# CPSC 4660 Compiler

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Class Index

## 1.1 Class List

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

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 Administration Class Reference

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

### Public Member Functions

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

  *Creates a new Administration object.*
- int currentLine ()
- Token getToken ()
- void newLine ()

  *Adds line number and resets correctLine.*
- void debugInfo (std::string text)

  *Print debugging info to the console if in debug mode.*
- void error (std::string text)

  *Display text for an error.*
- void emit (std::string text, int var=-1, int start=-1)

  *Emit assembly code to the output file.*
- int error_count ()

  *Return the number of errors.*

### Private Member Functions

- void checkError (Token ntoken)

  *Checks if current token is an error token.*

## Private Attributes

- std::ostream & fout

  *File to print all tokens to.*
- Scanner & scanner

  *The scanner to use on the input.*
- int lineNum

  *The current line number.*
- bool correctLine

  *True if the line has no errors so far.*
- int errorCount

  *The total number of errors so far.*
- bool debug

  *Wether or not to print debugging info.*

### 3.1.1 Constructor & Destructor Documentation

#### 3.1.1.1 Administration()

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

Creates a new Administration object.

**Parameters**

| | |
|---|---|
| *fout* | The output file stream. |
| *sc* | The scanner beign used by administration. |
| *debug* | 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**

| | |
|---|---|
| *ntoken* | The current token. |

#### 3.1.2.2 currentLine()

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

#### 3.1.2.3 debugInfo()

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

Print debugging info to the console if in debug mode.

**Parameters**

| | |
|---|---|
| *text* | The info to print. |

#### 3.1.2.4 emit()

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

Emit assembly code to the output file.

#### 3.1.2.5 error()

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

Display text for an error.

**Parameters**

| | |
|---|---|
| *text* | The error message. |

#### 3.1.2.6 error_count()

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

Return the number of errors.

#### 3.1.2.7 getToken()

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

#### 3.1.2.8 newLine()

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

Adds line number and resets correctLine.

### 3.1.3 Member Data Documentation

#### 3.1.3.1 correctLine

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

True if the line has no errors so far.

#### 3.1.3.2 debug

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

Wether or not to print debugging info.

#### 3.1.3.3 errorCount

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

The total number of errors so far.

#### 3.1.3.4 fout

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

File to print all tokens to.

#### 3.1.3.5 lineNum

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

The current line number.

#### 3.1.3.6 scanner

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

The scanner to use on the input.

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

- Administration.h

## 3.2 Assembler Class Reference

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

### Public Member Functions

- Assembler (istream &in, ostream &out)
- ∼Assembler ()
- void firstPass ()
- void secondPass ()

## Private Attributes

- int labelTable [MAXLABEL]
- int currentAddress
- istream ∗ insource
- ostream ∗ outsource

### 3.2.1 Constructor & Destructor Documentation

#### 3.2.1.1 Assembler()

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

#### 3.2.1.2 ∼Assembler()

```
Assembler::∼Assembler ( )
```

### 3.2.2 Member Function Documentation

#### 3.2.2.1 firstPass()

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

#### 3.2.2.2 secondPass()

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

### 3.2.3 Member Data Documentation

#### 3.2.3.1 currentAddress

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

#### 3.2.3.2 insource

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

#### 3.2.3.3 labelTable

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

#### 3.2.3.4 outsource

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

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

- Assembler.h

## 3.3 BlockTable Class Reference

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

### Public Member Functions

- BlockTable ()

  *Default Constructor for a BlockTable.*
- bool search (int lookId)

  *Searches the current level of the blocktable for a table entry.*
- bool define (int nid, Kind nkind, Type ntype, int nsize, int nval, int displace)

  *Creates a new table entry and puts it into the current block if it doesnt already exist.*
- bool define (TableEntry &entry)

  *Overloaded define function that takes in a table entry to define.*
- TableEntry find (int lookId, bool &error)

  *Searches the entire blocktable for the table entry.*
- bool pushBlock ()

  *Creates and pushes a new blocktable onto the currect block.*
- void popBlock ()

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

  *The current block level.*

## Private Attributes

- std::vector< std::map< int, TableEntry > > table

  *Vector of maps storing the table entries for a block (the block table)*
- int blockLevel

  *The current blocklevel.*

## 3.3.1 Constructor & Destructor Documentation

### 3.3.1.1 BlockTable()

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

Default Constructor for a BlockTable.

## 3.3.2 Member Function Documentation

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

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

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

**Parameters**

| | |
|---|---|
| *nid* | The id of the table entry |
| *nkind* | The kind of the table entry |
| *ntype* | The type of the table entry |
| *nsize* | The memory size required by the table entry |
| *nval* | The value of the table entry |

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

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

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

**Parameters**

| | |
|---|---|
| *entry* | The table entry that will be define |

#### 3.3.2.3 find()

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

Searches the entire blocktable for the table entry.

**Parameters**

| | |
|---|---|
| *look←<br>Id* | The id of the table entry being searched for |
| *error* | The error check for when the table entry does not exist |

#### 3.3.2.4 level()

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

The current block level.

#### 3.3.2.5 popBlock()

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

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

#### 3.3.2.6 pushBlock()

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

Creates and pushes a new blocktable onto the currect block.

#### 3.3.2.7 search()

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

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

**Parameters**

|  |  |
|---|---|
| *lookID* | The id of the table entry being searched for |

### 3.3.3 Member Data Documentation

#### 3.3.3.1 blockLevel

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

The current blocklevel.

#### 3.3.3.2 table

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

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

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

- BlockTable.h

## 3.4 Parser Class Reference

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

### Public Member Functions

- Parser (Administration &admin)

  *Creates a new Parser object.*
- void parse ()

  *Parses a PL program.*

## Private Member Functions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  *Parses an index selector.*

- std::vector< Type > exprList (std::set< Symbol > stop)

  *Parses a expression list from the stream of tokens.*
- Type expr (std::set< Symbol > stop)

  *Parses a expression from the stream of tokens.*
- Type primeExpr (std::set< Symbol > stop)

  *Parses a primary expression from the stream of tokens.*
- Type simpleExpr (std::set< Symbol > stop)

  *Parses a simple expression from the stream of tokens.*
- void guardedList (std::set< Symbol > stop, int &start, int next)

  *Parses a list of guarded commands.*
- void guardedComm (std::set< Symbol > stop, int &start, int next)

  *Parses a guarded command.*
- Type term (std::set< Symbol > stop)

  *Parses a term from the stream of tokens.*
- Type factor (std::set< Symbol > stop)

  *Parses a factor from the stream of tokens.*
- std::string primeOp (std::set< Symbol > stop)

  *Parses a primary operator from the stream of tokens.*
- std::string relOp (std::set< Symbol > stop)

  *Parses a realtional operator from the stream of tokens.*
- std::string addOp (std::set< Symbol > stop)

  *Parses a plus or minus operator from the stream of tokens.*
- std::string multOp (std::set< Symbol > stop)

  *Parses a multiplication or division or modulus operator from the stream of tokens.*
- std::pair< Type, int > constant (std::set< Symbol > stop)

  *Parses a const non-terminal.*
- Type cPrime (std::set< Symbol > stop)

  *Parses a const num non-terminal.*
- Type typeSym (std::set< Symbol > stop)

  *Parses a definition type from the stream of tokens.*
- int boolSym (std::set< Symbol > stop)

  *Parses a true or false from the stream of tokens.*
- void fieldList (std::set< Symbol > stop, std::vector< TableEntry > &fields)

  *Parses the a list of all the fields and their corresponding types declared.*
- void recordSection (std::set< Symbol > stop, std::vector< TableEntry > &fields)

  *Parses a list of idetifiers of the same type declared in a record.*
- void procBlock (std::set< Symbol > stop, int id, int start, int var, int proc)

  *Parses the block for a procedure declaration.*
- void formParamList (std::set< Symbol > stop, std::vector< TableEntry > &params)

  *Parses the parameter list when a procdure is being declared.*
- void paramDef (std::set< Symbol > stop, std::vector< TableEntry > &params)

  *Parses a list of idetifiers being passed into the procedure, can be tagged with "var" meaning it is pass by reference, pass by value otherwise.*
- std::vector< Type > actParamList (std::set< Symbol > stop)

  *Parases the list of parameters when a procedure is being called.*
- Type actParam (std::set< Symbol > stop)

  *Parses the individual paramters inside the paramater list when a procedure is called.*
- Type selec (std::set< Symbol > stop, TableEntry entry)

  *Parses whether the varaible being accessed is in a record or expression.*
- Type fieldSelec (std::set< Symbol > stop, TableEntry entry)

  *Parses field/variable being selected from a record.*

## Private Attributes

- int label
- Administration & admin

  *The administration object for errors and holding the scanner and symbol table.*
- Token look

  *The look ahead token.*
- BlockTable blocks

## 3.4.1 Constructor & Destructor Documentation

### 3.4.1.1 Parser()

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

Creates a new Parser object.

**Parameters**

| | |
|---|---|
| *admin* | An administration object for handling errors and holding our scanner etc. for now. |

## 3.4.2 Member Function Documentation

### 3.4.2.1 actParam()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

### 3.4.2.2 actParamList()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.3 addOp()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.4 assignStmt()

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

Parses an assignment statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.5 block()

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

Parses a block from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *entries* | The entries being added to the block |

#### 3.4.2.6 boolSym()

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

Parses a true or false from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.7 constant()

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

Parses a const non-terminal.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.8 constDef()

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

Parses a constant definitions from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.9 cPrime()

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

Parses a const num non-terminal.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.10 def()

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

Parses a definition from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.11 defPart()

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

Parses a definition part from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.12 doStmt()

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

Parses a do statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.13 emptyStmt()

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

Parses an empty statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.14 expr()

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

Parses a expression from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.15 exprList()

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

Parses a expression list from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.16 factor()

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

Parses a factor from the stream of tokens.

#### Parameters

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

### 3.4.2.17 fieldList()

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

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

in a record.

#### Parameters

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *fields* | The field of the record being declared. |

### 3.4.2.18 fieldSelec()

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

Parses field/variable being selected from a record.

#### Parameters

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *entry* | The table entry of the record being accessed. |

### 3.4.2.19 formParamList()

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

Parses the parameter list when a procdure is being declared.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *params* | The parameters of the procedure being defined. |

#### 3.4.2.20 guardedComm()

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

Parses a guarded command.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.21 guardedList()

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

Parses a list of guarded commands.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.22 idxSelect()

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

Parses an index selector.

ie) A[i].

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *entry* | The Table entry being created |

#### 3.4.2.23 ifStmt()

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

Parses an if statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.24 match()

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

Match a Token and move to the next one.

**Parameters**

| | |
|---|---|
| *symbol* | The symbol being matched |
| *stop* | The stopsets used to recover from the error. |

#### 3.4.2.25 multOp()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.26 NewLabel()

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

#### 3.4.2.27 paramDef()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *params* | The parameters of the procedure being defined. |

#### 3.4.2.28 parse()

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

Parses a PL program.

#### 3.4.2.29 primeExpr()

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

Parses a primary expression from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.30 primeOp()

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

Parses a primary operator from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.31 procBlock()

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

Parses the block for a procedure declaration.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *id* | The id of the procedure. |

#### 3.4.2.32 procDef()

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

Parses a procedure definition from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.33 procStmt()

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

Parses a procedure call.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.34 program()

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

Parses a program from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.35 readStmt()

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

Parses a read statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.36 recordSection()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *fields* | The field of the record being declared. |

#### 3.4.2.37 relOp()

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

Parses a realtional operator from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.38 selec()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *entry* | The table entry of the record being accesssed. |

#### 3.4.2.39 simpleExpr()

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

Parses a simple expression from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.40 stmt()

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

Parses a statement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.41 stmtPart()

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

Parses the statement part of the program.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.42 syntaxCheck()

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

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

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.43 syntaxError()

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

Process a syntax error and perform error recovery.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from the error. |

#### 3.4.2.44 term()

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

Parses a term from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.45 typeSym()

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

Parses a definition type from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.46 vacsList()

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

Parses a variable access list.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.47 varAccess()

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

Parses variable access.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.48 varDef()

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

Parses a varaible definition from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.49 varList()

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

Parses a varaible list from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

#### 3.4.2.50 vPrime()

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

Parses a varaible vs array from the stream of tokens.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |
| *type* | The type of the table entry that is being created |

#### 3.4.2.51 writeStmt()

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

Parses a write stamtement.

**Parameters**

| | |
|---|---|
| *stop* | The stopsets used to recover from an error. |

### 3.4.3 Member Data Documentation

#### 3.4.3.1 admin

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

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

#### 3.4.3.2 blocks

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

#### 3.4.3.3 label

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

#### 3.4.3.4 look

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

The look ahead token.

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

- Parser.h

## 3.5 Scanner Class Reference

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

### Public Member Functions

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

  *Constructor for the scanner, initializes the private varaibles to appropriate values.*
- ∼Scanner ()

  *Destructor of rthe 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 inputed 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 postion of the char the scanner is reading.*

## 3.5.1 Constructor & Destructor Documentation

### 3.5.1.1 Scanner()

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

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

**Parameters**

| | |
|---|---|
| *ifs* | The file stream. |
| *symTab* | The symbol table |

#### 3.5.1.2 ∼Scanner()

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

Destructor of rthe scanner.

### 3.5.2 Member Function Documentation

#### 3.5.2.1 getToken()

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

Get the next Token in the line.

#### 3.5.2.2 isSpecial()

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

Checks the inputed char against all possible symbols.

**Parameters**

| | |
|---|---|
| *inchar* | The current char being read in |

**Returns**

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

#### 3.5.2.3 isWhitespace()

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

Checks the input symbol against Whitespace whether tab or space.

**Parameters**

| | |
|---|---|
| *inchar* | The current char being read in |

**Returns**

true if the char is whitespace, false otherwise.

#### 3.5.2.4 recognizeName()

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

Read and generate tokens for keywords and ID's, also checks for invalid characters and returns a CHAR_ERR token and checks the symbol table is filled then return a FULL_TAB error token.

**Returns**

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

#### 3.5.2.5 recognizeNumeral()

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

Read and generate a token for any number/digit.

**Returns**

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

#### 3.5.2.6 recognizeSpecial()

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

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

**Returns**

a token for the special symbol scanned.

### 3.5.3 Member Data Documentation

#### 3.5.3.1 fin

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

The file stream.

#### 3.5.3.2 line

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

The current line the scanner is reading.

#### 3.5.3.3 pos

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

The postion of the char the scanner is reading.

#### 3.5.3.4 symTab

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

The symbol table.

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

- Scanner.h

## 3.6 SymbolTable Class Reference

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

### Public Member Functions

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

  *Searches for a lexeme in the symbol table and returns its token.*
- int insert (const std::string &str)

  *Inserts a new lexeme into the symbol table if it is not already there.*
- Token & getToken (int idx, bool &found)

  *Get a reference to the token in the symbol table by its index.*
- int hash (const std::string &str)

  *Computes a rolling hash for a given string using the MOD constant.*
- bool full ()

  *Returns true if the table is full.*
- int getLoad ()

  *Returns the number items in the table.*
- std::string toString ()

  *Returns a string representation of the table.*

## Private Member Functions

- std::pair< int, Token & > probe (int idx, std::string lexeme)

  *Given a position linear probe until the token with the given lexeme is found or an empty token is found.*
- void loadKey (Symbol sym, const std::string &lexeme)

  *Load a token for a reserved keyword into the table.*
- void loadKeywords ()

  *Loads all reserved keywords into the symbol table.*

## Private Attributes

- std::vector< Token > table

  *Backing array for the hash table.*
- int load

  *The number of elements in the hash table.*

## 3.6.1 Constructor & Destructor Documentation

#### 3.6.1.1 SymbolTable()

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

## 3.6.2 Member Function Documentation

#### 3.6.2.1 full()

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

Returns true if the table is full.

#### 3.6.2.2 getLoad()

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

Returns the number items in the table.

#### 3.6.2.3 getToken()

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

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

**Parameters**

| | |
|---|---|
| *idx* | The index of the token. |
| *found* | |

**Returns**

a reference to the token or a dummy empty token.

**Exceptions**

| | |
|---|---|
| *out_of_range* | error if the idx is out of bounds. |

#### 3.6.2.4 hash()

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

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

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

**Parameters**

| | |
|---|---|
| *str* | The string to hash. |

**Returns**

the integer hash value of the string.

#### 3.6.2.5 insert()

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

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

**Parameters**

| | |
|---|---|
| *str* | Insert a string into the hash table. |

**Returns**

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

**Exceptions**

| | |
|---|---|
| *length_error* | if the symbol table is full. |

#### 3.6.2.6 loadKey()

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

Load a token for a reserved keyword into the table.

**Parameters**

| | |
|---|---|
| *lexeme* | The tokens's lexeme. |
| *sym* | The token's symbol. |

#### 3.6.2.7 loadKeywords()

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

Loads all reserved keywords into the symbol table.

#### 3.6.2.8 probe()

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

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

**Parameters**

| | |
|---|---|
| *idx* | The initial position to start probing. Generally the lexemes hash value. |
| *lexeme* | The lexeme to probe for. |

**Returns**

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

#### 3.6.2.9 search()

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

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

**Parameters**

| | |
|---|---|
| *str* | The lexeme to search for. |

**Returns**

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

#### 3.6.2.10 toString()

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

Returns a string representation of the table.

### 3.6.3 Member Data Documentation

#### 3.6.3.1 load

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

The number of elements in the hash table.

#### 3.6.3.2 table

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

Backing array for the hash table.

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

- SymbolTable.h

## 3.7 TableEntry Class Reference

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

### Public Member Functions

- TableEntry ()

  *Default Constructor that creates a empty table entry set to default values.*
- TableEntry (int nid, Kind nkind, Type ntype, int nsize, int nval, int disp)

  *Overloaded constructor that creates the table entry with the input values.*
- int findEntry (TableEntry &entry)

  *Check if the table entry input is a param or field of a procedure or record.*
- int findEntry (int id)

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

### Public Attributes

- int id

  *The table entry id.*
- Kind tkind

  *The kind of table entry.*
- Type ttype

  *The type of the table entry.*
- int size

  *The size of the required memory for the table entry.*
- int val

  *The value of the table entry.*
- std::vector< TableEntry > entries

  *The field/params of a record/procedure respectively.*
- int level
- int displace
- int startLabel

### 3.7.1 Constructor & Destructor Documentation

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

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

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

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

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

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

**Parameters**

| | |
|---|---|
| *nid* | The id of the table entry |
| *nkind* | The Kind of the table entry |
| *ntype* | The Type of the table entry |
| *nsize* | The memory size required by the table entry |
| *nval* | The value of the table entry |

## 3.7.2 Member Function Documentation

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

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

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

**Parameters**

| | |
|---|---|
| *entry* | The table entry being searched for |

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

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

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

**Parameters**

| | |
|---|---|
| *The* | id of the table entry being searched for |

## 3.7.3 Member Data Documentation

### 3.7.3.1 displace

```
int TableEntry::displace
```

#### 3.7.3.2 entries

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

The field/params of a record/procedure respectively.

#### 3.7.3.3 id

```
int TableEntry::id
```

The table entry id.

#### 3.7.3.4 level

```
int TableEntry::level
```

#### 3.7.3.5 size

```
int TableEntry::size
```

The size of the required memory for the table entry.

#### 3.7.3.6 startLabel

```
int TableEntry::startLabel
```

#### 3.7.3.7 tkind

```
Kind TableEntry::tkind
```

The kind of table entry.

#### 3.7.3.8 ttype

```
Type TableEntry::ttype
```

The type of the table entry.

#### 3.7.3.9 val

```
int TableEntry::val
```

The value of the table entry.

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

- TableEntry.h

## 3.8 Token Class Reference

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

### Public Member Functions

- Token ()

  *Creates a new default token.*
- Token (Symbol sym, std::string lexeme="", int val=-1)

  *Creates a new token.*
- Token (const Token &tok)

  *Copy Constructor.*
- Symbol getSymbol () const

  *Returns the symbol.*
- std::string getLexeme () const

  *Returns the lexeme.*
- int getVal () const

  *Returns the value.*
- void setSymbol (Symbol sym)

  *Sets the symbol.*
- void setLexeme (std::string lexeme)

  *Sets the lexeme.*
- void setVal (int val)

  *Sets the value.*
- std::string toString ()

  *Returns a string representation of the Token.*

## Private Attributes

- Symbol sname

  *The token's symbol.*
- std::string lexeme

  *The tokens lexeme.*
- int val

  *The numeric value of the token.*

## 3.8.1 Constructor & Destructor Documentation

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

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

Creates a new default token.

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

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

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

Creates a new token.

**Parameters**

| | |
|---|---|
| *sym* | The symbol for the token. |
| *lexeme* | The lexeme for the token. Default "". |
| *val* | The numerical value to give to the token. Default -1. |

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

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

Copy Constructor.

## 3.8.2 Member Function Documentation

#### 3.8.2.1 getLexeme()

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

Returns the lexeme.

#### 3.8.2.2 getSymbol()

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

Returns the symbol.

#### 3.8.2.3 getVal()

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

Returns the value.

#### 3.8.2.4 setLexeme()

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

Sets the lexeme.

**Parameters**

| | |
|---|---|
| *lexeme* | The lexeme to give the token. |

#### 3.8.2.5 setSymbol()

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

Sets the symbol.

**Parameters**

| | |
|---|---|
| *sym* | The symbol to give the token. |

#### 3.8.2.6 setVal()

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

Sets the value.

**Parameters**

| | |
|---|---|
| *val* | The value to give the token. |

#### 3.8.2.7 toString()

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

Returns a string representation of the Token.

### 3.8.3 Member Data Documentation

#### 3.8.3.1 lexeme

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

The tokens lexeme.

#### 3.8.3.2 sname

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

The token's symbol.

#### 3.8.3.3 val

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

The numeric value of the token.

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

- Token.h

# Chapter 4

# File Documentation

## 4.1 Administration.h File Reference

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

#### Classes

- class Administration

#### Variables

- const int MAX_ERRORS = 10

### 4.1.1 Variable Documentation

#### 4.1.1.1 MAX_ERRORS

```
const int MAX_ERRORS = 10
```

## 4.2 Assembler.h File Reference

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

## Classes

- class Assembler

## Variables

- const int MAXLABEL = 500

### 4.2.1 Variable Documentation

#### 4.2.1.1 MAXLABEL

```
const int MAXLABEL = 500
```

## 4.3 BlockTable.h File Reference

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

## Classes

- class BlockTable

## Macros

- #define MAXBLOCK 10

### 4.3.1 Macro Definition Documentation

#### 4.3.1.1 MAXBLOCK

```
#define MAXBLOCK 10
```

## 4.4 Grammar.h File Reference

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

### Enumerations

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

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

### Functions

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

  *Check if a symbol is in a set.*
- std::set< Symbol > munion (std::vector< std::set< Symbol >> stopSets)

  *Union a vector of stopsets together.*

### Variables

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

  *Map from non terminals to thier first sets of symbols.*

### 4.4.1 Enumeration Type Documentation

#### 4.4.1.1 NT

```
enum NT
```

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

**Enumerator**

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

### 4.4.2 Function Documentation

#### 4.4.2.1 in()

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

Check if a symbol is in a set.

Helper for checking stop set membership.

**Parameters**

| | |
|---|---|
| *S* | The symbol set to check. |
| *sym* | The symbol to check. |

**Returns**

true if sym is in S.

#### 4.4.2.2 munion()

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

Union a vector of stopsets together.

**Parameters**

| | |
|---|---|
| *stopSets* | A vector of Symbol sets to union. |

**Returns**

a set of all of the given stopsets.

### 4.4.3 Variable Documentation

#### 4.4.3.1 First

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

Map from non terminals to thier first sets of symbols.

## 4.5 Parser.h File Reference

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

### Classes

- class Parser

## 4.6 Scanner.h File Reference

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

### Classes

- class Scanner

## 4.7 Symbol.h File Reference

```
#include <map>
```

### Enumerations

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

  *Enum containing all possible Symbols.*

### Variables

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

  *Map from all symbols to string versions of themselves for printing.*
- const std::map< std::string, Symbol > SpecialSym

  *Map for all special lexemes to their symbol.*
- const std::map< std::string, Symbol > WordSym

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

## 4.7.1 Enumeration Type Documentation

### 4.7.1.1 Symbol

```
enum Symbol
```

Enum containing all possible Symbols.

**Enumerator**

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

**Enumerator**

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

### 4.7.2 Variable Documentation

#### 4.7.2.1 SpecialSym

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

**Initial value:**

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

Map for all special lexemes to their symbol.

#### 4.7.2.2 SymbolToString

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

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

#### 4.7.2.3 WordSym

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

**Initial value:**

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

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

## 4.8 SymbolTable.h File Reference

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

### Classes

- class SymbolTable

### Variables

- const int MOD = 307
- const int PRIME = 67
- const int ID_MAX_CHARS = 10

### 4.8.1 Variable Documentation

#### 4.8.1.1 ID_MAX_CHARS

```
const int ID_MAX_CHARS = 10
```

#### 4.8.1.2 MOD

```
const int MOD = 307
```

#### 4.8.1.3 PRIME

```
const int PRIME = 67
```

## 4.9 TableEntry.h File Reference

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

### Classes

- class TableEntry

## 4.10 Token.h File Reference

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

### Classes

- class Token

## 4.11 Types.h File Reference

### Enumerations

- enum Kind {
  CONSTANT =500, VARIABLE, K_ARRAY, PROCEDURE,
  UNDEFINED, K_RECORD }

  *Enum containing all the kinds of table entries.*
- enum Type { INTEGER =600, BOOLEAN, UNIVERSAL, T_FLOAT }

  *Enum containing all the Types of table entries.*

### Variables

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

  *Mapping the Kinds to strings representing the kinds.*
- const std::map< Type, std::string > TypeToString

  *Mapping the Type to strings representing the types.*

### 4.11.1 Enumeration Type Documentation

#### 4.11.1.1 Kind

```
enum Kind
```

Enum containing all the kinds of table entries.

**Enumerator**

| | |
|---|---|
| CONSTANT | |
| VARIABLE | |
| K_ARRAY | |
| PROCEDURE | |
| UNDEFINED | |
| K_RECORD | |

#### 4.11.1.2 Type

```
enum Type
```

Enum containing all the Types of table entries.

**Enumerator**

| | |
|---|---|
| INTEGER | |
| BOOLEAN | |
| UNIVERSAL | |
| T_FLOAT | |

### 4.11.2 Variable Documentation

#### 4.11.2.1 KindToString

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

**Initial value:**

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

Mapping the Kinds to strings representing the kinds.

#### 4.11.2.2 TypeToString

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

**Initial value:**

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

Mapping the Type to strings representing the types.

# Index