

## CPSC 4660 Compiler

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Administration Class Reference . . . . .	5
3.1.1	Constructor & Destructor Documentation . . . . .	6
3.1.1.1	Administration() . . . . .	6
3.1.2	Member Function Documentation . . . . .	6
3.1.2.1	checkError() . . . . .	6
3.1.2.2	currentLine() . . . . .	7
3.1.2.3	debugInfo() . . . . .	7
3.1.2.4	emit() . . . . .	7
3.1.2.5	error() . . . . .	7
3.1.2.6	error_count() . . . . .	8
3.1.2.7	getToken() . . . . .	8
3.1.2.8	newLine() . . . . .	8
3.1.3	Member Data Documentation . . . . .	8
3.1.3.1	correctLine . . . . .	8
3.1.3.2	debug . . . . .	8
3.1.3.3	errorCount . . . . .	9
3.1.3.4	fout . . . . .	9

3.1.3.5	lineNum	9
3.1.3.6	scanner	9
3.2	Assembler Class Reference	9
3.2.1	Constructor & Destructor Documentation	10
3.2.1.1	Assembler()	10
3.2.1.2	~Assembler()	10
3.2.2	Member Function Documentation	10
3.2.2.1	firstPass()	10
3.2.2.2	secondPass()	10
3.2.3	Member Data Documentation	10
3.2.3.1	currentAddress	11
3.2.3.2	insource	11
3.2.3.3	labelTable	11
3.2.3.4	outsource	11
3.3	BlockTable Class Reference	11
3.3.1	Constructor & Destructor Documentation	12
3.3.1.1	BlockTable()	12
3.3.2	Member Function Documentation	12
3.3.2.1	define() [1/2]	12
3.3.2.2	define() [2/2]	12
3.3.2.3	find()	13
3.3.2.4	level()	13
3.3.2.5	popBlock()	13
3.3.2.6	pushBlock()	13
3.3.2.7	search()	13
3.3.3	Member Data Documentation	14
3.3.3.1	blockLevel	14
3.3.3.2	table	14
3.4	Parser Class Reference	14
3.4.1	Constructor & Destructor Documentation	17

3.4.1.1	Parser()	17
3.4.2	Member Function Documentation	17
3.4.2.1	actParam()	17
3.4.2.2	actParamList()	17
3.4.2.3	addOp()	19
3.4.2.4	assignStmt()	19
3.4.2.5	block()	19
3.4.2.6	boolSym()	20
3.4.2.7	constant()	20
3.4.2.8	constDef()	20
3.4.2.9	cPrime()	20
3.4.2.10	def()	21
3.4.2.11	defPart()	21
3.4.2.12	doStmt()	21
3.4.2.13	emptyStmt()	22
3.4.2.14	expr()	22
3.4.2.15	exprList()	22
3.4.2.16	factor()	22
3.4.2.17	fieldList()	23
3.4.2.18	fieldSelec()	23
3.4.2.19	formParamList()	23
3.4.2.20	guardedComm()	24
3.4.2.21	guardedList()	24
3.4.2.22	idxSelect()	24
3.4.2.23	ifStmt()	25
3.4.2.24	match()	25
3.4.2.25	multOp()	25
3.4.2.26	NewLabel()	26
3.4.2.27	paramDef()	26
3.4.2.28	parse()	26

3.4.2.29	<a href="#">primeExpr()</a>	26
3.4.2.30	<a href="#">primeOp()</a>	26
3.4.2.31	<a href="#">procBlock()</a>	28
3.4.2.32	<a href="#">procDef()</a>	28
3.4.2.33	<a href="#">procStmt()</a>	28
3.4.2.34	<a href="#">program()</a>	29
3.4.2.35	<a href="#">readStmt()</a>	29
3.4.2.36	<a href="#">recordSection()</a>	29
3.4.2.37	<a href="#">relOp()</a>	29
3.4.2.38	<a href="#">selec()</a>	31
3.4.2.39	<a href="#">simpleExpr()</a>	31
3.4.2.40	<a href="#">stmt()</a>	31
3.4.2.41	<a href="#">stmtPart()</a>	32
3.4.2.42	<a href="#">syntaxCheck()</a>	32
3.4.2.43	<a href="#">syntaxError()</a>	32
3.4.2.44	<a href="#">term()</a>	32
3.4.2.45	<a href="#">typeSym()</a>	33
3.4.2.46	<a href="#">vacList()</a>	33
3.4.2.47	<a href="#">varAccess()</a>	33
3.4.2.48	<a href="#">varDef()</a>	34
3.4.2.49	<a href="#">varList()</a>	34
3.4.2.50	<a href="#">vPrime()</a>	34
3.4.2.51	<a href="#">writeStmt()</a>	34
3.4.3	<a href="#">Member Data Documentation</a>	35
3.4.3.1	<a href="#">admin</a>	35
3.4.3.2	<a href="#">blocks</a>	35
3.4.3.3	<a href="#">label</a>	35
3.4.3.4	<a href="#">look</a>	35
3.5	<a href="#">Scanner Class Reference</a>	35
3.5.1	<a href="#">Constructor &amp; Destructor Documentation</a>	36

3.5.1.1	Scanner()	36
3.5.1.2	~Scanner()	37
3.5.2	Member Function Documentation	37
3.5.2.1	getToken()	37
3.5.2.2	isSpecial()	37
3.5.2.3	isWhitespace()	37
3.5.2.4	recognizeName()	38
3.5.2.5	recognizeNumeral()	38
3.5.2.6	recognizeSpecial()	38
3.5.3	Member Data Documentation	38
3.5.3.1	fin	39
3.5.3.2	line	39
3.5.3.3	pos	39
3.5.3.4	symTab	39
3.6	SymbolTable Class Reference	39
3.6.1	Constructor & Destructor Documentation	40
3.6.1.1	SymbolTable()	40
3.6.2	Member Function Documentation	40
3.6.2.1	full()	40
3.6.2.2	getLoad()	40
3.6.2.3	getToken()	40
3.6.2.4	hash()	41
3.6.2.5	insert()	41
3.6.2.6	loadKey()	42
3.6.2.7	loadKeywords()	42
3.6.2.8	probe()	42
3.6.2.9	search()	43
3.6.2.10	toString()	43
3.6.3	Member Data Documentation	43
3.6.3.1	load	43

3.6.3.2	table	43
3.7	TableEntry Class Reference	44
3.7.1	Constructor & Destructor Documentation	44
3.7.1.1	TableEntry() [1/2]	44
3.7.1.2	TableEntry() [2/2]	44
3.7.2	Member Function Documentation	45
3.7.2.1	findEntry() [1/2]	45
3.7.2.2	findEntry() [2/2]	45
3.7.3	Member Data Documentation	45
3.7.3.1	displace	45
3.7.3.2	entries	46
3.7.3.3	id	46
3.7.3.4	level	46
3.7.3.5	size	46
3.7.3.6	startLabel	46
3.7.3.7	tkind	46
3.7.3.8	ttype	47
3.7.3.9	val	47
3.8	Token Class Reference	47
3.8.1	Constructor & Destructor Documentation	48
3.8.1.1	Token() [1/3]	48
3.8.1.2	Token() [2/3]	48
3.8.1.3	Token() [3/3]	48
3.8.2	Member Function Documentation	48
3.8.2.1	getLexeme()	49
3.8.2.2	getSymbol()	49
3.8.2.3	getVal()	49
3.8.2.4	setLexeme()	49
3.8.2.5	setSymbol()	49
3.8.2.6	setVal()	50
3.8.2.7	toString()	50
3.8.3	Member Data Documentation	50
3.8.3.1	lexeme	50
3.8.3.2	sname	50
3.8.3.3	val	50



<b>4</b>	<b>File Documentation</b>	<b>51</b>
4.1	Administration.h File Reference	51
4.1.1	Variable Documentation	51
4.1.1.1	MAX_ERRORS	51
4.2	Assembler.h File Reference	51
4.2.1	Variable Documentation	52
4.2.1.1	MAXLABEL	52
4.3	BlockTable.h File Reference	52
4.3.1	Macro Definition Documentation	52
4.3.1.1	MAXBLOCK	52
4.4	Grammar.h File Reference	53
4.4.1	Enumeration Type Documentation	53
4.4.1.1	NT	53
4.4.2	Function Documentation	55
4.4.2.1	in()	55
4.4.2.2	munion()	55
4.4.3	Variable Documentation	55
4.4.3.1	First	55
4.5	Parser.h File Reference	56
4.6	Scanner.h File Reference	56
4.7	Symbol.h File Reference	56
4.7.1	Enumeration Type Documentation	57
4.7.1.1	Symbol	57
4.7.2	Variable Documentation	58
4.7.2.1	SpecialSym	58
4.7.2.2	SymbolToString	59
4.7.2.3	WordSym	59
4.8	SymbolTable.h File Reference	59
4.8.1	Variable Documentation	60
4.8.1.1	ID_MAX_CHARS	60
4.8.1.2	MOD	60
4.8.1.3	PRIME	60
4.9	TableEntry.h File Reference	60
4.10	Token.h File Reference	60
4.11	Types.h File Reference	61
4.11.1	Enumeration Type Documentation	61
4.11.1.1	Kind	61
4.11.1.2	Type	61
4.11.2	Variable Documentation	62
4.11.2.1	KindToString	62
4.11.2.2	TypeToString	62
	<b>Index</b>	<b>63</b>



# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">Administration</a>	5
<a href="#">Assembler</a>	9
<a href="#">BlockTable</a>	11
<a href="#">Parser</a>	14
<a href="#">Scanner</a>	35
<a href="#">SymbolTable</a>	39
<a href="#">TableEntry</a>	44
<a href="#">Token</a>	47



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Administration.h</a>	51
<a href="#">Assembler.h</a>	51
<a href="#">BlockTable.h</a>	52
<a href="#">Grammar.h</a>	53
<a href="#">Parser.h</a>	56
<a href="#">Scanner.h</a>	56
<a href="#">Symbol.h</a>	56
<a href="#">SymbolTable.h</a>	59
<a href="#">TableEntry.h</a>	60
<a href="#">Token.h</a>	60
<a href="#">Types.h</a>	61



## Chapter 3

# Class Documentation

### 3.1 Administration Class Reference

```
#include <Administration.h>
```

#### Public Member Functions

- `Administration` (`std::ostream &fout`, `Scanner &sc`, `bool debug=false`)  
*Creates a new `Administration` object.*
- `int currentLine ()`
- `Token getToken ()`
- `void newLine ()`  
*Adds line number and resets `correctLine`.*
- `void debugInfo (std::string text)`  
*Print debugging info to the console if in debug mode.*
- `void error (std::string text)`  
*Display text for an error.*
- `void emit (std::string text, int var=-1, int start=-1)`  
*Emit assembly code to the output file.*
- `int error_count ()`  
*Return the number of errors.*

#### Private Member Functions

- `void checkError (Token ntoken)`  
*Checks if current token is an error token.*

## Private Attributes

- `std::ostream & fout`  
*File to print all tokens to.*
- `Scanner & scanner`  
*The scanner to use on the input.*
- `int lineNum`  
*The current line number.*
- `bool correctLine`  
*True if the line has no errors so far.*
- `int errorCount`  
*The total number of errors so far.*
- `bool debug`  
*Wether or not to print debugging info.*

## 3.1.1 Constructor & Destructor Documentation

### 3.1.1.1 Administration()

```
Administration::Administration (
    std::ostream & fout,
    Scanner & sc,
    bool debug = false )
```

Creates a new `Administration` object.

#### Parameters

<i>fout</i>	The output file stream.
<i>sc</i>	The scanner beign used by administration.
<i>debug</i>	Set debug mode. Default false.

## 3.1.2 Member Function Documentation

### 3.1.2.1 checkError()

```
void Administration::checkError (
    Token ntoken ) [private]
```

Checks if current token is an error token.



## Parameters

<i>ntoken</i>	The current token.
---------------	--------------------

3.1.2.2 `currentLine()`

```
int Administration::currentLine ( ) [inline]
```

3.1.2.3 `debugInfo()`

```
void Administration::debugInfo (
    std::string text )
```

Print debugging info to the console if in debug mode.

## Parameters

<i>text</i>	The info to print.
-------------	--------------------

3.1.2.4 `emit()`

```
void Administration::emit (
    std::string text,
    int var = -1,
    int start = -1 )
```

Emit assembly code to the output file.

3.1.2.5 `error()`

```
void Administration::error (
    std::string text )
```

Display text for an error.

## Parameters

<i>text</i>	The error message.
-------------	--------------------

#### 3.1.2.6 error\_count()

```
int Administration::error_count ( ) [inline]
```

Return the number of errors.

#### 3.1.2.7 getToken()

```
Token Administration::getToken ( )
```

#### 3.1.2.8 newLine()

```
void Administration::newLine ( )
```

Adds line number and resets correctLine.

### 3.1.3 Member Data Documentation

#### 3.1.3.1 correctLine

```
bool Administration::correctLine [private]
```

True if the line has no errors so far.

#### 3.1.3.2 debug

```
bool Administration::debug [private]
```

Wether or not to print debugging info.

### 3.1.3.3 errorCount

```
int Administration::errorCount [private]
```

The total number of errors so far.

### 3.1.3.4 fout

```
std::ostream& Administration::fout [private]
```

File to print all tokens to.

### 3.1.3.5 lineNum

```
int Administration::lineNum [private]
```

The current line number.

### 3.1.3.6 scanner

```
Scanner& Administration::scanner [private]
```

The scanner to use on the input.

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

- [Administration.h](#)

## 3.2 Assembler Class Reference

```
#include <Assembler.h>
```

### Public Member Functions

- [Assembler](#) (istream &[in](#), ostream &out)
- [~Assembler](#) ()
- void [firstPass](#) ()
- void [secondPass](#) ()

## Private Attributes

- int [labelTable](#) [[MAXLABEL](#)]
- int [currentAddress](#)
- istream \* [insource](#)
- ostream \* [outsource](#)

## 3.2.1 Constructor & Destructor Documentation

### 3.2.1.1 Assembler()

```
Assembler::Assembler (
    istream & in,
    ostream & out )
```

### 3.2.1.2 ~Assembler()

```
Assembler::~~Assembler ( )
```

## 3.2.2 Member Function Documentation

### 3.2.2.1 firstPass()

```
void Assembler::firstPass ( )
```

### 3.2.2.2 secondPass()

```
void Assembler::secondPass ( )
```

## 3.2.3 Member Data Documentation

## 3.2.3.1 currentAddress

```
int Assembler::currentAddress [private]
```

## 3.2.3.2 insource

```
istream* Assembler::insource [private]
```

## 3.2.3.3 labelTable

```
int Assembler::labelTable[MAXLABEL] [private]
```

## 3.2.3.4 outsource

```
ostream* Assembler::outsource [private]
```

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

- [Assembler.h](#)

## 3.3 BlockTable Class Reference

```
#include <BlockTable.h>
```

### Public Member Functions

- [BlockTable](#) ()  
*Default Constructor for a [BlockTable](#).*
- bool [search](#) (int lookId)  
*Searches the current level of the blocktable for a table entry.*
- bool [define](#) (int nid, [Kind](#) nkind, [Type](#) ntype, int nsize, int nval, int displace)  
*Creates a new table entry and puts it into the current block if it doesnt already exist.*
- bool [define](#) ([TableEntry](#) &entry)  
*Overloaded define function that takes in a table entry to define.*
- [TableEntry](#) [find](#) (int lookId, bool &error)  
*Searches the entire blocktable for the table entry.*
- bool [pushBlock](#) ()  
*Creates and pushes a new blocktable onto the current block.*
- void [popBlock](#) ()  
*Removes the highest level (most recent) block of the blocktable.*
- int [level](#) ()  
*The current block level.*

## Private Attributes

- `std::vector< std::map< int, TableEntry > > table`  
*Vector of maps storing the table entries for a block (the block table)*
- `int blockLevel`  
*The current blocklevel.*

## 3.3.1 Constructor & Destructor Documentation

### 3.3.1.1 [BlockTable\(\)](#)

```
BlockTable::BlockTable ( )
```

Default Constructor for a [BlockTable](#).

## 3.3.2 Member Function Documentation

### 3.3.2.1 [define\(\)](#) [1/2]

```
bool BlockTable::define (
    int nid,
    Kind nkind,
    Type ntype,
    int nsize,
    int nval,
    int displace )
```

Creates a new table entry and puts it into the current block if it doesnt already exist.

#### Parameters

<i>nid</i>	The id of the table entry
<i>nkind</i>	The kind of the table entry
<i>ntype</i>	The type of the table entry
<i>nsize</i>	The memory size required by the table entry
<i>nval</i>	The value of the table entry

### 3.3.2.2 [define\(\)](#) [2/2]

```
bool BlockTable::define (
    TableEntry & entry )
```

Overloaded define function that takes in a table entry to define.

#### Parameters

<i>entry</i>	The table entry that will be define
--------------	-------------------------------------

#### 3.3.2.3 find()

```
TableEntry BlockTable::find (
    int lookId,
    bool & error )
```

Searches the entire blocktable for the table entry.

#### Parameters

<i>lookId</i>	The id of the table entry being searched for
<i>error</i>	The error check for when the table entry does not exist

#### 3.3.2.4 level()

```
int BlockTable::level ( ) [inline]
```

The current block level.

#### 3.3.2.5 popBlock()

```
void BlockTable::popBlock ( )
```

Removes the highest level (most recent) block of the blocktable.

#### 3.3.2.6 pushBlock()

```
bool BlockTable::pushBlock ( )
```

Creates and pushes a new blocktable onto the current block.

#### 3.3.2.7 search()

```
bool BlockTable::search (
    int lookId )
```

Searches the current level of the blocktable for a table entry.

#### Parameters

<i>lookID</i>	The id of the table entry being searched for
---------------	--

### 3.3.3 Member Data Documentation

#### 3.3.3.1 blockLevel

```
int BlockTable::blockLevel [private]
```

The current blocklevel.

#### 3.3.3.2 table

```
std::vector<std::map<int, TableEntry> > BlockTable::table [private]
```

Vector of maps storing the table entries for a block (the block table)

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

- [BlockTable.h](#)

## 3.4 Parser Class Reference

```
#include <Parser.h>
```

#### Public Member Functions

- [Parser](#) ([Administration](#) &admin)  
*Creates a new [Parser](#) object.*
- void [parse](#) ()  
*Parses a PL program.*



## Private Member Functions

- int `NewLabel` ()
- void `match` (`Symbol` symbol, `std::set`< `Symbol` > stop)
 

*Match a `Token` and move to the next one.*
- void `syntaxError` (`std::set`< `Symbol` > stop)
 

*Process a syntax error and perform error recovery.*
- void `syntaxCheck` (`std::set`< `Symbol` > stop)
 

*Checks the next token to see if it will be valid.*
- void `program` (`std::set`< `Symbol` > stop)
 

*Parses a program from the stream of tokens.*
- void `block` (`std::set`< `Symbol` > stop, `std::vector`< `TableEntry` > entries, int startlabel, int varlabel)
 

*Parses a block from the stream of tokens.*
- int `defPart` (`std::set`< `Symbol` > stop)
 

*Parses a definition part from the stream of tokens.*
- int `def` (`std::set`< `Symbol` > stop, int &start)
 

*Parses a definition from the stream of tokens.*
- void `constDef` (`std::set`< `Symbol` > stop)
 

*Parses a constant definitions from the stream of tokens.*
- void `procDef` (`std::set`< `Symbol` > stop)
 

*Parses a procedure definition from the stream of tokens.*
- void `stmtPart` (`std::set`< `Symbol` > stop)
 

*Parses the statement part of the program.*
- void `stmt` (`std::set`< `Symbol` > stop)
 

*Parses a statement.*
- void `emptyStmt` (`std::set`< `Symbol` > stop)
 

*Parses an empty statement.*
- void `readStmt` (`std::set`< `Symbol` > stop)
 

*Parses a read statement.*
- void `writeStmt` (`std::set`< `Symbol` > stop)
 

*Parses a write statement.*
- void `assignStmt` (`std::set`< `Symbol` > stop)
 

*Parses an assignment statement.*
- void `procStmt` (`std::set`< `Symbol` > stop)
 

*Parses a procedure call.*
- void `ifStmt` (`std::set`< `Symbol` > stop)
 

*Parses an if statement.*
- void `doStmt` (`std::set`< `Symbol` > stop)
 

*Parses a do statement.*
- `std::vector`< `Type` > `vacsList` (`std::set`< `Symbol` > stop)
 

*Parses a variable access list.*
- `Type` `varAccess` (`std::set`< `Symbol` > stop, bool &isConst)
 

*Parses variable access.*
- int `varDef` (`std::set`< `Symbol` > stop, int &start)
 

*Parses a variable definition from the stream of tokens.*
- int `vPrime` (`std::set`< `Symbol` > stop, `Type` type, int &start)
 

*Parses a variable vs array from the stream of tokens.*
- `std::vector`< int > `varList` (`std::set`< `Symbol` > stop)
 

*Parses a variable list from the stream of tokens.*
- `Type` `idxSelect` (`std::set`< `Symbol` > stop, `TableEntry` entry)
 

*Parses an index selector.*

- `std::vector< Type > exprList (std::set< Symbol > stop)`  
*Parses a expression list from the stream of tokens.*
- `Type expr (std::set< Symbol > stop)`  
*Parses a expression from the stream of tokens.*
- `Type primeExpr (std::set< Symbol > stop)`  
*Parses a primary expression from the stream of tokens.*
- `Type simpleExpr (std::set< Symbol > stop)`  
*Parses a simple expression from the stream of tokens.*
- `void guardedList (std::set< Symbol > stop, int &start, int next)`  
*Parses a list of guarded commands.*
- `void guardedComm (std::set< Symbol > stop, int &start, int next)`  
*Parses a guarded command.*
- `Type term (std::set< Symbol > stop)`  
*Parses a term from the stream of tokens.*
- `Type factor (std::set< Symbol > stop)`  
*Parses a factor from the stream of tokens.*
- `std::string primeOp (std::set< Symbol > stop)`  
*Parses a primary operator from the stream of tokens.*
- `std::string relOp (std::set< Symbol > stop)`  
*Parses a realtional operator from the stream of tokens.*
- `std::string addOp (std::set< Symbol > stop)`  
*Parses a plus or minus operator from the stream of tokens.*
- `std::string multOp (std::set< Symbol > stop)`  
*Parses a multiplication or division or modulus operator from the stream of tokens.*
- `std::pair< Type, int > constant (std::set< Symbol > stop)`  
*Parses a const non-terminal.*
- `Type cPrime (std::set< Symbol > stop)`  
*Parses a const num non-terminal.*
- `Type typeSym (std::set< Symbol > stop)`  
*Parses a definition type from the stream of tokens.*
- `int boolSym (std::set< Symbol > stop)`  
*Parses a true or false from the stream of tokens.*
- `void fieldList (std::set< Symbol > stop, std::vector< TableEntry > &fields)`  
*Parses the a list of all the fields and their corresponding types declared.*
- `void recordSection (std::set< Symbol > stop, std::vector< TableEntry > &fields)`  
*Parses a list of idetifiers of the same type declared in a record.*
- `void procBlock (std::set< Symbol > stop, int id, int start, int var, int proc)`  
*Parses the block for a procedure declaration.*
- `void formParamList (std::set< Symbol > stop, std::vector< TableEntry > &params)`  
*Parses the parameter list when a procdure is being declared.*
- `void paramDef (std::set< Symbol > stop, std::vector< TableEntry > &params)`  
*Parses a list of idetifiers being passed into the procedure, can be tagged with "var" meaning it is pass by reference, pass by value otherwise.*
- `std::vector< Type > actParamList (std::set< Symbol > stop)`  
*Parases the list of parameters when a procedure is being called.*
- `Type actParam (std::set< Symbol > stop)`  
*Parses the individual paramters inside the paramater list when a procedure is called.*
- `Type selec (std::set< Symbol > stop, TableEntry entry)`  
*Parses whether the varaible being accessed is in a record or expression.*
- `Type fieldSelec (std::set< Symbol > stop, TableEntry entry)`  
*Parses field/variable being selected from a record.*

## Private Attributes

- `int label`
- `Administration & admin`  
*The administration object for errors and holding the scanner and symbol table.*
- `Token look`  
*The look ahead token.*
- `BlockTable blocks`

## 3.4.1 Constructor & Destructor Documentation

### 3.4.1.1 Parser()

```
Parser::Parser (
    Administration & admin )
```

Creates a new `Parser` object.

#### Parameters

<code>admin</code>	An administration object for handling errors and holding our scanner etc. for now.
--------------------	--

## 3.4.2 Member Function Documentation

### 3.4.2.1 actParam()

```
Type Parser::actParam (
    std::set< Symbol > stop ) [private]
```

Parses the individual paramters inside the paramater list when a procedure is called.

#### Parameters

<code>stop</code>	The stopsets used to recover from an error.
-------------------	---

### 3.4.2.2 actParamList()

```
std::vector<Type> Parser::actParamList (
    std::set< Symbol > stop ) [private]
```

Parases the list of parameters when a procedure is being called.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.3 addOp()**

```
std::string Parser::addOp (  
    std::set< Symbol > stop ) [private]
```

Parses a plus or minus operator from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.4 assignStmt()**

```
void Parser::assignStmt (  
    std::set< Symbol > stop ) [private]
```

Parses an assignment statement.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.5 block()**

```
void Parser::block (  
    std::set< Symbol > stop,  
    std::vector< TableEntry > entries,  
    int startlabel,  
    int varlabel ) [private]
```

Parses a block from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
<i>entries</i>	The entries being added to the block

#### 3.4.2.6 boolSym()

```
int Parser::boolSym (
    std::set< Symbol > stop ) [private]
```

Parses a true or false from the stream of tokens.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.7 constant()

```
std::pair<Type,int> Parser::constant (
    std::set< Symbol > stop ) [private]
```

Parses a const non-terminal.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.8 constDef()

```
void Parser::constDef (
    std::set< Symbol > stop ) [private]
```

Parses a constant definitions from the stream of tokens.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.9 cPrime()

```
Type Parser::cPrime (
    std::set< Symbol > stop ) [private]
```

Parses a const num non-terminal.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.10 def()

```
int Parser::def (
    std::set< Symbol > stop,
    int & start ) [private]
```

Parses a definition from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.11 defPart()

```
int Parser::defPart (
    std::set< Symbol > stop ) [private]
```

Parses a definition part from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.12 doStmt()

```
void Parser::doStmt (
    std::set< Symbol > stop ) [private]
```

Parses a do statement.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

### 3.4.2.13 emptyStmt()

```
void Parser::emptyStmt (
    std::set< Symbol > stop ) [private]
```

Parses an empty statement.

#### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

### 3.4.2.14 expr()

```
Type Parser::expr (
    std::set< Symbol > stop ) [private]
```

Parses a expression from the stream of tokens.

#### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

### 3.4.2.15 exprList()

```
std::vector<Type> Parser::exprList (
    std::set< Symbol > stop ) [private]
```

Parses a expression list from the stream of tokens.

#### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

### 3.4.2.16 factor()

```
Type Parser::factor (
    std::set< Symbol > stop ) [private]
```

Parses a factor from the stream of tokens.



## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.17 fieldList()

```
void Parser::fieldList (
    std::set< Symbol > stop,
    std::vector< TableEntry > & fields ) [private]
```

Parses the a list of all the fields and their corresponding types declared.

in a record.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>fields</i>	The field of the record being declared.

## 3.4.2.18 fieldSelec()

```
Type Parser::fieldSelec (
    std::set< Symbol > stop,
    TableEntry entry ) [private]
```

Parses field/variable being selected from a record.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>entry</i>	The table entry of the record being accessed.

## 3.4.2.19 formParamList()

```
void Parser::formParamList (
    std::set< Symbol > stop,
    std::vector< TableEntry > & params ) [private]
```

Parses the parameter list when a procedure is being declared.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>params</i>	The parameters of the procedure being defined.

## 3.4.2.20 guardedComm()

```
void Parser::guardedComm (
    std::set< Symbol > stop,
    int & start,
    int next ) [private]
```

Parses a guarded command.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.21 guardedList()

```
void Parser::guardedList (
    std::set< Symbol > stop,
    int & start,
    int next ) [private]
```

Parses a list of guarded commands.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.22 idxSelect()

```
Type Parser::idxSelect (
    std::set< Symbol > stop,
    TableEntry entry ) [private]
```

Parses an index selector.

ie) A[i].

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>entry</i>	The Table entry being created

## 3.4.2.23 ifStmt()

```
void Parser::ifStmt (
    std::set< Symbol > stop ) [private]
```

Parses an if statement.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.24 match()

```
void Parser::match (
    Symbol symbol,
    std::set< Symbol > stop ) [private]
```

Match a [Token](#) and move to the next one.

## Parameters

<i>symbol</i>	The symbol being matched
<i>stop</i>	The stopsets used to recover from the error.

## 3.4.2.25 multOp()

```
std::string Parser::multOp (
    std::set< Symbol > stop ) [private]
```

Parses a multiplication or division or modulus operator from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.26 NewLabel()**

```
int Parser::NewLabel ( ) [private]
```

**3.4.2.27 paramDef()**

```
void Parser::paramDef (
    std::set< Symbol > stop,
    std::vector< TableEntry > & params ) [private]
```

Parses a list of identifiers being passed into the procedure, can be tagged with "var" meaning it is pass by reference, pass by value otherwise.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
<i>params</i>	The parameters of the procedure being defined.

**3.4.2.28 parse()**

```
void Parser::parse ( )
```

Parses a PL program.

**3.4.2.29 primeExpr()**

```
Type Parser::primeExpr (
    std::set< Symbol > stop ) [private]
```

Parses a primary expression from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.30 primeOp()**

```
std::string Parser::primeOp (
    std::set< Symbol > stop ) [private]
```

Parses a primary operator from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.31 procBlock()**

```
void Parser::procBlock (
    std::set< Symbol > stop,
    int id,
    int start,
    int var,
    int proc ) [private]
```

Parses the block for a procedure declaration.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
<i>id</i>	The id of the procedure.

**3.4.2.32 procDef()**

```
void Parser::procDef (
    std::set< Symbol > stop ) [private]
```

Parses a procedure definition from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.33 procStmt()**

```
void Parser::procStmt (
    std::set< Symbol > stop ) [private]
```

Parses a procedure call.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.34 program()

```
void Parser::program (
    std::set< Symbol > stop ) [private]
```

Parses a program from the stream of tokens.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.35 readStmt()

```
void Parser::readStmt (
    std::set< Symbol > stop ) [private]
```

Parses a read statement.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.36 recordSection()

```
void Parser::recordSection (
    std::set< Symbol > stop,
    std::vector< TableEntry > & fields ) [private]
```

Parses a list of identifiers of the same type declared in a record.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>fields</i>	The field of the record being declared.

#### 3.4.2.37 relOp()

```
std::string Parser::relOp (
    std::set< Symbol > stop ) [private]
```

Parses a relational operator from the stream of tokens.



## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

3.4.2.38 `selec()`

```
Type Parser::selec (  
    std::set< Symbol > stop,  
    TableEntry entry ) [private]
```

Parses whether the variable being accessed is in a record or expression.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
<i>entry</i>	The table entry of the record being accessed.

3.4.2.39 `simpleExpr()`

```
Type Parser::simpleExpr (  
    std::set< Symbol > stop ) [private]
```

Parses a simple expression from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

3.4.2.40 `stmt()`

```
void Parser::stmt (  
    std::set< Symbol > stop ) [private]
```

Parses a statement.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.41 stmtPart()

```
void Parser::stmtPart (
    std::set< Symbol > stop ) [private]
```

Parses the statement part of the program.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.42 syntaxCheck()

```
void Parser::syntaxCheck (
    std::set< Symbol > stop ) [private]
```

Checks the next token to see if it will be valid.

##### Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

#### 3.4.2.43 syntaxError()

```
void Parser::syntaxError (
    std::set< Symbol > stop ) [private]
```

Process a syntax error and perform error recovery.

##### Parameters

<i>stop</i>	The stopsets used to recover from the error.
-------------	--

#### 3.4.2.44 term()

```
Type Parser::term (
    std::set< Symbol > stop ) [private]
```

Parses a term from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.45 typeSym()

```
Type Parser::typeSym (
    std::set< Symbol > stop ) [private]
```

Parses a definition type from the stream of tokens.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.46 vacsList()

```
std::vector<Type> Parser::vacsList (
    std::set< Symbol > stop ) [private]
```

Parses a variable access list.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

## 3.4.2.47 varAccess()

```
Type Parser::varAccess (
    std::set< Symbol > stop,
    bool & isConst ) [private]
```

Parses variable access.

## Parameters

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.48 varDef()**

```
int Parser::varDef (
    std::set< Symbol > stop,
    int & start ) [private]
```

Parses a variable definition from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.49 varList()**

```
std::vector<int> Parser::varList (
    std::set< Symbol > stop ) [private]
```

Parses a variable list from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
-------------	---

**3.4.2.50 vPrime()**

```
int Parser::vPrime (
    std::set< Symbol > stop,
    Type type,
    int & start ) [private]
```

Parses a variable vs array from the stream of tokens.

**Parameters**

<i>stop</i>	The stopsets used to recover from an error.
<i>type</i>	The type of the table entry that is being created

**3.4.2.51 writeStmt()**

```
void Parser::writeStmt (
    std::set< Symbol > stop ) [private]
```

Parses a write statement.

## Parameters

<code>stop</code>	The stopsets used to recover from an error.
-------------------	---

### 3.4.3 Member Data Documentation

#### 3.4.3.1 admin

```
Administration& Parser::admin [private]
```

The administration object for errors and holding the scanner and symbol table.

#### 3.4.3.2 blocks

```
BlockTable Parser::blocks [private]
```

#### 3.4.3.3 label

```
int Parser::label [private]
```

#### 3.4.3.4 look

```
Token Parser::look [private]
```

The look ahead token.

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

- [Parser.h](#)

## 3.5 Scanner Class Reference

```
#include <Scanner.h>
```

## Public Member Functions

- [Scanner](#) (std::istream &ifs, [SymbolTable](#) &symTab)  
*Constructor for the scanner, initializes the private variables to appropriate values.*
- [~Scanner](#) ()  
*Destructor of the scanner.*
- [Token getToken](#) ()  
*Get the next [Token](#) in the line.*

## Private Member Functions

- bool [isWhitespace](#) (char inchar)  
*Checks the input symbol against Whitespace whether tab or space.*
- bool [isSpecial](#) (char inchar)  
*Checks the input char against all possible symbols.*
- [Token recognizeName](#) ()  
*Read and generate tokens for keywords and ID's, also checks for invalid characters and returns a CHAR\_ERR token and checks the symbol table is filled then return a FULL\_TAB error token.*
- [Token recognizeSpecial](#) ()  
*Read and generate a token for any of the special symbols.*
- [Token recognizeNumeral](#) ()  
*Read and generate a token for any number/digit.*

## Private Attributes

- std::istream & [fin](#)  
*The file stream.*
- [SymbolTable](#) & [symTab](#)  
*The symbol table.*
- std::string [line](#)  
*The current line the scanner is reading.*
- std::size\_t [pos](#)  
*The position of the char the scanner is reading.*

### 3.5.1 Constructor & Destructor Documentation

#### 3.5.1.1 Scanner()

```
Scanner::Scanner (
    std::istream & ifs,
    SymbolTable & symTab )
```

Constructor for the scanner, initializes the private variables to appropriate values.

## Parameters

<i>ifs</i>	The file stream.
<i>symTab</i>	The symbol table

## 3.5.1.2 ~Scanner()

```
Scanner::~Scanner ( ) [inline]
```

Destructor of rthe scanner.

## 3.5.2 Member Function Documentation

## 3.5.2.1 getToken()

```
Token Scanner::getToken ( )
```

Get the next [Token](#) in the line.

## 3.5.2.2 isSpecial()

```
bool Scanner::isSpecial (
    char inchar ) [private]
```

Checks the inputed char against all possible symbols.

## Parameters

<i>inchar</i>	The current char being read in
---------------	--------------------------------

## Returns

true if the char is a special symbol, false otherwise.

## 3.5.2.3 isWhitespace()

```
bool Scanner::isWhitespace (
    char inchar ) [private]
```

Checks the input symbol against Whitespace whether tab or space.

**Parameters**

<i>inchar</i>	The current char being read in
---------------	--------------------------------

**Returns**

true if the char is whitespace, false otherwise.

**3.5.2.4 recognizeName()**

```
Token Scanner::recognizeName ( ) [private]
```

Read and generate tokens for keywords and ID's, also checks for invalid characters and returns a CHAR\_ERR token and checks the symbol table is filled then return a FULL\_TAB error token.

**Returns**

An ID or keyword token for the scanned lexeme, or an error token.

**3.5.2.5 recognizeNumeral()**

```
Token Scanner::recognizeNumeral ( ) [private]
```

Read and generate a token for any number/digit.

**Returns**

a token for the number with the actual value in it.

**3.5.2.6 recognizeSpecial()**

```
Token Scanner::recognizeSpecial ( ) [private]
```

Read and generate a token for any of the special symbols.

**Returns**

a token for the special symbol scanned.

**3.5.3 Member Data Documentation**



### 3.5.3.1 fin

```
std::istream& Scanner::fin [private]
```

The file stream.

### 3.5.3.2 line

```
std::string Scanner::line [private]
```

The current line the scanner is reading.

### 3.5.3.3 pos

```
std::size_t Scanner::pos [private]
```

The position of the char the scanner is reading.

### 3.5.3.4 symTab

```
SymbolTable& Scanner::symTab [private]
```

The symbol table.

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

- [Scanner.h](#)

## 3.6 SymbolTable Class Reference

```
#include <SymbolTable.h>
```

### Public Member Functions

- [SymbolTable](#) ()
- int [search](#) (const std::string &str)  
*Searches for a lexeme in the symbol table and returns its token.*
- int [insert](#) (const std::string &str)  
*Inserts a new lexeme into the symbol table if it is not already there.*
- [Token](#) & [getToken](#) (int idx, bool &found)  
*Get a reference to the token in the symbol table by its index.*
- int [hash](#) (const std::string &str)  
*Computes a rolling hash for a given string using the MOD constant.*
- bool [full](#) ()  
*Returns true if the table is full.*
- int [getLoad](#) ()  
*Returns the number items in the table.*
- std::string [toString](#) ()  
*Returns a string representation of the table.*

## Private Member Functions

- `std::pair< int, Token & > probe` (int idx, std::string lexeme)  
*Given a position linear probe until the token with the given lexeme is found or an empty token is found.*
- `void loadKey` (Symbol sym, const std::string &lexeme)  
*Load a token for a reserved keyword into the table.*
- `void loadKeywords` ()  
*Loads all reserved keywords into the symbol table.*

## Private Attributes

- `std::vector< Token > table`  
*Backing array for the hash table.*
- `int load`  
*The number of elements in the hash table.*

## 3.6.1 Constructor & Destructor Documentation

### 3.6.1.1 SymbolTable()

```
SymbolTable::SymbolTable ( )
```

## 3.6.2 Member Function Documentation

### 3.6.2.1 full()

```
bool SymbolTable::full ( )
```

Returns true if the table is full.

### 3.6.2.2 getLoad()

```
int SymbolTable::getLoad ( )
```

Returns the number items in the table.

### 3.6.2.3 getToken()

```
Token& SymbolTable::getToken (
    int idx,
    bool & found )
```

Get a reference to the token in the symbol table by its index.

## Parameters

<i>idx</i>	The index of the token.
<i>found</i>	

## Returns

a reference to the token or a dummy empty token.

## Exceptions

<i>out_of_range</i>	error if the <i>idx</i> is out of bounds.
---------------------	---

## 3.6.2.4 hash()

```
int SymbolTable::hash (
    const std::string & str )
```

Computes a rolling hash for a given string using the MOD constant.

Only looks at a max of 10 characters from the string.

## Parameters

<i>str</i>	The string to hash.
------------	---------------------

## Returns

the integer hash value of the string.

## 3.6.2.5 insert()

```
int SymbolTable::insert (
    const std::string & str )
```

Inserts a new lexeme into the symbol table if it is not already there.

## Parameters

<i>str</i>	Insert a string into the hash table.
------------	--------------------------------------

**Returns**

The index of the token in the symbol table, or -1 if it exists.

**Exceptions**

<i>length_error</i>	if the symbol table is full.
---------------------	------------------------------

**3.6.2.6 loadKey()**

```
void SymbolTable::loadKey (
    Symbol sym,
    const std::string & lexeme ) [private]
```

Load a token for a reserved keyword into the table.

**Parameters**

<i>lexeme</i>	The tokens's lexeme.
<i>sym</i>	The token's symbol.

**3.6.2.7 loadKeywords()**

```
void SymbolTable::loadKeywords ( ) [private]
```

Loads all reserved keywords into the symbol table.

**3.6.2.8 probe()**

```
std::pair<int, Token> SymbolTable::probe (
    int idx,
    std::string lexeme ) [private]
```

Given a position linear probe until the token with the given lexeme is found or an empty token is found.

**Parameters**

<i>idx</i>	The initial position to start probing. Generally the lexemes hash value.
<i>lexeme</i>	The lexeme to probe for.

**Returns**

a pair with the position of the token and the lexeme.

**3.6.2.9 search()**

```
int SymbolTable::search (
    const std::string & str )
```

Searches for a lexeme in the symbol table and returns its token.

**Parameters**

<i>str</i>	The lexeme to search for.
------------	---------------------------

**Returns**

The index of the token in the symbol table, or -1 for not found.

**3.6.2.10 toString()**

```
std::string SymbolTable::toString ( )
```

Returns a string representation of the table.

**3.6.3 Member Data Documentation****3.6.3.1 load**

```
int SymbolTable::load [private]
```

The number of elements in the hash table.

**3.6.3.2 table**

```
std::vector<Token> SymbolTable::table [private]
```

Backing array for the hash table.

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

- [SymbolTable.h](#)

## 3.7 TableEntry Class Reference

```
#include <TableEntry.h>
```

### Public Member Functions

- [TableEntry](#) ()  
*Default Constructor that creates a empty table entry set to default values.*
- [TableEntry](#) (int nid, [Kind](#) nkind, [Type](#) ntype, int nsize, int nval, int disp)  
*Overloaded constructor that creates the table entry with the input values.*
- int [findEntry](#) ([TableEntry](#) &entry)  
*Check if the table entry input is a param or field of a procedure or record.*
- int [findEntry](#) (int id)  
*Overloaded function to check if a table entry is a param or field using its id of a procedure or record.*

### Public Attributes

- int [id](#)  
*The table entry id.*
- [Kind](#) [tkind](#)  
*The kind of table entry.*
- [Type](#) [ttype](#)  
*The type of the table entry.*
- int [size](#)  
*The size of the required memory for the table entry.*
- int [val](#)  
*The value of the table entry.*
- std::vector< [TableEntry](#) > [entries](#)  
*The field/params of a record/procedure respectively.*
- int [level](#)
- int [displace](#)
- int [startLabel](#)

### 3.7.1 Constructor & Destructor Documentation

#### 3.7.1.1 [TableEntry](#)() [1/2]

```
TableEntry::TableEntry ( ) [inline]
```

Default Constructor that creates a empty table entry set to default values.

#### 3.7.1.2 [TableEntry](#)() [2/2]

```
TableEntry::TableEntry (
    int nid,
    Kind nkind,
    Type ntype,
    int nsize,
    int nval,
    int disp ) [inline]
```

Overloaded constructor that creates the table entry with the input values.

## Parameters

<i>nid</i>	The id of the table entry
<i>nkind</i>	The Kind of the table entry
<i>ntype</i>	The Type of the table entry
<i>nsize</i>	The memory size required by the table entry
<i>nval</i>	The value of the table entry

## 3.7.2 Member Function Documentation

### 3.7.2.1 findEntry() [1/2]

```
int TableEntry::findEntry (
    TableEntry & entry ) [inline]
```

Check if the table entry input is a param or field of a procedure or record.

## Parameters

<i>entry</i>	The table entry being searched for
--------------	------------------------------------

### 3.7.2.2 findEntry() [2/2]

```
int TableEntry::findEntry (
    int id ) [inline]
```

Overloaded function to check if a table entry is a param or field using its id of a procedure or record.

## Parameters

<i>The</i>	id of the table entry being searched for
------------	--

## 3.7.3 Member Data Documentation

### 3.7.3.1 displace

```
int TableEntry::displace
```

### 3.7.3.2 entries

```
std::vector<TableEntry> TableEntry::entries
```

The field/params of a record/procedure respectively.

### 3.7.3.3 id

```
int TableEntry::id
```

The table entry id.

### 3.7.3.4 level

```
int TableEntry::level
```

### 3.7.3.5 size

```
int TableEntry::size
```

The size of the required memory for the table entry.

### 3.7.3.6 startLabel

```
int TableEntry::startLabel
```

### 3.7.3.7 tkind

```
Kind TableEntry::tkind
```

The kind of table entry.



### 3.7.3.8 ttype

Type TableEntry::ttype

The type of the table entry.

### 3.7.3.9 val

int TableEntry::val

The value of the table entry.

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

- [TableEntry.h](#)

## 3.8 Token Class Reference

```
#include <Token.h>
```

### Public Member Functions

- [Token](#) ()  
*Creates a new default token.*
- [Token](#) ([Symbol](#) sym, std::string [lexeme](#)="", int [val](#)=-1)  
*Creates a new token.*
- [Token](#) (const [Token](#) &tok)  
*Copy Constructor.*
- [Symbol](#) [getSymbol](#) () const  
*Returns the symbol.*
- std::string [getLexeme](#) () const  
*Returns the lexeme.*
- int [getVal](#) () const  
*Returns the value.*
- void [setSymbol](#) ([Symbol](#) sym)  
*Sets the symbol.*
- void [setLexeme](#) (std::string [lexeme](#))  
*Sets the lexeme.*
- void [setVal](#) (int [val](#))  
*Sets the value.*
- std::string [toString](#) ()  
*Returns a string representation of the [Token](#).*

## Private Attributes

- [Symbol](#) `sname`  
*The token's symbol.*
- `std::string` [lexeme](#)  
*The tokens lexeme.*
- `int` [val](#)  
*The numeric value of the token.*

## 3.8.1 Constructor & Destructor Documentation

### 3.8.1.1 `Token()` [1/3]

```
Token::Token ( )
```

Creates a new default token.

Sets Symbol to EMPTY, lexeme to "", and value to -1.

### 3.8.1.2 `Token()` [2/3]

```
Token::Token (
    Symbol sym,
    std::string lexeme = "",
    int val = -1 )
```

Creates a new token.

#### Parameters

<code>sym</code>	The symbol for the token.
<code>lexeme</code>	The lexeme for the token. Default "".
<code>val</code>	The numerical value to give to the token. Default -1.

### 3.8.1.3 `Token()` [3/3]

```
Token::Token (
    const Token & tok )
```

Copy Constructor.

## 3.8.2 Member Function Documentation

### 3.8.2.1 getLexeme()

```
std::string Token::getLexeme ( ) const
```

Returns the lexeme.

### 3.8.2.2 getSymbol()

```
Symbol Token::getSymbol ( ) const
```

Returns the symbol.

### 3.8.2.3 getVal()

```
int Token::getVal ( ) const
```

Returns the value.

### 3.8.2.4 setLexeme()

```
void Token::setLexeme (
    std::string lexeme )
```

Sets the lexeme.

#### Parameters

<i>lexeme</i>	The lexeme to give the token.
---------------	-------------------------------

### 3.8.2.5 setSymbol()

```
void Token::setSymbol (
    Symbol sym )
```

Sets the symbol.

#### Parameters

<i>sym</i>	The symbol to give the token.
------------	-------------------------------

#### 3.8.2.6 setVal()

```
void Token::setVal (
    int val )
```

Sets the value.

##### Parameters

<i>val</i>	The value to give the token.
------------	------------------------------

#### 3.8.2.7 toString()

```
std::string Token::toString ( )
```

Returns a string representation of the [Token](#).

### 3.8.3 Member Data Documentation

#### 3.8.3.1 lexeme

```
std::string Token::lexeme [private]
```

The tokens lexeme.

#### 3.8.3.2 sname

```
Symbol Token::sname [private]
```

The token's symbol.

#### 3.8.3.3 val

```
int Token::val [private]
```

The numeric value of the token.

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

- [Token.h](#)

## Chapter 4

# File Documentation

### 4.1 Administration.h File Reference

```
#include <iostream>
#include "Token.h"
#include "Scanner.h"
```

#### Classes

- class [Administration](#)

#### Variables

- const int [MAX\\_ERRORS](#) = 10

#### 4.1.1 Variable Documentation

##### 4.1.1.1 MAX\_ERRORS

```
const int MAX_ERRORS = 10
```

### 4.2 Assembler.h File Reference

```
#include <iostream>
#include <string>
```

## Classes

- class [Assembler](#)

## Variables

- const int [MAXLABEL](#) = 500

### 4.2.1 Variable Documentation

#### 4.2.1.1 MAXLABEL

```
const int MAXLABEL = 500
```

## 4.3 BlockTable.h File Reference

```
#include <vector>
#include <map>
#include "TableEntry.h"
#include "Types.h"
```

## Classes

- class [BlockTable](#)

## Macros

- [#define MAXBLOCK 10](#)

### 4.3.1 Macro Definition Documentation

#### 4.3.1.1 MAXBLOCK

```
#define MAXBLOCK 10
```

## 4.4 Grammar.h File Reference

```
#include <Symbol.h>
#include <map>
#include <set>
```

### Enumerations

- enum `NT` {  
`NAME` = 400, `BOOL_SYM`, `NUM_NT`, `CONST_NT`,  
`IDX_SEL`, `VACS`, `FACTOR`, `MULT_OP`,  
`TERM`, `ADD_OP`, `SIMP_EXP`, `REL_OP`,  
`PRIM_EXP`, `PRIM_OP`, `EXP`, `GRCOM`,  
`GRCOM_LIST`, `DO_STMT`, `IF_STMT`, `PROC_STMT`,  
`VACS_LIST`, `ASC_STMT`, `EXP_LIST`, `WRITE_STMT`,  
`READ_STMT`, `EMPTY_STMT`, `STMT`, `STMT_PART`,  
`PROC_DEF`, `VAR_LIST`, `TYPE_SYM`, `CONST_DEF`,  
`DEF`, `VAR_DEF`, `DEF_PART`, `BLOCK`,  
`PROGRAM`, `VPRIME`, `FIELD_LIST`, `PROC_BLOCK`,  
`REC_SEC`, `FORM_PLIST`, `PARAM_DEF`, `ACT_PLIST`,  
`ACT_PARAM`, `SELECT`, `FIELD_SEL`, `CPRIME` }

*Enum to represent all non terminals that are possible in our language.*

### Functions

- bool `in` (std::set< `Symbol` > S, `Symbol` sym)  
*Check if a symbol is in a set.*
- std::set< `Symbol` > `munion` (std::vector< std::set< `Symbol` >> stopSets)  
*Union a vector of stopsets together.*

### Variables

- const std::map< `NT`, std::set< `Symbol` > > `First`  
*Map from non terminals to thier first sets of symbols.*

#### 4.4.1 Enumeration Type Documentation

##### 4.4.1.1 NT

```
enum NT
```

Enum to represent all non terminals that are possible in our language.

## Enumerator

NAME	
BOOL_SYM	
NUM_NT	
CONST_NT	
IDX_SEL	
VACS	
FACTOR	
MULT_OP	
TERM	
ADD_OP	
SIMP_EXP	
REL_OP	
PRIM_EXP	
PRIM_OP	
EXP	
GRCOM	
GRCOM_LIST	
DO_STMT	
IF_STMT	
PROC_STMT	
VACS_LIST	
ASC_STMT	
EXP_LIST	
WRITE_STMT	
READ_STMT	
EMPTY_STMT	
STMT	
STMT_PART	
PROC_DEF	
VAR_LIST	
TYPE_SYM	
CONST_DEF	
DEF	
VAR_DEF	
DEF_PART	
BLOCK	
PROGRAM	
VPRIME	
FIELD_LIST	
PROC_BLOCK	
REC_SEC	
FORM_PLIST	
PARAM_DEF	
ACT_PLIST	
ACT_PARAM	
SELECT	
FIELD_SEL	
CPRIME	



## 4.4.2 Function Documentation

### 4.4.2.1 in()

```
bool in (
    std::set< Symbol > S,
    Symbol sym )
```

Check if a symbol is in a set.

Helper for checking stop set membership.

#### Parameters

<i>S</i>	The symbol set to check.
<i>sym</i>	The symbol to check.

#### Returns

true if sym is in S.

### 4.4.2.2 munion()

```
std::set<Symbol> munion (
    std::vector< std::set< Symbol >> stopSets )
```

Union a vector of stopsets together.

#### Parameters

<i>stopSets</i>	A vector of Symbol sets to union.
-----------------	-----------------------------------

#### Returns

a set of all of the given stopsets.

## 4.4.3 Variable Documentation

### 4.4.3.1 First

```
const std::map<NT, std::set<Symbol> > First
```

Map from non terminals to thier first sets of symbols.

## 4.5 Parser.h File Reference

```
#include <iostream>
#include <set>
#include "Symbol.h"
#include "Token.h"
#include "TableEntry.h"
#include "Administration.h"
#include "BlockTable.h"
```

### Classes

- class [Parser](#)

## 4.6 Scanner.h File Reference

```
#include "SymbolTable.h"
#include "Token.h"
#include <map>
#include <iostream>
```

### Classes

- class [Scanner](#)

## 4.7 Symbol.h File Reference

```
#include <map>
```

### Enumerations

- enum [Symbol](#) {  
    [DOT](#) = 256, [COMMA](#), [SEMI](#), [LHSQR](#),  
    [RHSQR](#), [AMP](#), [BAR](#), [TILD](#),  
    [LESS](#), [EQUAL](#), [GREAT](#), [PLUS](#),  
    [MINUS](#), [TIMES](#), [FSLASH](#), [BSLASH](#),  
    [LHRND](#), [RHRND](#), [INIT](#), [GUARD](#),  
    [ARROW](#), [DOLLAR](#), [INT](#), [BOOL](#),  
    [FALSE](#), [TRUE](#), [BEGIN](#), [END](#),  
    [CONST](#), [ARRAY](#), [PROC](#), [SKIP](#),  
    [READ](#), [WRITE](#), [CALL](#), [IF](#),  
    [FI](#), [DO](#), [OD](#), [ID](#),  
    [KEY](#), [ENDFILE](#), [EMPTY](#), [EPSILON](#),  
    [NEWLINE](#), [NUM](#), [RECORD](#), [FLOAT](#),  
    [VAR](#), [NAME\\_ERR](#), [NUM\\_ERR](#), [CHAR\\_ERR](#) }

*Enum containing all possible Symbols.*

## Variables

- `const std::map< Symbol, std::string > SymbolToString`  
*Map from all symbols to string versions of themselves for printing.*
- `const std::map< std::string, Symbol > SpecialSym`  
*Map for all special lexemes to their symbol.*
- `const std::map< std::string, Symbol > WordSym`  
*Map for all keywords (word symbols) to their symbol.*

### 4.7.1 Enumeration Type Documentation

#### 4.7.1.1 [Symbol](#)

`enum Symbol`

Enum containing all possible Symbols.

##### Enumerator

DOT	
COMMA	
SEMI	
LHSQR	
RHSQR	
AMP	
BAR	
TILD	
LESS	
EQUAL	
GREAT	
PLUS	
MINUS	
TIMES	
FSLASH	
BSLASH	
LHRND	
RHRND	
INIT	
GUARD	
ARROW	
DOLLAR	
INT	
BOOL	
FALSE	
TRUE	
BEGIN	
END	
CONST	
ARRAY	

### Enumerator

PROC	
SKIP	
READ	
WRITE	
CALL	
IF	
FI	
DO	
OD	
ID	
KEY	
ENDFILE	
EMPTY	
EPSILON	
NEWLINE	
NUM	
RECORD	
FLOAT	
VAR	
NAME_ERR	
NUM_ERR	
CHAR_ERR	

## 4.7.2 Variable Documentation

### 4.7.2.1 SpecialSym

```
const std::map<std::string, Symbol> SpecialSym
```

**Initial value:**

```
{
    { ".", Symbol::DOT },
    { ",", Symbol::COMMA },
    { ";", Symbol::SEMI },
    { "[", Symbol::LHSQR },
    { "]", Symbol::RHSQR },
    { "&", Symbol::AMP },
    { "|", Symbol::BAR },
    { "~", Symbol::TILD },
    { "<", Symbol::LESS },
    { "=", Symbol::EQUAL },
    { ">", Symbol::GREAT },
    { "+", Symbol::PLUS },
    { "-", Symbol::MINUS },
    { "*", Symbol::TIMES },
    { "/", Symbol::FSLASH },
    { "\\ ", Symbol::BSLASH },
    { "(", Symbol::LHRND },
    { ")", Symbol::RHRND },
    { ":", Symbol::INIT },
    { "[ ]", Symbol::GUARD },
    { "->", Symbol::ARROW }
}
```

Map for all special lexemes to their symbol.

#### 4.7.2.2 SymbolToString

```
const std::map<Symbol, std::string> SymbolToString
```

Map from all symbols to string versions of themselves for printing.

#### 4.7.2.3 WordSym

```
const std::map<std::string, Symbol> WordSym
```

**Initial value:**

```
{
  {"begin", Symbol::BEGIN},
  {"end", Symbol::END},
  {"const", Symbol::CONST},
  {"array", Symbol::ARRAY},
  {"proc", Symbol::PROC},
  {"skip", Symbol::SKIP},
  {"read", Symbol::READ},
  {"write", Symbol::WRITE},
  {"call", Symbol::CALL},
  {"if", Symbol::IF},
  {"fi", Symbol::FI},
  {"do", Symbol::DO},
  {"od", Symbol::OD},
  {"integer", Symbol::INT},
  {"Boolean", Symbol::BOOL},
  {"true", Symbol::TRUE},
  {"false", Symbol::FALSE},
  {"record", Symbol::RECORD},
  {"var", Symbol::VAR},
  {"float", Symbol::FLOAT}
}
```

Map for all keywords (word symbols) to their symbol.

## 4.8 SymbolTable.h File Reference

```
#include "Token.h"
#include <vector>
#include <string>
```

### Classes

- class [SymbolTable](#)

### Variables

- const int [MOD](#) = 307
- const int [PRIME](#) = 67
- const int [ID\\_MAX\\_CHARS](#) = 10

## 4.8.1 Variable Documentation

### 4.8.1.1 ID\_MAX\_CHARS

```
const int ID_MAX_CHARS = 10
```

### 4.8.1.2 MOD

```
const int MOD = 307
```

### 4.8.1.3 PRIME

```
const int PRIME = 67
```

## 4.9 TableEntry.h File Reference

```
#include <vector>
#include "Types.h"
```

### Classes

- class [TableEntry](#)

## 4.10 Token.h File Reference

```
#include "Symbol.h"
#include <iostream>
#include <string>
```

### Classes

- class [Token](#)

## 4.11 Types.h File Reference

### Enumerations

- enum [Kind](#) {  
    [CONSTANT](#) =500, [VARIABLE](#), [K\\_ARRAY](#), [PROCEDURE](#),  
    [UNDEFINED](#), [K\\_RECORD](#) }  
*Enum containing all the kinds of table entries.*
- enum [Type](#) { [INTEGER](#) =600, [BOOLEAN](#), [UNIVERSAL](#), [T\\_FLOAT](#) }  
*Enum containing all the Types of table entries.*

### Variables

- const std::map< [Kind](#), std::string > [KindToString](#)  
*Mapping the Kinds to strings representing the kinds.*
- const std::map< [Type](#), std::string > [TypeToString](#)  
*Mapping the Type to strings representing the types.*

### 4.11.1 Enumeration Type Documentation

#### 4.11.1.1 Kind

enum [Kind](#)

Enum containing all the kinds of table entries.

##### Enumerator

CONSTANT	
VARIABLE	
K_ARRAY	
PROCEDURE	
UNDEFINED	
K_RECORD	

#### 4.11.1.2 Type

enum [Type](#)

Enum containing all the Types of table entries.

### Enumerator

INTEGER	
BOOLEAN	
UNIVERSAL	
T_FLOAT	

## 4.11.2 Variable Documentation

### 4.11.2.1 KindToString

```
const std::map<Kind, std::string> KindToString
```

#### Initial value:

```
{
  {CONSTANT, "'Constant'"},
  {VARIABLE, "'Variable'"},
  {K_ARRAY, "'Array'"},
  {PROCEDURE, "'Procedure'"},
  {UNDEFINED, "'Undefined'"},
  {K_RECORD, "'Record'"}
}
```

Mapping the Kinds to strings representing the kinds.

### 4.11.2.2 TypeToString

```
const std::map<Type, std::string> TypeToString
```

#### Initial value:

```
{
  {INTEGER, "'Integer'"},
  {BOOLEAN, "'Boolean'"},
  {UNIVERSAL, "'Universal'"},
  {T_FLOAT, "'Float'"}
}
```

Mapping the Type to strings representing the types.



# Index

- ~Assembler
  - Assembler, [10](#)
- ~Scanner
  - Scanner, [37](#)
- actParam
  - Parser, [17](#)
- actParamList
  - Parser, [17](#)
- addOp
  - Parser, [19](#)
- admin
  - Parser, [35](#)
- Administration, [5](#)
  - Administration, [6](#)
  - checkError, [6](#)
  - correctLine, [8](#)
  - currentLine, [7](#)
  - debug, [8](#)
  - debugInfo, [7](#)
  - emit, [7](#)
  - error, [7](#)
  - error\_count, [8](#)
  - errorCount, [8](#)
  - fout, [9](#)
  - getToken, [8](#)
  - lineNum, [9](#)
  - newLine, [8](#)
  - scanner, [9](#)
- Administration.h, [51](#)
  - MAX\_ERRORS, [51](#)
- Assembler, [9](#)
  - ~Assembler, [10](#)
  - Assembler, [10](#)
  - currentAddress, [10](#)
  - firstPass, [10](#)
  - insource, [11](#)
  - labelTable, [11](#)
  - outsource, [11](#)
  - secondPass, [10](#)
- Assembler.h, [51](#)
  - MAXLABEL, [52](#)
- assignStmt
  - Parser, [19](#)
- block
  - Parser, [19](#)
- blockLevel
  - BlockTable, [14](#)
- BlockTable, [11](#)
  - blockLevel, [14](#)
  - BlockTable, [12](#)
  - define, [12](#)
  - find, [13](#)
  - level, [13](#)
  - popBlock, [13](#)
  - pushBlock, [13](#)
  - search, [13](#)
  - table, [14](#)
- BlockTable.h, [52](#)
  - MAXBLOCK, [52](#)
- blocks
  - Parser, [35](#)
- boolSym
  - Parser, [20](#)
- cPrime
  - Parser, [20](#)
- checkError
  - Administration, [6](#)
- constDef
  - Parser, [20](#)
- constant
  - Parser, [20](#)
- correctLine
  - Administration, [8](#)
- currentAddress
  - Assembler, [10](#)
- currentLine
  - Administration, [7](#)
- debug
  - Administration, [8](#)
- debugInfo
  - Administration, [7](#)
- def
  - Parser, [21](#)
- defPart
  - Parser, [21](#)
- define
  - BlockTable, [12](#)
- displace
  - TableEntry, [45](#)
- doStmt
  - Parser, [21](#)
- emit
  - Administration, [7](#)
- emptyStmt
  - Parser, [21](#)

- entries
  - TableEntry, 45
- error
  - Administration, 7
- error\_count
  - Administration, 8
- errorCount
  - Administration, 8
- expr
  - Parser, 22
- exprList
  - Parser, 22
- factor
  - Parser, 22
- fieldList
  - Parser, 23
- fieldSelec
  - Parser, 23
- fin
  - Scanner, 38
- find
  - BlockTable, 13
- findEntry
  - TableEntry, 45
- First
  - Grammar.h, 55
- firstPass
  - Assembler, 10
- formParamList
  - Parser, 23
- fout
  - Administration, 9
- full
  - SymbolTable, 40
- getLexeme
  - Token, 48
- getLoad
  - SymbolTable, 40
- getSymbol
  - Token, 49
- getToken
  - Administration, 8
  - Scanner, 37
  - SymbolTable, 40
- getVal
  - Token, 49
- Grammar.h, 53
  - First, 55
  - in, 55
  - munion, 55
  - NT, 53
- guardedComm
  - Parser, 24
- guardedList
  - Parser, 24
- hash
  - SymbolTable, 41
- ID\_MAX\_CHARS
  - SymbolTable.h, 60
- id
  - TableEntry, 46
- idxSelect
  - Parser, 24
- ifStmt
  - Parser, 25
- in
  - Grammar.h, 55
- insert
  - SymbolTable, 41
- insource
  - Assembler, 11
- isSpecial
  - Scanner, 37
- isWhitespace
  - Scanner, 37
- Kind
  - Types.h, 61
- KindToString
  - Types.h, 62
- label
  - Parser, 35
- labelTable
  - Assembler, 11
- level
  - BlockTable, 13
  - TableEntry, 46
- lexeme
  - Token, 50
- line
  - Scanner, 39
- lineNum
  - Administration, 9
- load
  - SymbolTable, 43
- loadKey
  - SymbolTable, 42
- loadKeywords
  - SymbolTable, 42
- look
  - Parser, 35
- MAX\_ERRORS
  - Administration.h, 51
- MAXBLOCK
  - BlockTable.h, 52
- MAXLABEL
  - Assembler.h, 52
- MOD
  - SymbolTable.h, 60
- match
  - Parser, 25
- multOp

- Parser, [25](#)
- munion
  - Grammar.h, [55](#)
- NewLabel
  - Parser, [25](#)
- newLine
  - Administration, [8](#)
- NT
  - Grammar.h, [53](#)
- outsource
  - Assembler, [11](#)
- PRIME
  - SymbolTable.h, [60](#)
- paramDef
  - Parser, [26](#)
- parse
  - Parser, [26](#)
- Parser, [14](#)
  - actParam, [17](#)
  - actParamList, [17](#)
  - addOp, [19](#)
  - admin, [35](#)
  - assignStmt, [19](#)
  - block, [19](#)
  - blocks, [35](#)
  - boolSym, [20](#)
  - cPrime, [20](#)
  - constDef, [20](#)
  - constant, [20](#)
  - def, [21](#)
  - defPart, [21](#)
  - doStmt, [21](#)
  - emptyStmt, [21](#)
  - expr, [22](#)
  - exprList, [22](#)
  - factor, [22](#)
  - fieldList, [23](#)
  - fieldSelec, [23](#)
  - formParamList, [23](#)
  - guardedComm, [24](#)
  - guardedList, [24](#)
  - idxSelect, [24](#)
  - ifStmt, [25](#)
  - label, [35](#)
  - look, [35](#)
  - match, [25](#)
  - multOp, [25](#)
  - NewLabel, [25](#)
  - paramDef, [26](#)
  - parse, [26](#)
  - Parser, [17](#)
  - primeExpr, [26](#)
  - primeOp, [26](#)
  - procBlock, [28](#)
  - procDef, [28](#)
  - procStmt, [28](#)
- program, [29](#)
- readStmt, [29](#)
- recordSection, [29](#)
- relOp, [29](#)
- selec, [31](#)
- simpleExpr, [31](#)
- stmt, [31](#)
- stmtPart, [31](#)
- syntaxCheck, [32](#)
- syntaxError, [32](#)
- term, [32](#)
- typeSym, [33](#)
- vPrime, [34](#)
- vacsList, [33](#)
- varAccess, [33](#)
- varDef, [33](#)
- varList, [34](#)
- writeStmt, [34](#)
- Parser.h, [56](#)
- popBlock
  - BlockTable, [13](#)
- pos
  - Scanner, [39](#)
- primeExpr
  - Parser, [26](#)
- primeOp
  - Parser, [26](#)
- probe
  - SymbolTable, [42](#)
- procBlock
  - Parser, [28](#)
- procDef
  - Parser, [28](#)
- procStmt
  - Parser, [28](#)
- program
  - Parser, [29](#)
- pushBlock
  - BlockTable, [13](#)
- readStmt
  - Parser, [29](#)
- recognizeName
  - Scanner, [38](#)
- recognizeNumeral
  - Scanner, [38](#)
- recognizeSpecial
  - Scanner, [38](#)
- recordSection
  - Parser, [29](#)
- relOp
  - Parser, [29](#)
- Scanner, [35](#)
  - ~Scanner, [37](#)
  - fin, [38](#)
  - getToken, [37](#)
  - isSpecial, [37](#)
  - isWhitespace, [37](#)

- line, 39
- pos, 39
- recognizeName, 38
- recognizeNumeral, 38
- recognizeSpecial, 38
- Scanner, 36
- symTab, 39
- scanner
  - Administration, 9
- Scanner.h, 56
- search
  - BlockTable, 13
  - SymbolTable, 43
- secondPass
  - Assembler, 10
- selec
  - Parser, 31
- setLexeme
  - Token, 49
- setSymbol
  - Token, 49
- setVal
  - Token, 50
- simpleExpr
  - Parser, 31
- size
  - TableEntry, 46
- sname
  - Token, 50
- SpecialSym
  - Symbol.h, 58
- startLabel
  - TableEntry, 46
- stmt
  - Parser, 31
- stmtPart
  - Parser, 31
- symTab
  - Scanner, 39
- Symbol
  - Symbol.h, 57
- Symbol.h, 56
  - SpecialSym, 58
  - Symbol, 57
  - SymbolToString, 58
  - WordSym, 59
- SymbolTable, 39
  - full, 40
  - getLoad, 40
  - getToken, 40
  - hash, 41
  - insert, 41
  - load, 43
  - loadKey, 42
  - loadKeywords, 42
  - probe, 42
  - search, 43
  - SymbolTable, 40
  - table, 43
  - toString, 43
- SymbolTable.h, 59
  - ID\_MAX\_CHARS, 60
  - MOD, 60
  - PRIME, 60
- SymbolToString
  - Symbol.h, 58
- syntaxCheck
  - Parser, 32
- syntaxError
  - Parser, 32
- table
  - BlockTable, 14
  - SymbolTable, 43
- TableEntry, 44
  - displace, 45
  - entries, 45
  - findEntry, 45
  - id, 46
  - level, 46
  - size, 46
  - startLabel, 46
  - TableEntry, 44
  - tkind, 46
  - ttype, 46
  - val, 47
- TableEntry.h, 60
- term
  - Parser, 32
- tkind
  - TableEntry, 46
- toString
  - SymbolTable, 43
  - Token, 50
- Token, 47
  - getLexeme, 48
  - getSymbol, 49
  - getVal, 49
  - lexeme, 50
  - setLexeme, 49
  - setSymbol, 49
  - setVal, 50
  - sname, 50
  - toString, 50
  - Token, 48
  - val, 50
- Token.h, 60
- ttype
  - TableEntry, 46
- Type
  - Types.h, 61
- typeSym
  - Parser, 33
- TypeToString
  - Types.h, 62
- Types.h, 61
  - Kind, 61

- KindToString, [62](#)
- Type, [61](#)
- TypeToString, [62](#)
- vPrime
  - Parser, [34](#)
- vacList
  - Parser, [33](#)
- val
  - TableEntry, [47](#)
  - Token, [50](#)
- varAccess
  - Parser, [33](#)
- varDef
  - Parser, [33](#)
- varList
  - Parser, [34](#)
- WordSym
  - Symbol.h, [59](#)
- writeStmt
  - Parser, [34](#)