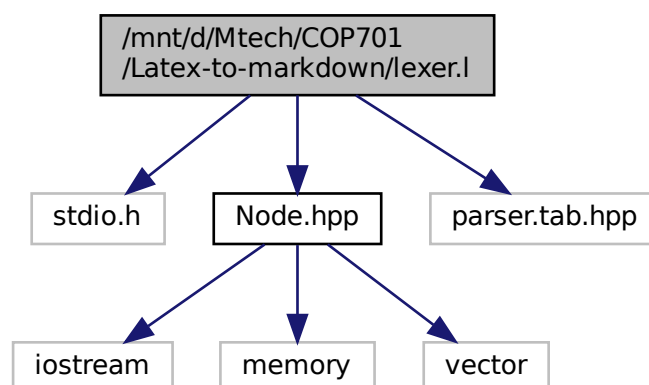
## Chapter 5

# 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` `UNO-` `LIST` `itemize` `OLIST` `enumerate` `CBLOCK` `verbatim` `ITEM` `item` `SOUT` `sout` `{BSLASH}``{ITALIC}` `{return ITALIC;` `}`

*Token for Ampersand.*

### 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`.

```
49      {ALPHABET}|{NUMERIC})+({ALPHABET}|{NUMERIC}) *)
50
54 TEXT      (({WORD}|{PUNCTUATION})+)
55
56
59 LCURB      \{
60
64 RCURB      \}
```

#### 5.1.2.2 yywrap()

```
int yywrap ( )
```

Definition at line 258 of file `lexer.l`.

```
258      {
259      return 1;
260 }
```

### 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 subsection 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 Ampersand.

Token for left square bracket.

Token for pipe.

Token for dollar.

Token for newline.

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.

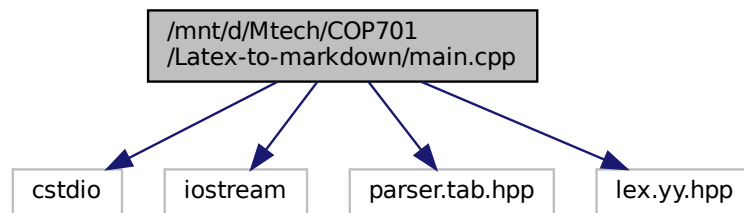
Token for block type.

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.

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

### 5.2.2.2 main()

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

Main function in the project.



**Parameters**

<i>argc</i>	Denotes the count of the arguments.
<i>argv</i>	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.

```

34      {
35          // yydebug = 1;
36          if (argc < 2) {
37              std::cerr << "Usage:  " << argv[0] << " <filename>" << std::endl;
38              return 1;
39          }
40
41
42          const char* inputFile = argv[1];
43          const char* outputFile = argv[2];
44
45          FILE *inputPtr, *outputPtr;
46
47          inputPtr = fopen(inputFile, "r");
48          if (inputPtr == NULL) {
49              cout<<"Error:  Unable to open latex File\n.";
50              return 0;
51          }
52
53          outputPtr = fopen(outputFile, "w");
54
55          if (outputPtr == NULL) {
56              cout<<"Error:  Unable to open Markdown File\n.";
57              return 0;
58          }
59
60          extern FILE *yyin;
61
62          yyin = inputPtr;
63
64
65          yyparse();
66
67          Node* temp = getRoot();
68          temp->convert2Markdown();
69
70          fprintf(outputPtr, "%s\n", temp->getValue()->c_str());
71
72          temp->printAST(temp, 0);
73
74          fclose(inputPtr);
75          fclose(outputPtr);
76          return 0;
77      }
78  }
```

**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]
```

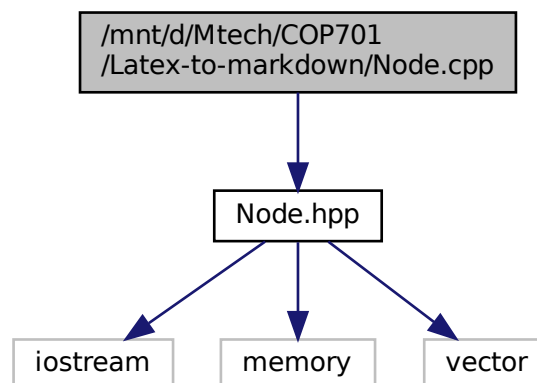
Used to debug the parsing process.

## 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

Function implementation for class [Node](#).

#### Author

Shreyash Chikte ( [shreyashsc9@gmail.com](mailto: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

<i>type</i>	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:
```

```

33         return "Gsentences_Node";
34         break;
35     case Oitem:
36         return "Oitem_Node";
37         break;
38     case Gdata:
39         return "Gdata_Node";
40         break;
41     case Sentence:
42         return "Sentence_Node";
43         break;
44     case Sentences:
45         return "Sentences_Node";
46         break;
47     case Ospace:
48         return "Ospace_Node";
49         break;
50     case Ospaces:
51         return "Ospaces_Node";
52         break;
53     case Tcontent:
54         return "Tcontent_Node";
55         break;
56     case Tstructure:
57         return "Tstructure_Node";
58         break;
59     case Tline:
60         return "Tline_Node";
61         break;
62     case Tlines:
63         return "Tlines_Node";
64         break;
65     case Lsentences:
66         return "Lsentences_Node";
67         break;
68     case Symbols:
69         return "Symbold_Node";
70         break;
71     case List:
72         return "List_Node";
73         break;
74     case Codecontent:
75         return "Codecontent_Node";
76         break;
77     case Start:
78         return "Start_Node";
79         break;
80     case Program:
81         return "Program_Node";
82         break;
83     case Blocks:
84         return "Blocks_Node";
85         break;
86     case Operationlist:
87         return "Operationlist_Node";
88         break;
89     case Ostatement:
90         return "Ostatement_Node";
91         break;
92     case Table:
93         return "Table_Node";
94         break;
95     case Thead:
96         return "Thead";
97         break;
98     case Url:
99         return "Url_Node";
100        break;
101    case Text:
102        return "Text_Node";
103        break;
104    case Space:
105        return "Space_Node";
106        break;
107    case Newline:
108        return "Newline_Node";
109        break;
110    case Italic:
111        return "Italic_Node";
112        break;
113    case Bold:
114        return "Bold_Node";
115        break;
116    case Section:
117        return "Section_Node";
118        break;
119    case Subsection:

```

```

120         return "Subsection_Node";
121     break;
122     case Subsubsection:
123         return "Subsubsection_Node";
124     break;
125     case Href:
126         return "Href_Node";
127     break;
128     case Hrule:
129         return "Hrule_Node";
130     break;
131     case Graphic:
132         return "Graphic_Node";
133     break;
134     case Paragraph:
135         return "Paragraph_Node";
136     break;
137     case Lcurb:
138         return "Left curly bracket Node";
139     break;
140     case Rcurb:
141         return "Right curly bracket Node";
142     break;
143     case Lsqrb:
144         return "Left square bracket Node";
145     break;
146     case Rsqrb:
147         return "Right square bracket Node";
148     break;
149     case Aper:
150         return "Ampersand_Node";
151     break;
152     case Pipe:
153         return "Pipe_Node";
154     break;
155     case Bslash:
156         return "Back Slash Node";
157     break;
158     case Code:
159         return "Code_Node";
160     break;
161     case Empty:
162         return "Empty_Node";
163     break;
164     case Hline:
165         return "Hline_Node";
166     break;
167     case Dmath:
168         return "Display Math Node";
169     break;
170     case Sout:
171         return "Strike Through Node";
172     break;
173     case Imath:
174         return "Inline math Node";
175     break;
176     default:
177         return "Unexpected_Node";
178     break;
179 }
180 }

```

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

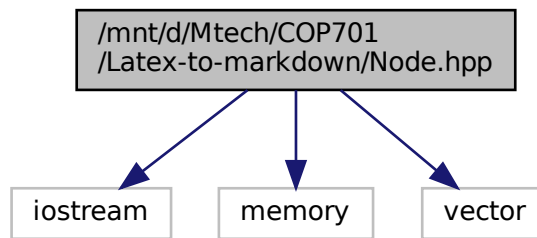
Class definition for the structure of the [Node](#) in AST.

```

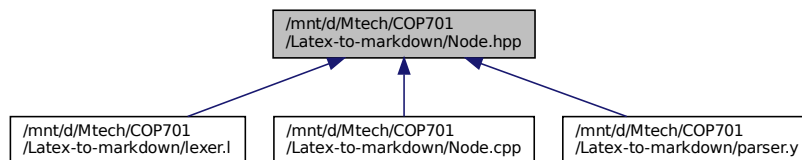
#include <iostream>
#include <memory>
#include <vector>

```

Include dependency graph for Node.hpp:



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



## 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](#) , [Rsqrbr](#) , [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](mailto: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	
Lsqr	
Rsqr	
Aper	
Pipe	
Bslash	
Code	
Empty	
Hline	
Dmath	
Sout	
Imath	

Definition at line 21 of file Node.hpp.

```

21     {
22         Operations,
23         Unoitem,
24         Gsentence,
25         Gsentences,
26         Oitem,
27         Gdata,
28         Sentence,
29         Sentences,
30         Ospace,
31         Ospaces,
32         Tcontent,
33         Tstructure,
34         Tline,
35         Tlines,
36         Lsentences,
37         Symbols,
38         List,
39         Codecontent,
40         Start,

```



```

41     Program,
42     Blocks,
43     Operationlist,
44     Ostatement,
45     Table,
46     Thead,
47     Url,
48     Text,
49     Space,
50     Newline,
51     Italic,
52     Bold,
53     Section,
54     Subsection,
55     Subsubsection,
56     Href,
57     Hruler,
58     Graphic,
59     Paragraph,
60     Lcurb,
61     Rcurb,
62     Lsqr,
63     Rsqr,
64     Aperc,
65     Pipe,
66     Bslash,
67     Code,
68     Empty,
69     Hline,
70     Dmath,
71     Sout,
72     Imath,
73 };

```

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

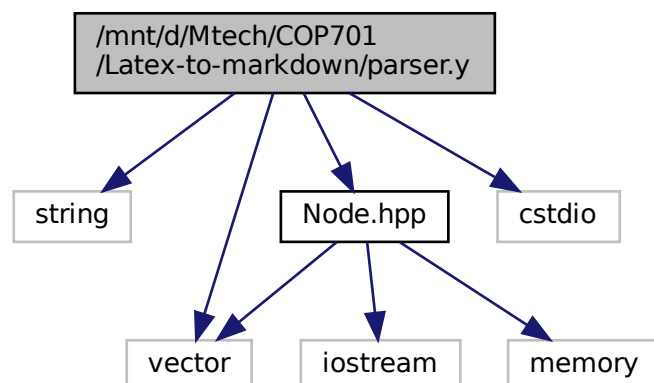
Bison file for parsing Latex code.

```

#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 `tc` = 0
- int `tr` = 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 graphics 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:

```
{  
    throw++
```

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 grammar

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 throw

```
throw = 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

/mnt/d/Mtech/COP701/Latex-to-markdown/Node.cpp, [21](#)  
/mnt/d/Mtech/COP701/Latex-to-markdown/Node.hpp, [24](#)  
/mnt/d/Mtech/COP701/Latex-to-markdown/README.md, [47](#)  
/mnt/d/Mtech/COP701/Latex-to-markdown/lexer.l, [15](#)  
/mnt/d/Mtech/COP701/Latex-to-markdown/main.cpp, [18](#)  
/mnt/d/Mtech/COP701/Latex-to-markdown/parser.y, [28](#)  
\_\_pad0\_\_  
    [parser.y, 33](#)  
\_\_pad10\_\_  
    [parser.y, 33](#)  
\_\_pad11\_\_  
    [parser.y, 33](#)  
\_\_pad12\_\_  
    [parser.y, 34](#)  
\_\_pad13\_\_  
    [parser.y, 34](#)  
\_\_pad14\_\_  
    [parser.y, 34](#)  
\_\_pad15\_\_  
    [parser.y, 34](#)  
\_\_pad16\_\_  
    [parser.y, 34](#)  
\_\_pad17\_\_  
    [parser.y, 35](#)  
\_\_pad18\_\_  
    [parser.y, 35](#)  
\_\_pad19\_\_  
    [parser.y, 35](#)  
\_\_pad1\_\_  
    [parser.y, 35](#)  
\_\_pad20\_\_  
    [parser.y, 35](#)  
\_\_pad21\_\_  
    [parser.y, 36](#)  
\_\_pad22\_\_  
    [parser.y, 36](#)  
\_\_pad23\_\_  
    [parser.y, 36](#)  
\_\_pad24\_\_  
    [parser.y, 36](#)  
\_\_pad25\_\_  
    [parser.y, 36](#)  
\_\_pad26\_\_  
    [parser.y, 37](#)  
\_\_pad27\_\_  
    [parser.y, 37](#)  
\_\_pad2\_\_  
    [parser.y, 37](#)  
\_\_pad3\_\_  
    [parser.y, 37](#)  
\_\_pad4\_\_  
    [parser.y, 37](#)  
\_\_pad5\_\_  
    [parser.y, 38](#)  
\_\_pad6\_\_  
    [parser.y, 38](#)  
\_\_pad7\_\_  
    [parser.y, 38](#)  
\_\_pad8\_\_  
    [parser.y, 38](#)  
\_\_pad9\_\_  
    [parser.y, 38](#)  
APER  
    [parser.y, 39](#)  
Aper  
    [Node.hpp, 27](#)  
Blocks  
    [Node.hpp, 27](#)  
Bold  
    [Node.hpp, 27](#)  
BSLASH  
    [parser.y, 39](#)  
Bslash  
    [Node.hpp, 27](#)  
clear  
    [parser.y, 31](#)  
cnt  
    [parser.y, 39](#)  
Code  
    [Node.hpp, 27](#)  
Codecontent  
    [Node.hpp, 27](#)  
codecontent  
    [parser.y, 39](#)  
convert2Markdown  
    [Node, 8](#)  
debug  
    [parser.y, 39](#)  
depth  
    [Node, 14](#)  
    [parser.y, 40](#)  
Dmath

- Node.hpp, [27](#)
- DMEND
  - parser.y, [40](#)
- ECBLOCK
  - parser.y, [40](#)
- EDOC
  - parser.y, [40](#)
- Empty
  - Node.hpp, [27](#)
- ETABLE
  - parser.y, [40](#)
- Gdata
  - Node.hpp, [26](#)
- getEnumValue
  - Node.cpp, [22](#)
- getRoot
  - main.cpp, [19](#)
  - parser.y, [31](#)
- getType
  - Node, [12](#)
- getValue
  - Node, [12](#)
- Graphic
  - Node.hpp, [27](#)
- Gsentence
  - Node.hpp, [26](#)
- Gsentences
  - Node.hpp, [26](#)
- Hline
  - Node.hpp, [27](#)
- Href
  - Node.hpp, [27](#)
- HRULE
  - parser.y, [41](#)
- Hrule
  - Node.hpp, [27](#)
- IMATH
  - parser.y, [41](#)
- Imath
  - Node.hpp, [27](#)
- Italic
  - Node.hpp, [27](#)
- Lcurb
  - Node.hpp, [27](#)
- lexer.l
  - sout, [16](#)
  - WORD, [16](#)
  - yywrap, [16](#)
- List
  - Node.hpp, [27](#)
- Lsentences
  - Node.hpp, [26](#)
- lsentences
  - parser.y, [41](#)
- LSQRB
  - parser.y, [41](#)
- Lsqrb
  - Node.hpp, [27](#)
- main
  - main.cpp, [19](#)
- main.cpp
  - getRoot, [19](#)
  - main, [19](#)
  - yydebug, [21](#)
  - yylex, [20](#)
  - yyparse, [20](#)
- NEWLINE
  - parser.y, [41](#)
- Newline
  - Node.hpp, [27](#)
- Node, [7](#)
  - convert2Markdown, [8](#)
  - depth, [14](#)
  - getType, [12](#)
  - getValue, [12](#)
  - Node, [8](#)
  - printAST, [13](#)
  - productions, [14](#)
  - rownum, [14](#)
  - setValue, [13](#)
  - tstruct, [14](#)
- node
  - parser.y, [42](#)
- Node.cpp
  - getEnumValue, [22](#)
- Node.hpp
  - Aper, [27](#)
  - Blocks, [27](#)
  - Bold, [27](#)
  - Bslash, [27](#)
  - Code, [27](#)
  - Codecontent, [27](#)
  - Dmath, [27](#)
  - Empty, [27](#)
  - Gdata, [26](#)
  - Graphic, [27](#)
  - Gsentence, [26](#)
  - Gsentences, [26](#)
  - Hline, [27](#)
  - Href, [27](#)
  - Hrule, [27](#)
  - Imath, [27](#)
  - Italic, [27](#)
  - Lcurb, [27](#)
  - List, [27](#)
  - Lsentences, [26](#)
  - Lsqrb, [27](#)
  - Newline, [27](#)
  - Oitem, [26](#)
  - Operationlist, [27](#)
  - Operations, [26](#)

- Ospace, [26](#)
- Ospaces, [26](#)
- Ostatement, [27](#)
- Paragraph, [27](#)
- Pipe, [27](#)
- Program, [27](#)
- Rcurb, [27](#)
- Rsqr, [27](#)
- Section, [27](#)
- Sentence, [26](#)
- Sentences, [26](#)
- Sout, [27](#)
- Space, [27](#)
- Start, [27](#)
- Subsection, [27](#)
- Subsubsection, [27](#)
- symbol, [26](#)
- Symbols, [26](#)
- Table, [27](#)
- Tcontent, [26](#)
- Text, [27](#)
- Thead, [27](#)
- Tline, [26](#)
- Tlines, [26](#)
- Tstructure, [26](#)
- Unoitem, [26](#)
- Url, [27](#)
- Oitem
  - Node.hpp, [26](#)
- oitem
  - parser.y, [42](#)
- Operationlist
  - Node.hpp, [27](#)
- Operations
  - Node.hpp, [26](#)
- Ospace
  - Node.hpp, [26](#)
- ospace
  - parser.y, [42](#)
- Ospaces
  - Node.hpp, [26](#)
- ospaces
  - parser.y, [42](#)
- Ostatement
  - Node.hpp, [27](#)
- ostatement
  - parser.y, [42](#)
- PARAGRAPH
  - parser.y, [43](#)
- Paragraph
  - Node.hpp, [27](#)
- parser.y
  - \_\_pad0\_\_, [33](#)
  - \_\_pad10\_\_, [33](#)
  - \_\_pad11\_\_, [33](#)
  - \_\_pad12\_\_, [34](#)
  - \_\_pad13\_\_, [34](#)
  - \_\_pad14\_\_, [34](#)
  - \_\_pad15\_\_, [34](#)
  - \_\_pad16\_\_, [34](#)
  - \_\_pad17\_\_, [35](#)
  - \_\_pad18\_\_, [35](#)
  - \_\_pad19\_\_, [35](#)
  - \_\_pad1\_\_, [35](#)
  - \_\_pad20\_\_, [35](#)
  - \_\_pad21\_\_, [36](#)
  - \_\_pad22\_\_, [36](#)
  - \_\_pad23\_\_, [36](#)
  - \_\_pad24\_\_, [36](#)
  - \_\_pad25\_\_, [36](#)
  - \_\_pad26\_\_, [37](#)
  - \_\_pad27\_\_, [37](#)
  - \_\_pad2\_\_, [37](#)
  - \_\_pad3\_\_, [37](#)
  - \_\_pad4\_\_, [37](#)
  - \_\_pad5\_\_, [38](#)
  - \_\_pad6\_\_, [38](#)
  - \_\_pad7\_\_, [38](#)
  - \_\_pad8\_\_, [38](#)
  - \_\_pad9\_\_, [38](#)
  - APER, [39](#)
  - BSLASH, [39](#)
  - clear, [31](#)
  - cnt, [39](#)
  - codecontent, [39](#)
  - debug, [39](#)
  - depth, [40](#)
  - DMEND, [40](#)
  - ECBLOCK, [40](#)
  - EDOC, [40](#)
  - ETABLE, [40](#)
  - getRoot, [31](#)
  - HRULE, [41](#)
  - IMATH, [41](#)
  - lsentences, [41](#)
  - LSQRB, [41](#)
  - NEWLINE, [41](#)
  - node, [42](#)
  - oitem, [42](#)
  - ospace, [42](#)
  - ospaces, [42](#)
  - ostatement, [42](#)
  - PARAGRAPH, [43](#)
  - PIPE, [43](#)
  - program, [43](#)
  - push\_back, [31](#), [32](#)
  - RCURB, [43](#)
  - requires, [43](#)
  - root, [44](#)
  - rownum, [44](#)
  - RSQRB, [44](#)
  - sentences, [44](#)
  - setValue, [32](#), [33](#)
  - SPACE, [44](#)
  - st, [45](#)



- start, [45](#)
- str, [45](#)
- tcol, [45](#)
- temp, [45](#)
- tlines, [46](#)
- throw, [46](#)
- tstr, [46](#)
- tstruct, [46](#)
- tstructure, [46](#)
- unoitem, [46](#)
- yylex, [33](#)
- PIPE
  - parser.y, [43](#)
- Pipe
  - Node.hpp, [27](#)
- printAST
  - Node, [13](#)
- productions
  - Node, [14](#)
- Program
  - Node.hpp, [27](#)
- program
  - parser.y, [43](#)
- push\_back
  - parser.y, [31](#), [32](#)
- RCURB
  - parser.y, [43](#)
- Rcurb
  - Node.hpp, [27](#)
- requires
  - parser.y, [43](#)
- root
  - parser.y, [44](#)
- rownum
  - Node, [14](#)
  - parser.y, [44](#)
- RSQRB
  - parser.y, [44](#)
- RsqrB
  - Node.hpp, [27](#)
- Section
  - Node.hpp, [27](#)
- Sentence
  - Node.hpp, [26](#)
- Sentences
  - Node.hpp, [26](#)
- sentences
  - parser.y, [44](#)
- setValue
  - Node, [13](#)
  - parser.y, [32](#), [33](#)
- Sout
  - Node.hpp, [27](#)
- sout
  - lexer.l, [16](#)
- SPACE
  - parser.y, [44](#)
- Space
  - Node.hpp, [27](#)
- st
  - parser.y, [45](#)
- Start
  - Node.hpp, [27](#)
- start
  - parser.y, [45](#)
- str
  - parser.y, [45](#)
- Subsection
  - Node.hpp, [27](#)
- Subsubsection
  - Node.hpp, [27](#)
- symbol
  - Node.hpp, [26](#)
- Symbols
  - Node.hpp, [26](#)
- Table
  - Node.hpp, [27](#)
- tcol
  - parser.y, [45](#)
- Tcontent
  - Node.hpp, [26](#)
- temp
  - parser.y, [45](#)
- Text
  - Node.hpp, [27](#)
- Thead
  - Node.hpp, [27](#)
- Tline
  - Node.hpp, [26](#)
- Tlines
  - Node.hpp, [26](#)
- tlines
  - parser.y, [46](#)
- throw
  - parser.y, [46](#)
- tstr
  - parser.y, [46](#)
- tstruct
  - Node, [14](#)
  - parser.y, [46](#)
- Tstructure
  - Node.hpp, [26](#)
- tstructure
  - parser.y, [46](#)
- Unoitem
  - Node.hpp, [26](#)
- unoitem
  - parser.y, [46](#)
- Url
  - Node.hpp, [27](#)
- WORD
  - lexer.l, [16](#)

---

- yydebug
  - main.cpp, [21](#)
- yylex
  - main.cpp, [20](#)
  - parser.y, [33](#)
- yyparse
  - main.cpp, [20](#)
- yywrap
  - lexer.l, [16](#)