

## 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	debugInfo() . . . . .	6
3.1.2.3	error() . . . . .	7
3.1.2.4	getToken() . . . . .	7
3.1.2.5	newLine() . . . . .	7
3.1.3	Member Data Documentation . . . . .	7
3.1.3.1	correctLine . . . . .	7
3.1.3.2	debug . . . . .	7
3.1.3.3	errorCount . . . . .	8
3.1.3.4	fout . . . . .	8
3.1.3.5	lineNum . . . . .	8
3.1.3.6	scanner . . . . .	8
3.2	Parser Class Reference . . . . .	8

3.2.1	Constructor & Destructor Documentation . . . . .	10
3.2.1.1	Parser() . . . . .	10
3.2.2	Member Function Documentation . . . . .	11
3.2.2.1	addOp() . . . . .	11
3.2.2.2	assignStmt() . . . . .	11
3.2.2.3	block() . . . . .	11
3.2.2.4	boolSym() . . . . .	11
3.2.2.5	constant() . . . . .	12
3.2.2.6	constDef() . . . . .	12
3.2.2.7	def() . . . . .	12
3.2.2.8	defPart() . . . . .	12
3.2.2.9	doStmt() . . . . .	12
3.2.2.10	emptyStmt() . . . . .	12
3.2.2.11	expr() . . . . .	13
3.2.2.12	exprList() . . . . .	13
3.2.2.13	factor() . . . . .	13
3.2.2.14	guardedComm() . . . . .	13
3.2.2.15	guardedList() . . . . .	13
3.2.2.16	idxSelect() . . . . .	13
3.2.2.17	ifStmt() . . . . .	14
3.2.2.18	match() . . . . .	14
3.2.2.19	multOp() . . . . .	14
3.2.2.20	parse() . . . . .	14
3.2.2.21	primeExpr() . . . . .	14
3.2.2.22	primeOp() . . . . .	15
3.2.2.23	procDef() . . . . .	15
3.2.2.24	procStmt() . . . . .	15
3.2.2.25	program() . . . . .	15
3.2.2.26	readStmt() . . . . .	15
3.2.2.27	relOp() . . . . .	15

3.2.2.28	<a href="#">simpleExpr()</a>	16
3.2.2.29	<a href="#">stmt()</a>	16
3.2.2.30	<a href="#">stmtPart()</a>	16
3.2.2.31	<a href="#">syntaxCheck()</a>	16
3.2.2.32	<a href="#">syntaxError()</a>	16
3.2.2.33	<a href="#">term()</a>	17
3.2.2.34	<a href="#">typeSym()</a>	17
3.2.2.35	<a href="#">vacList()</a>	17
3.2.2.36	<a href="#">varAccess()</a>	17
3.2.2.37	<a href="#">varDef()</a>	17
3.2.2.38	<a href="#">varList()</a>	18
3.2.2.39	<a href="#">vPrime()</a>	18
3.2.2.40	<a href="#">writeStmt()</a>	18
3.2.3	<a href="#">Member Data Documentation</a>	18
3.2.3.1	<a href="#">admin</a>	18
3.2.3.2	<a href="#">look</a>	18
3.3	<a href="#">Scanner Class Reference</a>	19
3.3.1	<a href="#">Constructor &amp; Destructor Documentation</a>	19
3.3.1.1	<a href="#">Scanner()</a>	19
3.3.1.2	<a href="#">~Scanner()</a>	20
3.3.2	<a href="#">Member Function Documentation</a>	20
3.3.2.1	<a href="#">getToken()</a>	20
3.3.2.2	<a href="#">isSpecial()</a>	20
3.3.2.3	<a href="#">isWhitespace()</a>	20
3.3.2.4	<a href="#">recognizeName()</a>	21
3.3.2.5	<a href="#">recognizeNumeral()</a>	21
3.3.2.6	<a href="#">recognizeSpecial()</a>	21
3.3.3	<a href="#">Member Data Documentation</a>	21
3.3.3.1	<a href="#">fin</a>	22
3.3.3.2	<a href="#">line</a>	22

3.3.3.3	pos	22
3.3.3.4	symmap	22
3.3.3.5	symtable	22
3.4	SymbolTable Class Reference	22
3.4.1	Constructor & Destructor Documentation	23
3.4.1.1	SymbolTable()	23
3.4.2	Member Function Documentation	23
3.4.2.1	full()	24
3.4.2.2	getLoad()	24
3.4.2.3	hash()	24
3.4.2.4	insert()	24
3.4.2.5	loadKey()	25
3.4.2.6	loadKeywords()	25
3.4.2.7	probe()	25
3.4.2.8	search()	26
3.4.2.9	toString()	26
3.4.3	Member Data Documentation	26
3.4.3.1	load	26
3.4.3.2	table	26
3.5	Token Class Reference	27
3.5.1	Constructor & Destructor Documentation	27
3.5.1.1	Token() [1/2]	27
3.5.1.2	Token() [2/2]	27
3.5.2	Member Function Documentation	28
3.5.2.1	getLexeme()	28
3.5.2.2	getSymbol()	28
3.5.2.3	getVal()	28
3.5.2.4	setLexeme()	28
3.5.2.5	setSymbol()	29
3.5.2.6	setVal()	29
3.5.2.7	toString()	29
3.5.3	Member Data Documentation	29
3.5.3.1	lexeme	29
3.5.3.2	sname	30
3.5.3.3	val	30

<b>4 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 Grammar.h File Reference . . . . .	31
4.2.1 Enumeration Type Documentation . . . . .	32
4.2.1.1 NT . . . . .	32
4.2.2 Function Documentation . . . . .	33
4.2.2.1 in() . . . . .	33
4.2.2.2 munion() . . . . .	34
4.2.3 Variable Documentation . . . . .	34
4.2.3.1 First . . . . .	34
4.3 Parser.h File Reference . . . . .	34
4.4 Scanner.h File Reference . . . . .	35
4.5 Symbol.h File Reference . . . . .	35
4.5.1 Enumeration Type Documentation . . . . .	35
4.5.1.1 Symbol . . . . .	35
4.5.2 Variable Documentation . . . . .	37
4.5.2.1 SymbolToString . . . . .	37
4.6 SymbolTable.h File Reference . . . . .	37
4.6.1 Variable Documentation . . . . .	37
4.6.1.1 ID_MAX_CHARS . . . . .	37
4.6.1.2 MOD . . . . .	37
4.6.1.3 PRIME . . . . .	38
4.7 Token.h File Reference . . . . .	38
<b>Index</b>	<b>39</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="#">Parser</a>	8
<a href="#">Scanner</a>	19
<a href="#">SymbolTable</a>	22
<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="#">Grammar.h</a>	31
<a href="#">Parser.h</a>	34
<a href="#">Scanner.h</a>	35
<a href="#">Symbol.h</a>	35
<a href="#">SymbolTable.h</a>	37
<a href="#">Token.h</a>	38



## 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::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 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.
-------------	--------------------

### 3.1.2.3 error()

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

Display text for an error.

#### Parameters

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

### 3.1.2.4 getToken()

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

### 3.1.2.5 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 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)  
*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.*
- void `vacsList` (`std::set< Symbol >` stop)  
*Parses a variable access list.*
- void `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)  
*Parses a variable vs array from the stream of tokens.*
- void `varList` (`std::set< Symbol >` stop)  
*Parses a variable list from the stream of tokens.*
- void `idxSelect` (`std::set< Symbol >` stop)  
*Parses an index selector.*
- void `exprList` (`std::set< Symbol >` stop)

- Parses a expression list from the stream of tokens.*
- void `expr` (std::set< [Symbol](#) > stop)
- Parses a expression from the stream of tokens.*
- void `primeExpr` (std::set< [Symbol](#) > stop)
- Parses a primary expression from the stream of tokens.*
- void `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.*
- void `term` (std::set< [Symbol](#) > stop)
- Parses a term from the stream of tokens.*
- void `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 realtional 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.*
- void `constant` (std::set< [Symbol](#) > stop)
- Parses a const non-terminal.*
- void `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.*

## Private Attributes

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

## 3.2.1 Constructor & Destructor Documentation

### 3.2.1.1 Parser()

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

Creates a new [Parser](#) object.

## Parameters

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

## 3.2.2 Member Function Documentation

### 3.2.2.1 addOp()

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

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

### 3.2.2.2 assignStmt()

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

Parses an assignment statement.

### 3.2.2.3 block()

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

Parses a block from the stream of tokens.

### 3.2.2.4 boolSym()

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

Parses a true or false from the stream of tokens.

#### 3.2.2.5 constant()

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

Parses a const non-terminal.

#### 3.2.2.6 constDef()

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

Parses a constant definitions from the stream of tokens.

#### 3.2.2.7 def()

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

Parses a definition from the stream of tokens.

#### 3.2.2.8 defPart()

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

Parses a definition part from the stream of tokens.

#### 3.2.2.9 doStmt()

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

Parses a do statement.

#### 3.2.2.10 emptyStmt()

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

Parses an empty statement.

#### 3.2.2.11 `expr()`

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

Parses a expression from the stream of tokens.

#### 3.2.2.12 `exprList()`

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

Parses a expression list from the stream of tokens.

#### 3.2.2.13 `factor()`

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

Parses a factor from the stream of tokens.

#### 3.2.2.14 `guardedComm()`

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

Parses a guarded command.

#### 3.2.2.15 `guardedList()`

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

Parses a list of guarded commands.

#### 3.2.2.16 `idxSelect()`

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

Parses an index selector.

ie)  $A[i]$ .

### 3.2.2.17 ifStmt()

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

Parses an if statement.

### 3.2.2.18 match()

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

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

#### Parameters

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

### 3.2.2.19 multOp()

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

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

### 3.2.2.20 parse()

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

Parses a PL program.

### 3.2.2.21 primeExpr()

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

Parses a primary expression from the stream of tokens.

#### 3.2.2.22 primeOp()

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

Parses a primary operator from the stream of tokens.

#### 3.2.2.23 procDef()

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

Parses a procedure definition from the stream of tokens.

#### 3.2.2.24 procStmt()

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

Parses a procedure call.

#### 3.2.2.25 program()

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

Parses a program from the stream of tokens.

#### 3.2.2.26 readStmt()

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

Parses a read statement.

#### 3.2.2.27 relOp()

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

Parses a relational operator from the stream of tokens.

### 3.2.2.28 simpleExpr()

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

Parses a simple expression from the stream of tokens.

### 3.2.2.29 stmt()

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

Parses a statement.

### 3.2.2.30 stmtPart()

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

Parses the statement part of the program.

### 3.2.2.31 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.
-------------	---

### 3.2.2.32 syntaxError()

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

Process a syntax error and perform error recovery.



## Parameters

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

**3.2.2.33 term()**

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

Parses a term from the stream of tokens.

**3.2.2.34 typeSym()**

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

Parses a definition type from the stream of tokens.

**3.2.2.35 vacsList()**

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

Parses a variable access list.

**3.2.2.36 varAccess()**

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

Parses variable access.

**3.2.2.37 varDef()**

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

Parses a variable definition from the stream of tokens.

### 3.2.2.38 varList()

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

Parses a variable list from the stream of tokens.

### 3.2.2.39 vPrime()

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

Parses a variable vs array from the stream of tokens.

### 3.2.2.40 writeStmt()

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

Parses a write statement.

## 3.2.3 Member Data Documentation

### 3.2.3.1 admin

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

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

### 3.2.3.2 look

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

The look ahead token.

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

- [Parser.h](#)

## 3.3 Scanner Class Reference

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

### Public Member Functions

- [Scanner](#) (std::istream &ifs, [SymbolTable](#) &symboltable)  
*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 inputted 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](#) & [symtable](#)  
*The Symbol Table being checked and filled with tokens.*
- std::string [line](#)  
*The current line the scanner is reading.*
- std::size\_t [pos](#)  
*The position of the char the scanner is reading.*
- std::map< std::string, [Symbol](#) > [symmap](#)  
*The map containing the symbols.*

### 3.3.1 Constructor & Destructor Documentation

#### 3.3.1.1 Scanner()

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

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

**Parameters**

<i>ifs</i>	The file stream.
<i>symboltable</i>	The Symbol Table used throughout the scan being updated.

**3.3.1.2 ~Scanner()**

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

Destructor of rthe scanner.

**3.3.2 Member Function Documentation****3.3.2.1 getToken()**

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

Get the next **Token** in the line.

**3.3.2.2 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.

**3.3.2.3 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.

**3.3.2.4 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.

**3.3.2.5 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.

**3.3.2.6 recognizeSpecial()**

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

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

**Returns**

a token for the special symbol scanned.

**3.3.3 Member Data Documentation**

#### 3.3.3.1 `fin`

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

The file stream.

#### 3.3.3.2 `line`

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

The current line the scanner is reading.

#### 3.3.3.3 `pos`

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

The position of the char the scanner is reading.

#### 3.3.3.4 `symmap`

```
std::map<std::string, Symbol> Scanner::symmap [private]
```

The map containing the symbols.

#### 3.3.3.5 `symtable`

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

The Symbol Table being checked and filled with tokens.

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

- [Scanner.h](#)

## 3.4 SymbolTable Class Reference

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

## Public Member Functions

- [SymbolTable](#) ()
- [Token search](#) (const std::string &str)  
*Searches for a lexeme in the symbol table and returns its token.*
- [Token insert](#) (const std::string &str)  
*Inserts a new lexeme into the symbol table.*
- 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.*

### 3.4.1 Constructor & Destructor Documentation

#### 3.4.1.1 SymbolTable()

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

### 3.4.2 Member Function Documentation

### 3.4.2.1 full()

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

Returns true if the table is full.

### 3.4.2.2 getLoad()

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

Returns the number items in the table.

### 3.4.2.3 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.

### 3.4.2.4 insert()

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

Inserts a new lexeme into the symbol table.

Creates a new ID token for the lexeme as once the reserve words are loaded the only thing loaded should be IDs.

#### Parameters

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



**Returns**

an ID token for the string or a FullTableError token if the table is full.

**3.4.2.5 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.

**3.4.2.6 loadKeywords()**

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

Loads all reserved keywords into the symbol table.

**3.4.2.7 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.

#### 3.4.2.8 search()

```
Token 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 token matching the lexeme or the EMPTY token if the table is full.

#### 3.4.2.9 toString()

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

Returns a string representation of the table.

### 3.4.3 Member Data Documentation

#### 3.4.3.1 load

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

The number of elements in the hash table.

#### 3.4.3.2 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](#)

## 3.5 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.*
- [Symbol](#) [getSymbol](#) ()  
*Returns the symbol.*
- std::string [getLexeme](#) ()  
*Returns the lexeme.*
- int [getVal](#) ()  
*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 or it's position in the symbol table.*

### 3.5.1 Constructor & Destructor Documentation

#### 3.5.1.1 [Token\(\)](#) [1/2]

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

Creates a new default token.

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

#### 3.5.1.2 [Token\(\)](#) [2/2]

```
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.

### 3.5.2 Member Function Documentation

#### 3.5.2.1 getLexeme()

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

Returns the lexeme.

#### 3.5.2.2 getSymbol()

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

Returns the symbol.

#### 3.5.2.3 getVal()

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

Returns the value.

#### 3.5.2.4 setLexeme()

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

Sets the lexeme.

## Parameters

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

#### 3.5.2.5 setSymbol()

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

Sets the symbol.

##### Parameters

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

#### 3.5.2.6 setVal()

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

Sets the value.

##### Parameters

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

#### 3.5.2.7 toString()

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

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

### 3.5.3 Member Data Documentation

#### 3.5.3.1 lexeme

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

The tokens lexeme.

### 3.5.3.2 `sname`

`Symbol Token::sname [private]`

The token's symbol.

### 3.5.3.3 `val`

`int Token::val [private]`

The numeric value of the token or it's position in the symbol table.

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 Grammar.h File Reference

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

## Enumerations

- enum `NT` {  
`NAME = 512, 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 }`

*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.*

### 4.2.1 Enumeration Type Documentation

#### 4.2.1.1 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	



## Enumerator

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	

## 4.2.2 Function Documentation

## 4.2.2.1 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.

**4.2.2.2 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.

**4.2.3 Variable Documentation****4.2.3.1 First**

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

Map from non terminals to thier first sets of symbols.

**4.3 Parser.h File Reference**

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

**Classes**

- class [Parser](#)

## 4.4 Scanner.h File Reference

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

### Classes

- class [Scanner](#)

## 4.5 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, NAME\_ERR, NUM\_ERR,  
CHAR\_ERR, FULL\_TAB }

*Enum containing all possible Symbols.*

### Variables

- const std::map< [Symbol](#), std::string > [SymbolToString](#)  
*Map from all symbols to string versions of themselves for printing.*

### 4.5.1 Enumeration Type Documentation

#### 4.5.1.1 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	
PROC	
SKIP	
READ	
WRITE	
CALL	
IF	
FI	
DO	
OD	
ID	
KEY	
ENDFILE	
EMPTY	
EPSILON	
NEWLINE	
NUM	
NAME_ERR	
NUM_ERR	
CHAR_ERR	
FULL_TAB	

## 4.5.2 Variable Documentation

### 4.5.2.1 SymbolToString

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

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

## 4.6 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

## 4.6.1 Variable Documentation

### 4.6.1.1 ID\_MAX\_CHARS

```
const int ID_MAX_CHARS = 10
```

### 4.6.1.2 MOD

```
const int MOD = 307
```

#### 4.6.1.3 PRIME

```
const int PRIME = 67
```

## 4.7 Token.h File Reference

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

### Classes

- class [Token](#)

# Index

- ~Scanner
  - Scanner, [20](#)
- addOp
  - Parser, [11](#)
- admin
  - Parser, [18](#)
- Administration, [5](#)
  - Administration, [6](#)
  - checkError, [6](#)
  - correctLine, [7](#)
  - debug, [7](#)
  - debugInfo, [6](#)
  - error, [7](#)
  - errorCount, [7](#)
  - fout, [8](#)
  - getToken, [7](#)
  - lineNum, [8](#)
  - newLine, [7](#)
  - scanner, [8](#)
- Administration.h, [31](#)
  - MAX\_ERRORS, [31](#)
- assignStmt
  - Parser, [11](#)
- block
  - Parser, [11](#)
- boolSym
  - Parser, [11](#)
- checkError
  - Administration, [6](#)
- constDef
  - Parser, [12](#)
- constant
  - Parser, [11](#)
- correctLine
  - Administration, [7](#)
- debug
  - Administration, [7](#)
- debugInfo
  - Administration, [6](#)
- def
  - Parser, [12](#)
- defPart
  - Parser, [12](#)
- doStmt
  - Parser, [12](#)
- emptyStmt
  - Parser, [12](#)
- error
  - Administration, [7](#)
- errorCount
  - Administration, [7](#)
- expr
  - Parser, [12](#)
- exprList
  - Parser, [13](#)
- factor
  - Parser, [13](#)
- fin
  - Scanner, [21](#)
- First
  - Grammar.h, [34](#)
- fout
  - Administration, [8](#)
- full
  - SymbolTable, [23](#)
- getLexeme
  - Token, [28](#)
- getLoad
  - SymbolTable, [24](#)
- getSymbol
  - Token, [28](#)
- getToken
  - Administration, [7](#)
  - Scanner, [20](#)
- getVal
  - Token, [28](#)
- Grammar.h, [31](#)
  - First, [34](#)
  - in, [33](#)
  - munion, [34](#)
  - NT, [32](#)
- guardedComm
  - Parser, [13](#)
- guardedList
  - Parser, [13](#)
- hash
  - SymbolTable, [24](#)
- ID\_MAX\_CHARS
  - SymbolTable.h, [37](#)
- idxSelect
  - Parser, [13](#)
- ifStmt

- Parser, 13
- in
  - Grammar.h, 33
- insert
  - SymbolTable, 24
- isSpecial
  - Scanner, 20
- isWhitespace
  - Scanner, 20
- lexeme
  - Token, 29
- line
  - Scanner, 22
- lineNum
  - Administration, 8
- load
  - SymbolTable, 26
- loadKey
  - SymbolTable, 25
- loadKeywords
  - SymbolTable, 25
- look
  - Parser, 18
- MAX\_ERRORS
  - Administration.h, 31
- MOD
  - SymbolTable.h, 37
- match
  - Parser, 14
- multOp
  - Parser, 14
- munion
  - Grammar.h, 34
- newLine
  - Administration, 7
- NT
  - Grammar.h, 32
- PRIME
  - SymbolTable.h, 37
- parse
  - Parser, 14
- Parser, 8
  - addOp, 11
  - admin, 18
  - assignStmt, 11
  - block, 11
  - boolSym, 11
  - constDef, 12
  - constant, 11
  - def, 12
  - defPart, 12
  - doStmt, 12
  - emptyStmt, 12
  - expr, 12
  - exprList, 13
  - factor, 13
  - guardedComm, 13
  - guardedList, 13
  - idxSelect, 13
  - ifStmt, 13
  - look, 18
  - match, 14
  - multOp, 14
  - parse, 14
  - Parser, 10
  - primeExpr, 14
  - primeOp, 14
  - procDef, 15
  - procStmt, 15
  - program, 15
  - readStmt, 15
  - relOp, 15
  - simpleExpr, 15
  - stmt, 16
  - stmtPart, 16
  - syntaxCheck, 16
  - syntaxError, 16
  - term, 17
  - typeSym, 17
  - vPrime, 18
  - vacsList, 17
  - varAccess, 17
  - varDef, 17
  - varList, 17
  - writeStmt, 18
- Parser.h, 34
- pos
  - Scanner, 22
- primeExpr
  - Parser, 14
- primeOp
  - Parser, 14
- probe
  - SymbolTable, 25
- procDef
  - Parser, 15
- procStmt
  - Parser, 15
- program
  - Parser, 15
- readStmt
  - Parser, 15
- recognizeName
  - Scanner, 21
- recognizeNumeral
  - Scanner, 21
- recognizeSpecial
  - Scanner, 21
- relOp
  - Parser, 15
- Scanner, 19
  - ~Scanner, 20



- fin, [21](#)
- getToken, [20](#)
- isSpecial, [20](#)
- isWhitespace, [20](#)
- line, [22](#)
- pos, [22](#)
- recognizeName, [21](#)
- recognizeNumeral, [21](#)
- recognizeSpecial, [21](#)
- Scanner, [19](#)
- symmap, [22](#)
- syntable, [22](#)
- scanner
  - Administration, [8](#)
- Scanner.h, [35](#)
- search
  - SymbolTable, [25](#)
- setLexeme
  - Token, [28](#)
- setSymbol
  - Token, [29](#)
- setVal
  - Token, [29](#)
- simpleExpr
  - Parser, [15](#)
- sname
  - Token, [29](#)
- stmt
  - Parser, [16](#)
- stmtPart
  - Parser, [16](#)
- Symbol
  - Symbol.h, [35](#)
- Symbol.h, [35](#)
  - Symbol, [35](#)
  - SymbolToString, [37](#)
- SymbolTable, [22](#)
  - full, [23](#)
  - getLoad, [24](#)
  - hash, [24](#)
  - insert, [24](#)
  - load, [26](#)
  - loadKey, [25](#)
  - loadKeywords, [25](#)
  - probe, [25](#)
  - search, [25](#)
  - SymbolTable, [23](#)
  - table, [26](#)
  - toString, [26](#)
- SymbolTable.h, [37](#)
  - ID\_MAX\_CHARS, [37](#)
  - MOD, [37](#)
  - PRIME, [37](#)
- SymbolToString
  - Symbol.h, [37](#)
- symmap
  - Scanner, [22](#)
- syntable
  - Scanner, [22](#)
- syntaxCheck
  - Parser, [16](#)
- syntaxError
  - Parser, [16](#)
- table
  - SymbolTable, [26](#)
- term
  - Parser, [17](#)
- toString
  - SymbolTable, [26](#)
  - Token, [29](#)
- Token, [27](#)
  - getLexeme, [28](#)
  - getSymbol, [28](#)
  - getVal, [28](#)
  - lexeme, [29](#)
  - setLexeme, [28](#)
  - setSymbol, [29](#)
  - setVal, [29](#)
  - sname, [29](#)
  - toString, [29](#)
  - Token, [27](#)
  - val, [30](#)
- Token.h, [38](#)
- typeSym
  - Parser, [17](#)
- vPrime
  - Parser, [18](#)
- vacList
  - Parser, [17](#)
- val
  - Token, [30](#)
- varAccess
  - Parser, [17](#)
- varDef
  - Parser, [17](#)
- varList
  - Parser, [17](#)
- writeStmt
  - Parser, [18](#)