# Duck Interpreter

Salil Maharjan
Final Version
Wed May 15 2019

# Class Index

## Class List

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

# File Index

## File List

Here is a list of all files with brief descriptions:

# Class Documentation

## DuckInterpreter Class Reference

```
#include <DuckInterpreter.hpp>
```

Collaboration diagram for DuckInterpreter:

### Public Member Functions

**DuckInterpreter** ()
**~DuckInterpreter** ()
void **RecordStatements** (string a_fileName)
void **RunInterpreter** ()

### Private Types

enum **StatementType** { **StatementType::ArithmeticStat**, **StatementType::IfStat**, **StatementType::ReadStat**, **StatementType::PrintStat**, **StatementType::StopStat**, **StatementType::EndStat**, **StatementType::GotoStat**, **StatementType::CommentStat** }

### Private Member Functions

int **ExecuteStatement** (string a_statement, int a_nextStatement)

int **ParseNextElement** (const string &a_statement, int a_nextPos, string &a_stringValue, double &numValue)
**StatementType GetStatementStype** (const string &a_string)
bool **InsideQuotes** (string a_checkItem, string a_statement)
bool **IsCommented** (string a_checkItem, string a_string)
void **EvaluateArithmeticStatement** (const string &a_statement)
double **EvaluateArithmenticExpression** (const string &a_statement, int nextPos)
int **GetOperatorPrecedence** (const string a_operator)
double **DoOperation** (double a_val1, double a_val2, string a_operation)
bool **CheckNotOperator** (string &a_statement)
int **EvaluateIfStatement** (string a_statement, int a_nextStatement)
string **EvaluateIfGotoExpression** (string &a_statement)
void **EvaluateQuotedPrompt** (string &a_statement, bool &quoted, int &nextPos, string &resultString, double &placeHolder)
void **EvaluatePrintStatement** (string a_statement)
void **EvaluateReadStatement** (string a_statement)
int **EvaluateGotoStatement** (const string &a_statement)

## Private Attributes

**Statement m_statements**
**SymbolTable m_symbolTable**
vector< string > **m_operatorStack**
vector< double > **m_numberStack**

## Detailed Description

**DuckInterpreter.hpp DuckInterpreter** Class header file that runs the interpreter. Uses **Statement.hpp** and **SymbolTable.hpp**.

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved.

Definition at line 16 of file DuckInterpreter.hpp.

## Member Enumeration Documentation

### enum DuckInterpreter::StatementType `[strong]`, `[private]`

**Enumerator:**

| | |
|---|---|
| ArithmeticStat | |
| IfStat | |
| ReadStat | |
| PrintStat | |
| StopStat | |
| EndStat | |
| GotoStat | |
| CommentStat | |

Definition at line 46 of file DuckInterpreter.hpp.

## Constructor & Destructor Documentation

### DuckInterpreter::DuckInterpreter ()

**DuckInterpreter.cpp** Implementation of **DuckInterpreter.cpp**

Created by Salil D. Maharjan on 3/13/19. Copyright © 2019 Salil D. Maharjan. All rights reserved. **DuckInterpreter::DuckInterpreter**. Constructor for **DuckInterpreter** class. Empty.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 21 of file DuckInterpreter.cpp.

### DuckInterpreter::~DuckInterpreter ()

**DuckInterpreter::~DuckInterpreter**. Destructor for **DuckInterpreter** class. Empty.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 29 of file DuckInterpreter.cpp.

## Member Function Documentation

### bool DuckInterpreter::CheckNotOperator (string & *a_statement*)`[private]`

**DuckInterpreter::CheckNotOperator**. Method to check for not(!) operator in the statement. Checks and counts for not(!) operators in the statement and determines if we need to perform the not(!) operation. Handles cases when there are multiple not operators and determines if the not operation cancels out or not. Removes the ! operation recorded from the statement and returns a bool if we need to perform the operation or not.

**Parameters:**

| | |
|---|---|
| *a_statement* | const string Holds the arithmetic expression. |

**Returns:**

bool Whether we need to perform the not(!) operation.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 573 of file DuckInterpreter.cpp.

### double DuckInterpreter::DoOperation (double *a_val1*, double *a_val2*, string *a_operation*) `[private]`

**DuckInterpreter::DoOperation**. Method to perform arithmetic operations. Does 'a_operation' on 'a_val1' and 'a_val2'. Used by EvaluateArithmeticExpression.

**Parameters:**

| | |
|---|---|
| *a_val1* | double Holds the first value to perform operation with. |

| | |
|---|---|
| *a_val2* | double Holds the second value to perform operation with. |
| *a_operation* | string String that holds what operation is to be performed. |

**Returns:**

double The result of performing operation "a_operation" on "a_val1" and "a_val2".

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 414 of file DuckInterpreter.cpp.

### double DuckInterpreter::EvaluateArithmenticExpression (const string & *a_statement*, int *a_nextPos*) `[private]`

**DuckInterpreter::EvaluateArithmenticExpression**. Method to perform arithmetic operations. Evaluate an arithmetic expression that is after the assignment operator (=). Used by EvaluateArithmeticStatement.

**Parameters:**

| | |
|---|---|
| *a_statement* | const string Holds the entire arithmetic expression. |
| *a_nextPos* | int Holds the value of the first position after the assignment operator to start evaluating the statement. |

**Returns:**

double The final result of the arithmetic expression in "a_statement".

**See also:**

**ParseNextElement**
**DoOperation**
**GetOperatorPrecedence**
**SymbolTable::GetVariableValue**
**SymbolTable::RecordVariableValue**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 459 of file DuckInterpreter.cpp.

### void DuckInterpreter::EvaluateArithmeticStatement (const string & *a_statement*) `[private]`

**DuckInterpreter::EvaluateArithmeticStatement**. Method to Evaluate Arithmetic Statements. We know at this point that we have an arithementic expression. Execute this statement. Any error will perminate the program. The function first checks for unary arithmetic statements, if not the statement is evaluated by getting the result variable, the assignment operator and then evaluating the rest of the arithmetic statement by calling EvaluateArithmeticExpression

**Parameters:**

| | |
|---|---|
| *a_statement* | const string String to find out the statement type of. |

**See also:**

EvaluateArithmeticExpression

**SymbolTable::GetVariableValue**
**SymbolTable::RecordVariableValue**

**Author:**
Salil Maharjan

**Date:**
03/13/19

Definition at line 311 of file DuckInterpreter.cpp.

### int DuckInterpreter::EvaluateGotoStatement (const string & *a_statement*)`[private]`

**DuckInterpreter::EvaluateGotoStatement**. Method to evaluate Goto statements. Evaluates goto statements. At this point, we know that it is a goto statement. The label is the second syntatic element. We parse it and verify that it is a valid label and return the corresponding value of its location.

**Parameters:**

| | |
|---|---|
| *a_statement* | const string Holds the goto statement. |

**Returns:**
int Position of the label specified in the goto statement.

**See also:**
**ParseNextElement**
**Statement::GetLabelLocation**
**Statement::GetStatement**

**Author:**
Salil Maharjan

**Date:**
03/13/19

Definition at line 866 of file DuckInterpreter.cpp.

### string DuckInterpreter::EvaluateIfGotoExpression (string & *a_statement*)`[private]`

**DuckInterpreter::EvaluateIfGotoExpression**. Method to evaluate goto expressions in an If statement. Searches till the end of the statement for the label and returns it. Then replaces the goto statement with ";"

**Parameters:**

| | |
|---|---|
| *a_statement* | string Holds the if statement. |

**Returns:**
string The label associated with the goto statement.

**Author:**
Salil Maharjan

**Date:**
03/13/19

Definition at line 662 of file DuckInterpreter.cpp.

### int DuckInterpreter::EvaluateIfStatement (string *a_statement*, int *a_nextStatement*) `[private]`

**DuckInterpreter::EvaluateIfStatement**. Method to evaluate If Statements. Evaluates an if statement to determine if the goto should be executed.

**Parameters:**

| | |
|---|---|
| *a_statement* | const string Holds the if statement. |
| *a_nextStatement* | int Current statement position. Used to return the next position of the statement. |

**Returns:**

int The next position of the statement that needs to be executed.

**See also:**

**ParseNextElement**
EevaluateIfGotoExpression
**CheckNotOperator**
EvaluateArithmeticExpression

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 615 of file DuckInterpreter.cpp.

### void DuckInterpreter::EvaluatePrintStatement (string *a_statement*)`[private]`

**DuckInterpreter::EvaluatePrintStatement**. Method to evaluate Print statements. Evaluates print statements. Asserts for "print" keyword, prints the quoted prompts and variables that are comma separated in the code.

**Parameters:**

| | |
|---|---|
| *a_statement* | string Holds the print statement. |

**See also:**

**ParseNextElement**
**EvaluateQuotedPrompt**
**SymbolTable::GetVariableValue**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 761 of file DuckInterpreter.cpp.

### void DuckInterpreter::EvaluateQuotedPrompt (string & *a_statement*, bool & *quoted*, int & *nextPos*, string & *resultString*, double & *placeHolder*)`[private]`

**DuckInterpreter::EvaluateQuotedPrompt**. Method to evaluate quoted prompts. Function that evaluates quoted prompts. Used by EvaluatePrintStatement and EvaluateReadStatement to evaluate prompts in quotations. Parameters are passed by reference from the calling function.

**Parameters:**

| | |
|---|---|
| *a_statement* | string Holds the statement with quoted prompt. |
| *quoted* | bool flag used to track quoted part of the statement. |
| *nextPos* | int Position of the current element in the statement where we need to start evaluating from. |
| *resultString* | string Position holder to get string element from ParseNextElement |
| *placeHolder* | double Position holder to get numeric element from ParseNextElement. |

**See also:**

**ParseNextElement**
**EvaluatePrintStatement**
**EvaluateReadStatement**
**Statement::StandardSpaceFormat**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 711 of file DuckInterpreter.cpp.

### void DuckInterpreter::EvaluateReadStatement (string *a_statement*) `[private]`

**DuckInterpreter::EvaluateReadStatement**. Method to evaluate Read statements. Evaluates read statements. Asserts for "read" keyword, prints the quoted prompts and gets input from the user for the specified variables.

**Parameters:**

| | |
|---|---|
| *a_statement* | string Holds the read statement. |

**See also:**

**ParseNextElement**
**EvaluateQuotedPrompt**
**SymbolTable::GetVariableValue**
**SymbolTable::RecordVariableValue**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 810 of file DuckInterpreter.cpp.

### int DuckInterpreter::ExecuteStatement (string *a_statement*, int *a_nextStatement*) `[private]`

**DuckInterpreter::ExecuteStatement**. Method to execute a statement. Gets the statement type of a_statement and executes it by calling the respective functions.

**Parameters:**

| | |
|---|---|
| *a_statement* | string The statement to be executed. |
| *a_nextStatement* | int Current statement number. |

**Returns:**

int Next statement number to execute.

**See also:**

**GetStatementStype**
**EvaluateArithmeticStatement**
**EvaluateIfStatement**
**EvaluatePrintStatement**
**EvaluateReadStatement**
**EvaluateGotoStatement**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 67 of file DuckInterpreter.cpp.

### int DuckInterpreter::GetOperatorPrecedence (const string *a_operator*) `[private]`

**DuckInterpreter::GetOperatorPrecedence**. Method to get the operator precedence. Checks a_operator and returns the precedence of the operator. Used by EvaluateArithmeticExpression.

**Parameters:**

| | |
|---|---|
| *a_operator* | const string String to find out the statement type of. |

**Returns:**

int The precedence of the passed operator 'a_operator'

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 388 of file DuckInterpreter.cpp.

### DuckInterpreter::StatementType DuckInterpreter::GetStatementStype (const string & *a_string*) `[private]`

**DuckInterpreter::GetStatementStype**. Method that gets the **Statement** type. Finds the statement type of a_string and returns it. Checks for specific identifiers that each statement has, does checks for comments and quotes and determines the statement type.

**Parameters:**

| | |
|---|---|
| *a_string* | const string String to find out the statement type of. |

**Returns:**

**DuckInterpreter::StatementType** The **Statement** Type of the passed string.

**See also:**

**InsideQuotes**
**IsCommented**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 228 of file DuckInterpreter.cpp.

### bool DuckInterpreter::InsideQuotes (string *a_checkItem*, string *a_string*) `[private]`

**DuckInterpreter::InsideQuotes**. Quote checker method. Checks if a_checkItem, which is a substring of a_string, is inside quotes in the string.

**Parameters:**

| | |
|---|---|
| *a_checkItem* | string Item to check if it is inside quotation. |
| *a_string* | string The main string that has check_item in it. |

**Returns:**

bool True/False whether a_checkItem is inside quotations or not.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 181 of file DuckInterpreter.cpp.

#### bool DuckInterpreter::IsCommented (string *a_checkItem*, string *a_string*)`[private]`

**DuckInterpreter::IsCommented**. Comment checker method. Checks if a_checkItem, which is a substring of a_string, is commented or not.

**Parameters:**

| | |
|---|---|
| *a_checkItem* | string Item to check if it is commented. |
| *a_string* | string The main string that has check_item in it. |

**Returns:**

bool True/False whether a_checkItem is after comments or not.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 204 of file DuckInterpreter.cpp.

#### int DuckInterpreter::ParseNextElement (const string & *a_statement*, int *a_nextPos*, string & *a_stringValue*, double & *numValue*)`[private]`

**DuckInterpreter::ParseNextElement**. Element Parse Method. Parses the next element of the statement. Uses istringstream to get each element separated by a white space.

**Parameters:**

| | |
|---|---|
| *a_statement* | const string The statement to parse. |
| *a_nextPos* | int The element number of the element to parse from the statement. |
| *a_stringValue* | string Captures the string elements from statement. "NULL" if the element is not a string. |
| *numValue* | double Captures numeric elements from the statement. INT_MAX if the element is not a number. |

**Returns:**

int Position of the next element.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 126 of file DuckInterpreter.cpp.

#### void DuckInterpreter::RecordStatements (string *a_fileName*)`[inline]`

Definition at line 24 of file DuckInterpreter.hpp.

#### void DuckInterpreter::RunInterpreter ()

**DuckInterpreter::RunInterpreter**. Method to run the interpreter. Gets the statements to execute until completed.

**See also:**

**Statement::GetStatement**
**ExecuteStatement**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 39 of file DuckInterpreter.cpp.

## Member Data Documentation

### vector<double> DuckInterpreter::m_numberStack `[private]`

Definition at line 43 of file DuckInterpreter.hpp.

### vector<string> DuckInterpreter::m_operatorStack `[private]`

Definition at line 41 of file DuckInterpreter.hpp.

### Statement DuckInterpreter::m_statements `[private]`

Definition at line 35 of file DuckInterpreter.hpp.

### SymbolTable DuckInterpreter::m_symbolTable `[private]`

Definition at line 38 of file DuckInterpreter.hpp.

### The documentation for this class was generated from the following files:

DuckInterp/DuckInterp/**DuckInterpreter.hpp**
DuckInterp/DuckInterp/**DuckInterpreter.cpp**

# Statement Class Reference

```
#include <Statement.hpp>
```

## Public Member Functions

**Statement** ()
**~Statement** ()
void **RecordStatements** (string a_sourceFileName)
string **GetStatement** (int a_statementNum)
int **GetLabelLocation** (string a_string)

## Private Member Functions

void **StandardSpaceFormat** (string &a_string)
bool **NeedSpace** (char element)

## Private Attributes

vector< string > **m_statements**
vector< string >::iterator **itr**
map< string, int > **m_labelToStatement**
map< string, int >::iterator **mpi**

## Detailed Description

**Statement.hpp Statement** Class header file. Records and provides statements and labels.

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved.

Definition at line 14 of file Statement.hpp.

## Constructor & Destructor Documentation

### Statement::Statement ()

**DuckInterpreter.cpp** Implementation of **Statement.hpp**

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved. **Statement::Statement**. Constructor for **Statement** class. Empty.

**Author:**
Salil Maharjan

**Date:**
03/13/19

Definition at line 21 of file Statement.cpp.

### Statement::~Statement ()

**Statement::~Statement**. Destructor for **Statement** class. Empty.

**Author:**
Salil Maharjan

**Date:**
03/13/19

Definition at line 29 of file Statement.cpp.

## Member Function Documentation

### int Statement::GetLabelLocation (string *a_string*)

**Statement::GetLabelLocation**. Accessor to get label location. Tries to get the location of the label "a_string" from the map "m_labeelToStatement". Throws invalid label error if label is not found.

**Parameters:**

| | |
|---|---|
| *a_string* | string Holds the name of the label |

**Returns:**
int The position of the label in the code if found.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 135 of file Statement.cpp.

### string Statement::GetStatement (int *a_statementNum*)`[inline]`

Definition at line 24 of file Statement.hpp.

### bool Statement::NeedSpace (char *element*)`[private]`

**Statement::NeedSpace**. Method to check if it is a character we need to cheeck spaces for. True if we need to check space for the character, False otherwise. Also used to check if it is safe to insert a space in the position of 'element'. If false, we insert a space. If true, we do nothing.

**Parameters:**

| | |
|---|---|
| *element* | char Element to check if is a character we need to check spaces for. |

**Returns:**

bool True/False if it is a character we need to check spaces for.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 161 of file Statement.cpp.

### void Statement::RecordStatements (string *a_sourceFileName*)

**Statement::RecordStatements**. Method to record statements from a source file. Records code statements on each line from file "a_sourceFileName". Every time it gets a line from the file, the method checks if there are any labels in the statement. If there are, it records them in "m_labelToStatement" with the label name as key and associated line number as value. After recording the label, it removes it from the statement. After removing the label, it calls StandardSpaceFormat method to ensure that consistent formatting is followed in the code. After standardizing the spaces, the method records the statement in the vector "m_statements".

**Parameters:**

| | |
|---|---|
| *a_sourceFileName* | string File that has the duck code statements. |

**See also:**

**StandardSpaceFormat**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 43 of file Statement.cpp.

### void Statement::StandardSpaceFormat (string & *a_string*)`[private]`

**Statement::StandardSpaceFormat**. Method to standardize spaces for the code statements. Get a uniform space formatting for the statements of the Duck language. Uses method NeedSpace to check for characters that need space considerations.

**Parameters:**

| | |
|---|---|
| *a_string* | string Holds the code statement to standardize spaces. |

**See also:**

**NeedSpace**

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 175 of file Statement.cpp.

## Member Data Documentation

#### vector<string>::iterator Statement::itr `[private]`

Definition at line 41 of file Statement.hpp.

#### map<string,int> Statement::m_labelToStatement `[private]`

Definition at line 44 of file Statement.hpp.

#### vector<string> Statement::m_statements `[private]`

Definition at line 40 of file Statement.hpp.

#### map<string, int>::iterator Statement::mpi `[private]`

Definition at line 45 of file Statement.hpp.

#### The documentation for this class was generated from the following files:

DuckInterp/DuckInterp/**Statement.hpp**
DuckInterp/DuckInterp/**Statement.cpp**

## SymbolTable Class Reference

`#include <SymbolTable.hpp>`

### Public Member Functions

**SymbolTable** ()
**~SymbolTable** ()

void **RecordVariableValue** (string a_variable, double a_value)
bool **GetVariableValue** (string a_variable, double &a_value)

## Private Attributes

unordered_map< string, double > **m_SymbolTable**

## Detailed Description

**SymbolTable.hpp SymbolTable** Class header file. Records value of variables for access. This class will provide a mapping between the variables and their associated data.

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved.

Definition at line 13 of file SymbolTable.hpp.

## Constructor & Destructor Documentation

### SymbolTable::SymbolTable ()

**SymbolTable.cpp** Implementation of **SymbolTable.hpp**

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved. **SymbolTable::SymbolTable**. Constructor for **SymbolTable** class. Empty.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 20 of file SymbolTable.cpp.

### SymbolTable::~SymbolTable ()

**SymbolTable::~SymbolTable**. Destructor for **SymbolTable** class. Empty.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 28 of file SymbolTable.cpp.

## Member Function Documentation

### bool SymbolTable::GetVariableValue (string *a_variable*, double & *a_value*)

**SymbolTable::GetVariableValue**. Accessor to get the variable value. Checks if the variable is in the map and return its corresponding value. Returns false if the value is not found.

**Author:**

Salil Maharjan

**Date:**

03/13/19

Definition at line 37 of file SymbolTable.cpp.

**void SymbolTable::RecordVariableValue (string *a_variable*, double *a_value*)**`[inline]`

Definition at line 20 of file SymbolTable.hpp.

### Member Data Documentation

**unordered_map<string, double> SymbolTable::m_SymbolTable**`[private]`

Definition at line 30 of file SymbolTable.hpp.

**The documentation for this class was generated from the following files:**

DuckInterp/DuckInterp/**SymbolTable.hpp**
DuckInterp/DuckInterp/**SymbolTable.cpp**

# File Documentation

## DuckInterp/DuckInterp/DuckInterpreter.cpp File Reference

```
#include "DuckInterpreter.hpp"
#include "SymbolTable.hpp"
#include "PrefixHeader.pch"
```

Include dependency graph for DuckInterpreter.cpp:

## DuckInterp/DuckInterp/DuckInterpreter.hpp File Reference

```
#include "PrefixHeader.pch"
#include "Statement.hpp"
#include "SymbolTable.hpp"
```

Include dependency graph for DuckInterpreter.hpp:

This graph shows which files directly or indirectly include this file:

### Classes

class **DuckInterpreter**

## DuckInterp/DuckInterp/main.cpp File Reference

```
#include "PrefixHeader.pch"
#include "DuckInterpreter.hpp"
```

Include dependency graph for main.cpp:

### Functions

int **main** (int argc, char *argv[])

### Function Documentation

#### int main (int *argc*, char * *argv*[])

**main.cpp** Main for the Duck Interpreter. Defines the entry point. Final Project.

Created by Salil Maharjan on 3/13/19. Copyright © 2019 Salil Maharjan. All rights reserved.

Definition at line 14 of file main.cpp.

## DuckInterp/DuckInterp/Statement.cpp File Reference

```
#include "Statement.hpp"
#include "PrefixHeader.pch"
#include "DuckInterpreter.hpp"
```

Include dependency graph for Statement.cpp:

## DuckInterp/DuckInterp/Statement.hpp File Reference

```
#include "PrefixHeader.pch"
```

Include dependency graph for Statement.hpp:

This graph shows which files directly or indirectly include this file:

### Classes

class **Statement**

## DuckInterp/DuckInterp/SymbolTable.cpp File Reference

```
#include "PrefixHeader.pch"
#include "SymbolTable.hpp"
```

Include dependency graph for SymbolTable.cpp:

## DuckInterp/DuckInterp/SymbolTable.hpp File Reference

This graph shows which files directly or indirectly include this file:

### Classes

class **SymbolTable**