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.
- 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

  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 debugInfo()

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

Print debugging info to the console if in debug mode.

Parameters

text	The info to print.

3.1.2.3 error()

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

Display text for an error.

Parameters

text	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 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.
- 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 varaible definition from the stream of tokens.
- void vPrime (std::set< Symbol > stop)

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

  Parses a varaible 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

admin	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

stop	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 realtional 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

stop	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

stop	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 varaible definition from the stream of tokens.

3.2.2.38 varList()

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

Parses a varaible list from the stream of tokens.

3.2.2.39 vPrime()

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

Parses a varaible vs array from the stream of tokens.

3.2.2.40 writeStmt()

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

Parses a write stamtement.

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 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 & 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 postion 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 varaibles to appropriate values.

Parameters

ifs	The file stream.
symboltable	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

inchar	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

inchar	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 postion 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

str	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

str	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

lexeme	The tokens's lexeme.
sym	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

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.

3.4.2.8 search()

```
Token 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 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

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

3.5.2 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

lexeme	The lexeme to give the token.

3.5.2.5 setSymbol()

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

Sets the symbol.

Parameters

sym	The symbol to give the token.

3.5.2.6 setVal()

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

Sets the value.

Parameters

val	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

S	The symbol set to check.
sym	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

stopSets	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