

## CPSC 4660 Compiler

Generated by Doxygen 1.8.5

Wed Mar 11 2020 14:21:54



# 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	debugInfo . . . . .	6
3.1.2.3	error . . . . .	6
3.1.2.4	getToken . . . . .	6
3.1.2.5	newLine . . . . .	6
3.1.3	Member Data Documentation . . . . .	6
3.1.3.1	correctLine . . . . .	6
3.1.3.2	debug . . . . .	6
3.1.3.3	errorCount . . . . .	7
3.1.3.4	fout . . . . .	7
3.1.3.5	lineNum . . . . .	7
3.1.3.6	scanner . . . . .	7
3.2	BlockTable Class Reference . . . . .	7
3.2.1	Constructor & Destructor Documentation . . . . .	8
3.2.1.1	BlockTable . . . . .	8
3.2.2	Member Function Documentation . . . . .	8
3.2.2.1	define . . . . .	8
3.2.2.2	define . . . . .	8
3.2.2.3	find . . . . .	8
3.2.2.4	popBlock . . . . .	8
3.2.2.5	pushBlock . . . . .	8

3.2.2.6	search	8
3.2.3	Member Data Documentation	9
3.2.3.1	blockLevel	9
3.2.3.2	table	9
3.3	Parser Class Reference	9
3.3.1	Constructor & Destructor Documentation	11
3.3.1.1	Parser	11
3.3.2	Member Function Documentation	11
3.3.2.1	actParam	11
3.3.2.2	actParamList	12
3.3.2.3	addOp	12
3.3.2.4	assignStmt	12
3.3.2.5	block	12
3.3.2.6	boolSym	12
3.3.2.7	constant	12
3.3.2.8	constDef	13
3.3.2.9	cPrime	13
3.3.2.10	def	13
3.3.2.11	defPart	13
3.3.2.12	doStmt	13
3.3.2.13	emptyStmt	13
3.3.2.14	expr	13
3.3.2.15	exprList	14
3.3.2.16	factor	14
3.3.2.17	fieldList	14
3.3.2.18	fieldSelec	14
3.3.2.19	formParamList	14
3.3.2.20	guardedComm	14
3.3.2.21	guardedList	15
3.3.2.22	idxSelect	15
3.3.2.23	ifStmt	15
3.3.2.24	match	15
3.3.2.25	multOp	15
3.3.2.26	paramDef	15
3.3.2.27	parse	16
3.3.2.28	primeExpr	16
3.3.2.29	primeOp	16
3.3.2.30	procBlock	16
3.3.2.31	procDef	16
3.3.2.32	procStmt	16

3.3.2.33	program	16
3.3.2.34	readStmt	17
3.3.2.35	recordSection	17
3.3.2.36	relOp	17
3.3.2.37	selec	17
3.3.2.38	simpleExpr	17
3.3.2.39	stmt	17
3.3.2.40	stmtPart	18
3.3.2.41	syntaxCheck	18
3.3.2.42	syntaxError	18
3.3.2.43	term	18
3.3.2.44	typeSym	18
3.3.2.45	vacsList	18
3.3.2.46	varAccess	18
3.3.2.47	varDef	19
3.3.2.48	varList	19
3.3.2.49	vPrime	19
3.3.2.50	writeStmt	19
3.3.3	Member Data Documentation	19
3.3.3.1	admin	19
3.3.3.2	blocks	19
3.3.3.3	look	19
3.4	Scanner Class Reference	20
3.4.1	Constructor & Destructor Documentation	20
3.4.1.1	Scanner	20
3.4.1.2	~Scanner	21
3.4.2	Member Function Documentation	21
3.4.2.1	getToken	21
3.4.2.2	isSpecial	21
3.4.2.3	isWhitespace	21
3.4.2.4	recognizeName	21
3.4.2.5	recognizeNumeral	21
3.4.2.6	recognizeSpecial	22
3.4.3	Member Data Documentation	22
3.4.3.1	fin	22
3.4.3.2	line	22
3.4.3.3	pos	22
3.4.3.4	symTab	22
3.5	SymbolTable Class Reference	22
3.5.1	Constructor & Destructor Documentation	23

3.5.1.1	SymbolTable	23
3.5.2	Member Function Documentation	23
3.5.2.1	full	23
3.5.2.2	getLoad	23
3.5.2.3	getToken	23
3.5.2.4	hash	23
3.5.2.5	insert	24
3.5.2.6	loadKey	24
3.5.2.7	loadKeywords	24
3.5.2.8	probe	24
3.5.2.9	search	24
3.5.2.10	toString	25
3.5.3	Member Data Documentation	25
3.5.3.1	load	25
3.5.3.2	table	25
3.6	TableEntry Class Reference	25
3.6.1	Constructor & Destructor Documentation	26
3.6.1.1	TableEntry	26
3.6.1.2	TableEntry	26
3.6.2	Member Function Documentation	26
3.6.2.1	findEntry	26
3.6.2.2	findEntry	26
3.6.3	Member Data Documentation	26
3.6.3.1	entries	26
3.6.3.2	id	26
3.6.3.3	size	27
3.6.3.4	tkind	27
3.6.3.5	ttype	27
3.6.3.6	val	27
3.7	Token Class Reference	27
3.7.1	Constructor & Destructor Documentation	28
3.7.1.1	Token	28
3.7.1.2	Token	28
3.7.1.3	Token	28
3.7.2	Member Function Documentation	28
3.7.2.1	getLexeme	28
3.7.2.2	getSymbol	28
3.7.2.3	getVal	28
3.7.2.4	setLexeme	28
3.7.2.5	setSymbol	29

3.7.2.6	setVal	29
3.7.2.7	toString	29
3.7.3	Member Data Documentation	29
3.7.3.1	lexeme	29
3.7.3.2	sname	29
3.7.3.3	val	29
<b>4</b>	<b>File Documentation</b>	<b>31</b>
4.1	Administration.h File Reference	31
4.1.1	Variable Documentation	31
4.1.1.1	MAX_ERRORS	31
4.2	BlockTable.h File Reference	31
4.2.1	Macro Definition Documentation	32
4.2.1.1	MAXBLOCK	32
4.3	Grammar.h File Reference	32
4.3.1	Enumeration Type Documentation	32
4.3.1.1	NT	32
4.3.2	Function Documentation	34
4.3.2.1	in	34
4.3.2.2	munion	34
4.3.3	Variable Documentation	34
4.3.3.1	First	34
4.4	Parser.h File Reference	34
4.5	Scanner.h File Reference	34
4.6	Symbol.h File Reference	35
4.6.1	Enumeration Type Documentation	35
4.6.1.1	Symbol	35
4.6.2	Variable Documentation	37
4.6.2.1	SpecialSym	37
4.6.2.2	SymbolToString	37
4.6.2.3	WordSym	37
4.7	SymbolTable.h File Reference	38
4.7.1	Variable Documentation	38
4.7.1.1	ID_MAX_CHARS	38
4.7.1.2	MOD	38
4.7.1.3	PRIME	38
4.8	TableEntry.h File Reference	38
4.9	Token.h File Reference	38
4.10	Types.h File Reference	39
4.10.1	Enumeration Type Documentation	39

---

4.10.1.1	Kind	39
4.10.1.2	Type	39
4.10.2	Variable Documentation	39
4.10.2.1	KindToString	39
4.10.2.2	TypeToString	40
 <b>Index</b>		 <b>41</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="#">BlockTable</a>	7
<a href="#">Parser</a>	9
<a href="#">Scanner</a>	20
<a href="#">SymbolTable</a>	22
<a href="#">TableEntry</a>	25
<a href="#">Token</a>	27



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Administration.h</a>	31
<a href="#">BlockTable.h</a>	31
<a href="#">Grammar.h</a>	32
<a href="#">Parser.h</a>	34
<a href="#">Scanner.h</a>	34
<a href="#">Symbol.h</a>	35
<a href="#">SymbolTable.h</a>	38
<a href="#">TableEntry.h</a>	38
<a href="#">Token.h</a>	38
<a href="#">Types.h</a>	39



## 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.*
- [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.*

#### 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](#)  
*Whether or not to print debugging info.*

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 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 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 void Administration::debugInfo ( std::string *text* )

Print debugging info to the console if in debug mode.

Parameters

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

#### 3.1.2.3 void Administration::error ( std::string *text* )

Display text for an error.

Parameters

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

#### 3.1.2.4 Token Administration::getToken ( )

#### 3.1.2.5 void Administration::newLine ( )

Adds line number and resets correctLine.

### 3.1.3 Member Data Documentation

#### 3.1.3.1 bool Administration::correctLine [private]

True if the line has no errors so far.

#### 3.1.3.2 bool Administration::debug [private]

Wether or not to print debugging info.

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

The total number of errors so far.

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

File to print all tokens to.

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

The current line number.

**3.1.3.6** `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 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)  
*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.*

### 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.*

### 3.2.1 Constructor & Destructor Documentation

#### 3.2.1.1 BlockTable::BlockTable ( )

Default Constructor for a [BlockTable](#).

### 3.2.2 Member Function Documentation

#### 3.2.2.1 bool BlockTable::define ( int *nid*, Kind *nkind*, Type *ntype*, int *nsize*, int *nval* )

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

#### 3.2.2.2 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
--------------	-------------------------------------

#### 3.2.2.3 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

#### 3.2.2.4 void BlockTable::popBlock ( )

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

#### 3.2.2.5 bool BlockTable::pushBlock ( )

Creates and pushes a new blocktable onto the current block.

#### 3.2.2.6 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
---------------	--

### 3.2.3 Member Data Documentation

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

The current blocklevel.

#### 3.2.3.2 `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](#)

## 3.3 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

- 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=std::vector< [TableEntry](#) >())  
*Parses a block from the stream of tokens.*
- void [defPart](#) (std::set< [Symbol](#) > stop)  
*Parses a definition part from the stream of tokens.*
- void [def](#) (std::set< [Symbol](#) > stop)  
*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)  
*Parses variable access.*
- void `varDef` (std::set< `Symbol` > stop)  
*Parses a variable definition from the stream of tokens.*
- void `vPrime` (std::set< `Symbol` > stop, `Type` type)  
*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)  
*Parses a list of guarded commands.*
- void `guardedComm` (std::set< `Symbol` > stop)  
*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.*
- void `primeOp` (std::set< `Symbol` > stop)  
*Parses a primary operator from the stream of tokens.*
- void `relOp` (std::set< `Symbol` > stop)  
*Parses a relational operator from the stream of tokens.*
- void `addOp` (std::set< `Symbol` > stop)  
*Parses a plus or minus operator from the stream of tokens.*
- void `multOp` (std::set< `Symbol` > stop)

- Parses a multiplication or division or modulus operator from the stream of tokens.*
- [Type 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.*
- void [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)
- 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)
- Parses 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

- [Administration & admin](#)  
*The administration object for errors and holding the scanner and symbol table.*
- [Token look](#)  
*The look ahead token.*
- [BlockTable blocks](#)

## 3.3.1 Constructor & Destructor Documentation

### 3.3.1.1 Parser::Parser ( [Administration & admin](#) )

Creates a new [Parser](#) object.

Parameters

<a href="#">admin</a>	An administration object for handling errors and holding our scanner etc. for now.
-----------------------	--

## 3.3.2 Member Function Documentation

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

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

## Parameters

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

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

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

## Parameters

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

### 3.3.2.3 `void 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.
-------------	---

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

Parses an assignment statement.

## Parameters

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

### 3.3.2.5 `void Parser::block ( std::set< Symbol > stop, std::vector< TableEntry > entries = std::vector< TableEntry >() ) [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

### 3.3.2.6 `void 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.
-------------	---

### 3.3.2.7 `Type Parser::constant ( std::set< Symbol > stop ) [private]`

Parses a const non-terminal.

## Parameters

---

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

### 3.3.2.8 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.
-------------	---

### 3.3.2.9 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.
-------------	---

### 3.3.2.10 void Parser::def ( std::set< Symbol > stop ) [private]

Parses a definition from the stream of tokens.

Parameters

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

### 3.3.2.11 void 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.
-------------	---

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

Parses a do statement.

Parameters

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

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

Parses an empty statement.

Parameters

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

### 3.3.2.14 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.
-------------	---

**3.3.2.15** `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.
-------------	---

**3.3.2.16** `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.
-------------	---

**3.3.2.17** `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.

**3.3.2.18** `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.

**3.3.2.19** `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.

**3.3.2.20** `void Parser::guardedComm ( std::set< Symbol > stop )` [private]

Parses a guarded command.

## Parameters

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

**3.3.2.21** void Parser::guardedList ( std::set< Symbol > *stop* ) [private]

Parses a list of guarded commands.

## Parameters

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

**3.3.2.22** 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

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

Parses an if statement.

## Parameters

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

**3.3.2.24** 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.

**3.3.2.25** void 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.
-------------	---

**3.3.2.26** 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.

## 3.3.2.27 void Parser::parse ( )

Parses a PL program.

## 3.3.2.28 Type Parser::primeExpr ( std::set&lt; Symbol &gt; stop ) [private]

Parses a primary expression from the stream of tokens.

## Parameters

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

## 3.3.2.29 void Parser::primeOp ( std::set&lt; Symbol &gt; stop ) [private]

Parses a primary operator from the stream of tokens.

## Parameters

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

## 3.3.2.30 void Parser::procBlock ( std::set&lt; Symbol &gt; stop, int id ) [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.

## 3.3.2.31 void Parser::procDef ( std::set&lt; Symbol &gt; stop ) [private]

Parses a procedure definition from the stream of tokens.

## Parameters

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

## 3.3.2.32 void Parser::procStmt ( std::set&lt; Symbol &gt; stop ) [private]

Parses a procedure call.

## Parameters

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

## 3.3.2.33 void Parser::program ( std::set&lt; Symbol &gt; stop ) [private]

Parses a program from the stream of tokens.



## Parameters

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

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

Parses a read statement.

## Parameters

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

**3.3.2.35** 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.

**3.3.2.36** void 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.
-------------	---

**3.3.2.37** 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.

**3.3.2.38** 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.
-------------	---

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

Parses a statement.

## Parameters

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

**3.3.2.40** 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.
-------------	---

**3.3.2.41** 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.
-------------	---

**3.3.2.42** 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.
-------------	--

**3.3.2.43** 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.
-------------	---

**3.3.2.44** 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.
-------------	---

**3.3.2.45** std::vector<Type> Parser::vacList ( std::set< Symbol > *stop* ) [private]

Parses a variable access list.

Parameters

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

**3.3.2.46** Type Parser::varAccess ( std::set< Symbol > *stop* ) [private]

Parses variable access.

## Parameters

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

**3.3.2.47** void Parser::varDef ( std::set< Symbol > *stop* ) [private]

Parses a variable definition from the stream of tokens.

## Parameters

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

**3.3.2.48** 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.
-------------	---

**3.3.2.49** void Parser::vPrime ( std::set< Symbol > *stop*, Type *type* ) [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

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

Parses a write statement.

## Parameters

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

**3.3.3 Member Data Documentation****3.3.3.1 Administration& Parser::admin** [private]

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

**3.3.3.2 BlockTable Parser::blocks** [private]**3.3.3.3 Token Parser::look** [private]

The look ahead token.

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

- [Parser.h](#)

## 3.4 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.*

### 3.4.1 Constructor & Destructor Documentation

#### 3.4.1.1 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
---------------	------------------

#### 3.4.1.2 Scanner::~Scanner( ) [inline]

Destructor of the scanner.

### 3.4.2 Member Function Documentation

#### 3.4.2.1 Token Scanner::getToken( )

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

#### 3.4.2.2 bool Scanner::isSpecial( char inchar ) [private]

Checks the input 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.

#### 3.4.2.3 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.

#### 3.4.2.4 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.

#### 3.4.2.5 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.

### 3.4.2.6 Token Scanner::recognizeSpecial ( ) [private]

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

Returns

a token for the special symbol scanned.

## 3.4.3 Member Data Documentation

### 3.4.3.1 std::istream& Scanner::fin [private]

The file stream.

### 3.4.3.2 std::string Scanner::line [private]

The current line the scanner is reading.

### 3.4.3.3 std::size\_t Scanner::pos [private]

The position of the char the scanner is reading.

### 3.4.3.4 SymbolTable& Scanner::symTab [private]

The symbol table.

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

- [Scanner.h](#)

## 3.5 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.*

### 3.5.1 Constructor & Destructor Documentation

#### 3.5.1.1 SymbolTable::SymbolTable ( )

### 3.5.2 Member Function Documentation

#### 3.5.2.1 bool SymbolTable::full ( )

Returns true if the table is full.

#### 3.5.2.2 int SymbolTable::getLoad ( )

Returns the number items in the table.

#### 3.5.2.3 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 idx is out of bounds.
---------------------	------------------------------------

#### 3.5.2.4 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.

3.5.2.5 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.
---------------------	------------------------------

3.5.2.6 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.

## 3.5.2.7 void SymbolTable::loadKeywords ( ) [private]

Loads all reserved keywords into the symbol table.

3.5.2.8 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.

3.5.2.9 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.

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

Returns a string representation of the table.

## 3.5.3 Member Data Documentation

3.5.3.1 `int SymbolTable::load` [private]

The number of elements in the hash table.

3.5.3.2 `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](#)

## 3.6 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)  
*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.*

### 3.6.1 Constructor & Destructor Documentation

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

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

#### 3.6.1.2 `TableEntry::TableEntry ( int nid, Kind nkind, Type ntype, int nsize, int nval )` `[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

### 3.6.2 Member Function Documentation

#### 3.6.2.1 `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
--------------	------------------------------------

#### 3.6.2.2 `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
------------	--

### 3.6.3 Member Data Documentation

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

The field/params of a record/procedure respectively.

#### 3.6.3.2 `int TableEntry::id`

The table entry id.

3.6.3.3 `int TableEntry::size`

The size of the required memory for the table entry.

3.6.3.4 `Kind TableEntry::tkind`

The kind of table entry.

3.6.3.5 `Type TableEntry::ttype`

The type of the table entry.

3.6.3.6 `int TableEntry::val`

The value of the table entry.

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

- [TableEntry.h](#)

## 3.7 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.*

## 3.7.1 Constructor & Destructor Documentation

### 3.7.1.1 `Token::Token ( )`

Creates a new default token.

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

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

Creates a new token.

Parameters

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

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

Copy Constructor.

## 3.7.2 Member Function Documentation

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

Returns the lexeme.

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

Returns the symbol.

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

Returns the value.

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

Sets the lexeme.

## Parameters

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

**3.7.2.5 void Token::setSymbol ( Symbol sym )**

Sets the symbol.

## Parameters

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

**3.7.2.6 void Token::setVal ( int val )**

Sets the value.

## Parameters

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

**3.7.2.7 std::string Token::toString ( )**

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

**3.7.3 Member Data Documentation****3.7.3.1 std::string Token::lexeme** `[private]`

The tokens lexeme.

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

The token's symbol.

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

### 4.2 BlockTable.h File Reference

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

#### Classes

- class [BlockTable](#)

#### Macros

- #define [MAXBLOCK](#) 10

### 4.2.1 Macro Definition Documentation

#### 4.2.1.1 #define MAXBLOCK 10

## 4.3 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.*

### 4.3.1 Enumeration Type Documentation

#### 4.3.1.1 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*

### 4.3.2 Function Documentation

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

Check if a symbol is in a set.

Helper for checking stop set membership.

##### Parameters

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

##### Returns

true if sym is in S.

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

Union a vector of stopsets together.

##### Parameters

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

##### Returns

a set of all of the given stopsets.

### 4.3.3 Variable Documentation

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

Map from non terminals to thier first sets of symbols.

## 4.4 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](#)

## 4.5 Scanner.h File Reference

```
#include "SymbolTable.h"
```

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

## Classes

- class [Scanner](#)

## 4.6 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.*

### 4.6.1 Enumeration Type Documentation

#### 4.6.1.1 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**  
**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**

## 4.6.2 Variable Documentation

### 4.6.2.1 `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.

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

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

### 4.6.2.3 `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.

## 4.7 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

#### 4.7.1 Variable Documentation

4.7.1.1 const int [ID\\_MAX\\_CHARS](#) = 10

4.7.1.2 const int [MOD](#) = 307

4.7.1.3 const int [PRIME](#) = 67

## 4.8 TableEntry.h File Reference

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

### Classes

- class [TableEntry](#)

## 4.9 Token.h File Reference

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

### Classes

- class [Token](#)

## 4.10 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.*

### 4.10.1 Enumeration Type Documentation

#### 4.10.1.1 enum Kind

Enum containing all the kinds of table entries.

##### Enumerator

**CONSTANT**  
**VARIABLE**  
**K\_ARRAY**  
**PROCEDURE**  
**UNDEFINED**  
**K\_RECORD**

#### 4.10.1.2 enum Type

Enum containing all the Types of table entries.

##### Enumerator

**INTEGER**  
**BOOLEAN**  
**UNIVERSAL**  
**T\_FLOAT**

### 4.10.2 Variable Documentation

#### 4.10.2.1 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.

#### 4.10.2.2 `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

- ~Scanner
  - Scanner, [21](#)
- ACT\_PARAM
  - Grammar.h, [33](#)
- ACT\_PLIST
  - Grammar.h, [33](#)
- ADD\_OP
  - Grammar.h, [33](#)
- AMP
  - Symbol.h, [36](#)
- ARRAY
  - Symbol.h, [36](#)
- ARROW
  - Symbol.h, [36](#)
- ASC\_STMT
  - Grammar.h, [33](#)
- actParam
  - Parser, [11](#)
- actParamList
  - Parser, [12](#)
- addOp
  - Parser, [12](#)
- admin
  - Parser, [19](#)
- Administration, [5](#)
  - Administration, [6](#)
  - checkError, [6](#)
  - correctLine, [6](#)
  - debug, [6](#)
  - debugInfo, [6](#)
  - error, [6](#)
  - errorCount, [6](#)
  - fout, [7](#)
  - getToken, [6](#)
  - lineNum, [7](#)
  - newLine, [6](#)
  - scanner, [7](#)
- Administration.h, [31](#)
  - MAX\_ERRORS, [31](#)
- assignStmt
  - Parser, [12](#)
- BAR
  - Symbol.h, [36](#)
- BEGIN
  - Symbol.h, [36](#)
- BLOCK
  - Grammar.h, [33](#)
- BOOL
  - Symbol.h, [36](#)
- BOOL\_SYM
  - Grammar.h, [32](#)
- BOOLEAN
  - Types.h, [39](#)
- BSLASH
  - Symbol.h, [36](#)
- block
  - Parser, [12](#)
- blockLevel
  - BlockTable, [9](#)
- BlockTable, [7](#)
  - blockLevel, [9](#)
  - BlockTable, [8](#)
  - BlockTable, [8](#)
  - define, [8](#)
  - find, [8](#)
  - popBlock, [8](#)
  - pushBlock, [8](#)
  - search, [8](#)
  - table, [9](#)
- BlockTable.h, [31](#)
  - MAXBLOCK, [32](#)
- blocks
  - Parser, [19](#)
- boolSym
  - Parser, [12](#)
- CALL
  - Symbol.h, [36](#)
- CHAR\_ERR
  - Symbol.h, [37](#)
- COMMA
  - Symbol.h, [35](#)
- CONST
  - Symbol.h, [36](#)
- CONST\_DEF
  - Grammar.h, [33](#)
- CONST\_NT
  - Grammar.h, [32](#)
- CONSTANT
  - Types.h, [39](#)
- CPRIME
  - Grammar.h, [33](#)
- cPrime
  - Parser, [13](#)
- checkError
  - Administration, [6](#)
- constDef
  - Parser, [13](#)

- constant
  - Parser, [12](#)
- correctLine
  - Administration, [6](#)
- DEF
  - Grammar.h, [33](#)
- DEF\_PART
  - Grammar.h, [33](#)
- DO
  - Symbol.h, [36](#)
- DO\_STMT
  - Grammar.h, [33](#)
- DOLLAR
  - Symbol.h, [36](#)
- DOT
  - Symbol.h, [35](#)
- debug
  - Administration, [6](#)
- debugInfo
  - Administration, [6](#)
- def
  - Parser, [13](#)
- defPart
  - Parser, [13](#)
- define
  - BlockTable, [8](#)
- doStmt
  - Parser, [13](#)
- EMPTY
  - Symbol.h, [36](#)
- EMPTY\_STMT
  - Grammar.h, [33](#)
- END
  - Symbol.h, [36](#)
- ENDFILE
  - Symbol.h, [36](#)
- EPSILON
  - Symbol.h, [36](#)
- EQUAL
  - Symbol.h, [36](#)
- EXP
  - Grammar.h, [33](#)
- EXP\_LIST
  - Grammar.h, [33](#)
- emptyStmt
  - Parser, [13](#)
- entries
  - TableEntry, [26](#)
- error
  - Administration, [6](#)
- errorCount
  - Administration, [6](#)
- expr
  - Parser, [13](#)
- exprList
  - Parser, [14](#)
- FACTOR
  - Grammar.h, [33](#)
- FALSE
  - Symbol.h, [36](#)
- FI
  - Symbol.h, [36](#)
- FIELD\_LIST
  - Grammar.h, [33](#)
- FIELD\_SEL
  - Grammar.h, [33](#)
- FLOAT
  - Symbol.h, [37](#)
- FORM\_PLIST
  - Grammar.h, [33](#)
- FSLASH
  - Symbol.h, [36](#)
- factor
  - Parser, [14](#)
- fieldList
  - Parser, [14](#)
- fieldSelec
  - Parser, [14](#)
- fin
  - Scanner, [22](#)
- find
  - BlockTable, [8](#)
- findEntry
  - TableEntry, [26](#)
- First
  - Grammar.h, [34](#)
- formParamList
  - Parser, [14](#)
- fout
  - Administration, [7](#)
- full
  - SymbolTable, [23](#)
- GRCOM
  - Grammar.h, [33](#)
- GRCOM\_LIST
  - Grammar.h, [33](#)
- GREAT
  - Symbol.h, [36](#)
- GUARD
  - Symbol.h, [36](#)
- getLexeme
  - Token, [28](#)
- getLoad
  - SymbolTable, [23](#)
- getSymbol
  - Token, [28](#)
- getToken
  - Administration, [6](#)
  - Scanner, [21](#)
  - SymbolTable, [23](#)
- getVal
  - Token, [28](#)
- Grammar.h
  - ACT\_PARAM, [33](#)

- ACT\_PLIST, [33](#)
- ADD\_OP, [33](#)
- ASC\_STMT, [33](#)
- BLOCK, [33](#)
- BOOL\_SYM, [32](#)
- CONST\_DEF, [33](#)
- CONST\_NT, [32](#)
- CPRIME, [33](#)
- DEF, [33](#)
- DEF\_PART, [33](#)
- DO\_STMT, [33](#)
- EMPTY\_STMT, [33](#)
- EXP, [33](#)
- EXP\_LIST, [33](#)
- FACTOR, [33](#)
- FIELD\_LIST, [33](#)
- FIELD\_SEL, [33](#)
- FORM\_PLIST, [33](#)
- GRCOM, [33](#)
- GRCOM\_LIST, [33](#)
- IDX\_SEL, [33](#)
- IF\_STMT, [33](#)
- MULT\_OP, [33](#)
- NAME, [32](#)
- NUM\_NT, [32](#)
- PARAM\_DEF, [33](#)
- PRIM\_EXP, [33](#)
- PRIM\_OP, [33](#)
- PROC\_BLOCK, [33](#)
- PROC\_DEF, [33](#)
- PROC\_STMT, [33](#)
- PROGRAM, [33](#)
- READ\_STMT, [33](#)
- REC\_SEC, [33](#)
- REL\_OP, [33](#)
- SELECT, [33](#)
- SIMP\_EXP, [33](#)
- STMT, [33](#)
- STMT\_PART, [33](#)
- TERM, [33](#)
- TYPE\_SYM, [33](#)
- VACS, [33](#)
- VACS\_LIST, [33](#)
- VAR\_DEF, [33](#)
- VAR\_LIST, [33](#)
- VPRIME, [33](#)
- WRITE\_STMT, [33](#)
- Grammar.h, [32](#)
  - First, [34](#)
  - in, [34](#)
  - munion, [34](#)
  - NT, [32](#)
- guardedComm
  - Parser, [14](#)
- guardedList
  - Parser, [15](#)
- hash
  - SymbolTable, [23](#)
- ID
  - Symbol.h, [36](#)
- IDX\_SEL
  - Grammar.h, [33](#)
- IF
  - Symbol.h, [36](#)
- IF\_STMT
  - Grammar.h, [33](#)
- INIT
  - Symbol.h, [36](#)
- INT
  - Symbol.h, [36](#)
- INTEGER
  - Types.h, [39](#)
- ID\_MAX\_CHARS
  - SymbolTable.h, [38](#)
- id
  - TableEntry, [26](#)
- idxSelect
  - Parser, [15](#)
- ifStmt
  - Parser, [15](#)
- in
  - Grammar.h, [34](#)
- insert
  - SymbolTable, [24](#)
- isSpecial
  - Scanner, [21](#)
- isWhitespace
  - Scanner, [21](#)
- K\_ARRAY
  - Types.h, [39](#)
- K\_RECORD
  - Types.h, [39](#)
- KEY
  - Symbol.h, [36](#)
- Kind
  - Types.h, [39](#)
- KindToString
  - Types.h, [39](#)
- LESS
  - Symbol.h, [36](#)
- LHRND
  - Symbol.h, [36](#)
- LHSQR
  - Symbol.h, [36](#)
- lexeme
  - Token, [29](#)
- line
  - Scanner, [22](#)
- lineNum
  - Administration, [7](#)
- load
  - SymbolTable, [25](#)
- loadKey
  - SymbolTable, [24](#)
- loadKeywords

- SymbolTable, [24](#)
- look
  - Parser, [19](#)
- MINUS
  - Symbol.h, [36](#)
- MULT\_OP
  - Grammar.h, [33](#)
- MAX\_ERRORS
  - Administration.h, [31](#)
- MAXBLOCK
  - BlockTable.h, [32](#)
- MOD
  - SymbolTable.h, [38](#)
- match
  - Parser, [15](#)
- multOp
  - Parser, [15](#)
- munion
  - Grammar.h, [34](#)
- NAME
  - Grammar.h, [32](#)
- NAME\_ERR
  - Symbol.h, [37](#)
- NEWLINE
  - Symbol.h, [36](#)
- NUM
  - Symbol.h, [36](#)
- NUM\_ERR
  - Symbol.h, [37](#)
- NUM\_NT
  - Grammar.h, [32](#)
- NT
  - Grammar.h, [32](#)
- newLine
  - Administration, [6](#)
- OD
  - Symbol.h, [36](#)
- PARAM\_DEF
  - Grammar.h, [33](#)
- PLUS
  - Symbol.h, [36](#)
- PRIM\_EXP
  - Grammar.h, [33](#)
- PRIM\_OP
  - Grammar.h, [33](#)
- PROC
  - Symbol.h, [36](#)
- PROC\_BLOCK
  - Grammar.h, [33](#)
- PROC\_DEF
  - Grammar.h, [33](#)
- PROC\_STMT
  - Grammar.h, [33](#)
- PROCEDURE
  - Types.h, [39](#)
- PROGRAM
  - Grammar.h, [33](#)
- PRIME
  - SymbolTable.h, [38](#)
- paramDef
  - Parser, [15](#)
- parse
  - Parser, [16](#)
- Parser, [9](#)
  - actParam, [11](#)
  - actParamList, [12](#)
  - addOp, [12](#)
  - admin, [19](#)
  - assignStmt, [12](#)
  - block, [12](#)
  - blocks, [19](#)
  - boolSym, [12](#)
  - cPrime, [13](#)
  - constDef, [13](#)
  - constant, [12](#)
  - def, [13](#)
  - defPart, [13](#)
  - doStmt, [13](#)
  - emptyStmt, [13](#)
  - expr, [13](#)
  - exprList, [14](#)
  - factor, [14](#)
  - fieldList, [14](#)
  - fieldSelec, [14](#)
  - formParamList, [14](#)
  - guardedComm, [14](#)
  - guardedList, [15](#)
  - idxSelect, [15](#)
  - ifStmt, [15](#)
  - look, [19](#)
  - match, [15](#)
  - multOp, [15](#)
  - paramDef, [15](#)
  - parse, [16](#)
  - Parser, [11](#)
  - primeExpr, [16](#)
  - primeOp, [16](#)
  - procBlock, [16](#)
  - procDef, [16](#)
  - procStmt, [16](#)
  - program, [16](#)
  - readStmt, [17](#)
  - recordSection, [17](#)
  - relOp, [17](#)
  - selec, [17](#)
  - simpleExpr, [17](#)
  - stmt, [17](#)
  - stmtPart, [18](#)
  - syntaxCheck, [18](#)
  - syntaxError, [18](#)
  - term, [18](#)
  - typeSym, [18](#)
  - vPrime, [19](#)

- vacsList, [18](#)
  - varAccess, [18](#)
  - varDef, [19](#)
  - varList, [19](#)
  - writeStmt, [19](#)
- Parser.h, [34](#)
- popBlock
  - BlockTable, [8](#)
- pos
  - Scanner, [22](#)
- primeExpr
  - Parser, [16](#)
- primeOp
  - Parser, [16](#)
- probe
  - SymbolTable, [24](#)
- procBlock
  - Parser, [16](#)
- procDef
  - Parser, [16](#)
- procStmt
  - Parser, [16](#)
- program
  - Parser, [16](#)
- pushBlock
  - BlockTable, [8](#)
- READ
  - Symbol.h, [36](#)
- READ\_STMT
  - Grammar.h, [33](#)
- REC\_SEC
  - Grammar.h, [33](#)
- RECORD
  - Symbol.h, [36](#)
- REL\_OP
  - Grammar.h, [33](#)
- RHRND
  - Symbol.h, [36](#)
- RHSQR
  - Symbol.h, [36](#)
- readStmt
  - Parser, [17](#)
- recognizeName
  - Scanner, [21](#)
- recognizeNumeral
  - Scanner, [21](#)
- recognizeSpecial
  - Scanner, [21](#)
- recordSection
  - Parser, [17](#)
- relOp
  - Parser, [17](#)
- SELECT
  - Grammar.h, [33](#)
- SEMI
  - Symbol.h, [36](#)
- SIMP\_EXP
  - Grammar.h, [33](#)
- SKIP
  - Symbol.h, [36](#)
- STMT
  - Grammar.h, [33](#)
- STMT\_PART
  - Grammar.h, [33](#)
- Scanner, [20](#)
  - ~Scanner, [21](#)
  - fin, [22](#)
  - getToken, [21](#)
  - isSpecial, [21](#)
  - isWhitespace, [21](#)
  - line, [22](#)
  - pos, [22](#)
  - recognizeName, [21](#)
  - recognizeNumeral, [21](#)
  - recognizeSpecial, [21](#)
  - Scanner, [20](#)
  - symTab, [22](#)
- scanner
  - Administration, [7](#)
- Scanner.h, [34](#)
- search
  - BlockTable, [8](#)
  - SymbolTable, [24](#)
- selec
  - Parser, [17](#)
- setLexeme
  - Token, [28](#)
- setSymbol
  - Token, [29](#)
- setVal
  - Token, [29](#)
- simpleExpr
  - Parser, [17](#)
- size
  - TableEntry, [26](#)
- sname
  - Token, [29](#)
- SpecialSym
  - Symbol.h, [37](#)
- stmt
  - Parser, [17](#)
- stmtPart
  - Parser, [18](#)
- symTab
  - Scanner, [22](#)
- Symbol
  - Symbol.h, [35](#)
- Symbol.h
  - AMP, [36](#)
  - ARRAY, [36](#)
  - ARROW, [36](#)
  - BAR, [36](#)
  - BEGIN, [36](#)
  - BOOL, [36](#)
  - BSLASH, [36](#)

- CALL, 36
- CHAR\_ERR, 37
- COMMA, 35
- CONST, 36
- DO, 36
- DOLLAR, 36
- DOT, 35
- EMPTY, 36
- END, 36
- ENDFILE, 36
- EPSILON, 36
- EQUAL, 36
- FALSE, 36
- FI, 36
- FLOAT, 37
- FSLASH, 36
- GREAT, 36
- GUARD, 36
- ID, 36
- IF, 36
- INIT, 36
- INT, 36
- KEY, 36
- LESS, 36
- LHRND, 36
- LHSQR, 36
- MINUS, 36
- NAME\_ERR, 37
- NEWLINE, 36
- NUM, 36
- NUM\_ERR, 37
- OD, 36
- PLUS, 36
- PROC, 36
- READ, 36
- RECORD, 36
- RHRND, 36
- RHSQR, 36
- SEMI, 36
- SKIP, 36
- TILD, 36
- TIMES, 36
- TRUE, 36
- VAR, 37
- WRITE, 36
- Symbol.h, 35
  - SpecialSym, 37
  - Symbol, 35
  - SymbolToString, 37
  - WordSym, 37
- SymbolTable, 22
  - full, 23
  - getLoad, 23
  - getToken, 23
  - hash, 23
  - insert, 24
  - load, 25
  - loadKey, 24
  - loadKeywords, 24
  - probe, 24
  - search, 24
  - SymbolTable, 23
  - SymbolTable, 23
  - table, 25
  - toString, 25
- SymbolTable.h, 38
  - ID\_MAX\_CHARS, 38
  - MOD, 38
  - PRIME, 38
- SymbolToString
  - Symbol.h, 37
- syntaxCheck
  - Parser, 18
- syntaxError
  - Parser, 18
- T\_FLOAT
  - Types.h, 39
- TERM
  - Grammar.h, 33
- TILD
  - Symbol.h, 36
- TIMES
  - Symbol.h, 36
- TRUE
  - Symbol.h, 36
- TYPE\_SYM
  - Grammar.h, 33
- table
  - BlockTable, 9
  - SymbolTable, 25
- TableEntry, 25
  - entries, 26
  - findEntry, 26
  - id, 26
  - size, 26
  - TableEntry, 26
  - TableEntry, 26
  - tkind, 27
  - ttype, 27
  - val, 27
- TableEntry.h, 38
- term
  - Parser, 18
- tkind
  - TableEntry, 27
- toString
  - SymbolTable, 25
  - Token, 29
- Token, 27
  - getLexeme, 28
  - getSymbol, 28
  - getVal, 28
  - lexeme, 29
  - setLexeme, 28
  - setSymbol, 29
  - setVal, 29

- sname, [29](#)
- toString, [29](#)
- Token, [28](#)
- val, [29](#)
- Token.h, [38](#)
- ttype
  - TableEntry, [27](#)
- Type
  - Types.h, [39](#)
- typeSym
  - Parser, [18](#)
- TypeToString
  - Types.h, [40](#)
- Types.h
  - BOOLEAN, [39](#)
  - CONSTANT, [39](#)
  - INTEGER, [39](#)
  - K\_ARRAY, [39](#)
  - K\_RECORD, [39](#)
  - PROCEDURE, [39](#)
  - T\_FLOAT, [39](#)
  - UNDEFINED, [39](#)
  - UNIVERSAL, [39](#)
  - VARIABLE, [39](#)
- Types.h, [39](#)
  - Kind, [39](#)
  - KindToString, [39](#)
  - Type, [39](#)
  - TypeToString, [40](#)
- UNDEFINED
  - Types.h, [39](#)
- UNIVERSAL
  - Types.h, [39](#)
- VACS
  - Grammar.h, [33](#)
- VACS\_LIST
  - Grammar.h, [33](#)
- VAR
  - Symbol.h, [37](#)
- VAR\_DEF
  - Grammar.h, [33](#)
- VAR\_LIST
  - Grammar.h, [33](#)
- VARIABLE
  - Types.h, [39](#)
- VPRIME
  - Grammar.h, [33](#)
- vPrime
  - Parser, [19](#)
- vacList
  - Parser, [18](#)
- val
  - TableEntry, [27](#)
  - Token, [29](#)
- varAccess
  - Parser, [18](#)
- varDef
  - Parser, [19](#)
- varList
  - Parser, [19](#)
- WRITE
  - Symbol.h, [36](#)
- WRITE\_STMT
  - Grammar.h, [33](#)
- WordSym
  - Symbol.h, [37](#)
- writeStmt
  - Parser, [19](#)