

Scanner

<https://github.com/cata-b/FLCDDCompiler>

Generated by Doxygen 1.8.16

1 Class Documentation	1
1.1 LexicalAnalyzer Class Reference	1
1.1.1 Member Function Documentation	2
1.1.2 Member Data Documentation	2
1.2 Set< KeyType, Hash, Equals > Class Template Reference	2
1.2.1 Detailed Description	4
1.2.2 Constructor & Destructor Documentation	4
1.2.3 Member Function Documentation	5
1.2.4 Member Data Documentation	6
1.3 Set< KeyType, Hash, Equals >::Iterator Class Reference	7
1.3.1 Detailed Description	8
1.4 SymbolTable Class Reference	8
1.4.1 Detailed Description	9
1.4.2 Member Function Documentation	9
1.5 SymbolTable::Position Class Reference	9
1.5.1 Detailed Description	10
1.6 SymbolTable::Position::Data Struct Reference	11
1.7 Token Struct Reference	11
1.8 Tokenizer Class Reference	11
1.8.1 Member Function Documentation	11
1.9 Tokenizer::Error Class Reference	14
Index	15

1 Class Documentation

1.1 LexicalAnalyzer Class Reference

Public Types

- enum **TokenType** {
KEYWORD, CONSTANT, OPERATOR, SEPARATOR,
IDENTIFIER, ERROR }

Static Public Member Functions

- static std::vector< [Token](#) > [Analyze](#) (std::vector< [Token](#) > tokens, [SymbolTable](#) &symbolTable, std::vector< std::tuple< [Token](#), TokenType, [SymbolTable::Position](#) >> &pif)
Analyzes and classifies tokens

Static Private Attributes

- static const std::vector< std::pair< std::regex, TokenType > > [TOKEN_CLASSES](#)
Maps a regular expression to a token type; the order of the elements is such that, if checking in order, tokens that would match more entries will match the correct entry (e.g. true will be classified as a constant, rather than an identifier)

1.1.1 Member Function Documentation

1.1.1.1 Analyze() `vector< Token > LexicalAnalyzer::Analyze (`
`std::vector< Token > tokens,`
`SymbolTable & symbolTable,`
`std::vector< std::tuple< Token, TokenType, SymbolTable::Position >> & pif)`
`[static]`

Analyzes and classifies tokens

Parameters

<i>tokens</i>	Tokens, as returned by Tokenizer::tokenize
<i>symbolTable</i>	Output parameter that will contain all the identifiers and constants from the tokens
<i>pif</i>	Output parameter that will contain tokens, their type, and their positions in the symbol table (if a token is not inserted in the symbol table, the position is the end of that table)

Returns

All the tokens that could not be classified

1.1.2 Member Data Documentation

1.1.2.1 TOKEN_CLASSES `const vector< pair< regex, LexicalAnalyzer::TokenType > > LexicalAnalyzer::TOKEN_CLASSES [static], [private]`

Initial value:

```
= {
    { regex(R"(int|uint|bool|string|read|print|error|rand|exit|for|while|if|else|break|continue)"),
      TokenType::KEYWORD },
    { regex(R"((?:[1-9]+[0-9]*)|0)"), TokenType::CONSTANT },
    { regex(R"((?:\+|\-)[1-9]+[0-9]*)"), TokenType::CONSTANT },
    { regex(R"(true|false)"), TokenType::CONSTANT },
    { regex(R"(\\"{a-zA-Z0-9\\+\\-*/%\\=<>[\\]}()?!_.|&*^",':; \\t}*)"), TokenType::CONSTANT },
    { regex(R"(=|!=|>|=|<=|\\|\\|&&|\\|+|-|\\*|\\/|\\%|=|<|>|^|\\|&|!|)"), TokenType::OPERATOR },
    { regex(R"((\\|\\|\\|\\|\\|\\|\\|'|:|;)"), TokenType::SEPARATOR },
    { regex(R"([a-zA-Z_][a-zA-Z0-9_]*)"), TokenType::IDENTIFIER }
}
```

Maps a regular expression to a token type; the order of the elements is such that, if checking in order, tokens that would match more entries will match the correct entry (e.g. true will be classified as a constant, rather than an identifier)

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

- LexicalAnalyzer.h
- LexicalAnalyzer.cpp

1.2 Set< KeyType, Hash, Equals > Class Template Reference

Hashtable with coalesced chaining

Classes

- class [Iterator](#)

Const iterator for a [Set](#). May become invalid/give errors after inserting data into the set; to check if an insertion would render the iterators invalid, use [willInvalidateIteratorsOnInsert\(\)](#)

Public Types

- typedef [Iterator](#) **const_iterator_type**

Public Member Functions

- [Set](#) ()
Initializes an empty [Set](#) with initial capacity 1
- [Set](#) (size_t initial_size)
Initializes an empty [Set](#) with a custom initial capacity
- std::pair< [Iterator](#), bool > **insert** (const KeyType &item)
Inserts a copy of an item into the set
- std::pair< [Iterator](#), bool > **insert** (KeyType &&item)
Inserts an item into the set
- [Iterator](#) **find** (const KeyType &item) const
Searches for an item in the set
- void **erase** ([Iterator](#) iterator)
Removes the item at an iterator
- size_t **size** () const
Returns
The number of items in the set
- size_t **capacity** () const
- bool **willInvalidateIteratorsOnInsert** () const
Returns
true if the next insert would trigger a resize, therefore invalidating the iterators
- [Iterator](#) **begin** () const
- [Iterator](#) **end** () const

Private Types

- enum **State** { **EMPTY**, **OCCUPIED**, **DELETED** }

Private Member Functions

- void **resize** (size_t newCapacity)
- template<typename refType >
std::pair< [Iterator](#), bool > **insert_** (refType item)
- template<typename refType >
std::pair< typename [Set](#)< KeyType, Hash, Equals >:: [Iterator](#), bool > **insert_** (refType item)

Private Attributes

- `int * next_`
Array of positions – each element points to the next one in the "bucket"
- `KeyType * data_`
- `int64_t firstEmpty_`
First empty (not deleted) position
- `State * state_`
- `size_t size_`
*The number of
State::OCCUPIED
cells*
- `size_t capacity_`
- `Hash hash_`
- `Equals equals_`

1.2.1 Detailed Description

```
template<class KeyType, class Hash = std::hash<KeyType>, class Equals = std::equal_to<KeyType>>
class Set< KeyType, Hash, Equals >
```

Hashtable with coalesced chaining

Template Parameters

<i>KeyType</i>	The type of the items held in this set
<i>Hash</i>	Hashing functor for an item of type KeyType
<i>Equals</i>	Equality functor for an item of type KeyType

1.2.2 Constructor & Destructor Documentation

1.2.2.1 Set() [1/2] `template<class KeyType, class Hash = std::hash<KeyType>, class Equals = std::equal_to<KeyType>> Set< KeyType, Hash, Equals >::Set ()`

Initializes an empty `Set` with initial capacity 1

1.2.2.2 Set() [2/2] `template<class KeyType , class Hash , class Equals > Set< KeyType, Hash, Equals >::Set (size_t initial_size)`

Initializes an empty `Set` with a custom initial capacity

Parameters

<i>initial_size</i>	The initial capacity
---------------------	----------------------

1.2.3 Member Function Documentation

1.2.3.1 insert() [1/2] `template<class KeyType, class Hash , class Equals >
std::pair< typename Set< KeyType, Hash, Equals >::Iterator, bool > Set< KeyType, Hash, Equals
>::insert (
 const KeyType & item)`

Inserts a copy of an item into the set

Parameters

<i>item</i>	The item to insert
-------------	--------------------

Returns

An iterator to an item equal to the parameter and whether the insertion happened (true) or a similar item was already in the set (false)

1.2.3.2 insert() [2/2] `template<class KeyType, class Hash , class Equals >
std::pair< typename Set< KeyType, Hash, Equals >::Iterator, bool > Set< KeyType, Hash, Equals
>::insert (
 KeyType && item)`

Inserts an item into the set

Parameters

<i>item</i>	The item to insert
-------------	--------------------

Returns

An iterator to an item equal to the parameter and whether the insertion happened (true) or a similar item was already in the set (false)

1.2.3.3 find() `template<class KeyType, class Hash , class Equals >
Set< KeyType, Hash, Equals >::Iterator Set< KeyType, Hash, Equals >::find (
 const KeyType & item) const`

Searches for an item in the set

Parameters

<i>item</i>	The item to search for
-------------	------------------------

Returns

An iterator to the similar item in the set or end()

```
1.2.3.4 erase() template<class KeyType , class Hash , class Equals >
void Set< KeyType, Hash, Equals >::erase (
    Iterator iterator )
```

Removes the item at an iterator

Parameters

<i>iterator</i>	The position of the item to remove
-----------------	------------------------------------

Throws `std::runtime_error` if the iterator points outside the container (e.g. is equal to end())

```
1.2.3.5 capacity() template<class KeyType, class Hash = std::hash<KeyType>, class Equals =
std::equal_to<KeyType>>
size_t Set< KeyType, Hash, Equals >::capacity ( ) const
```

Returns

The capacity of the set

The number of insertions (without subtracting deletions) is compared with the capacity when resizing

1.2.4 Member Data Documentation

```
1.2.4.1 next_ template<class KeyType, class Hash = std::hash<KeyType>, class Equals = std::
::equal_to<KeyType>>
int* Set< KeyType, Hash, Equals >::next_ [private]
```

Array of positions – each element points to the next one in the "bucket"

```
1.2.4.2 firstEmpty_ template<class KeyType, class Hash = std::hash<KeyType>, class Equals =
std::equal_to<KeyType>>
int64_t Set< KeyType, Hash, Equals >::firstEmpty_ [private]
```

First empty (not deleted) position

```

1.2.4.3 size_ template<class KeyType, class Hash = std::hash<KeyType>, class Equals = std::equal_to<KeyType>>
size_t Set< KeyType, Hash, Equals >::size_ [private]

```

The number of
State::OCCUPIED

cells

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

- Set.hpp

1.3 Set< KeyType, Hash, Equals >::Iterator Class Reference

Const iterator for a [Set](#). May become invalid/give errors after inserting data into the set; to check if an insertion would render the iterators invalid, use [willInvalidateIteratorsOnInsert\(\)](#)

Public Types

- typedef std::input_iterator_tag **iterator_category**
- typedef KeyType **value_type**
- typedef std::ptrdiff_t **difference_type**

Public Member Functions

- **Iterator** (const KeyType *data_begin, const State *state_begin, size_t index, size_t max)
- **Iterator** (const [Iterator](#) &other)
- **Iterator** ([Iterator](#) &&other)
- [Iterator](#) & **operator++** ()
- [Iterator](#) **operator++** (int)
- [Iterator](#) & **operator=** (const [Iterator](#) &other)
- [Iterator](#) & **operator=** ([Iterator](#) &&other)
- bool **operator==** ([Iterator](#) other) const
- bool **operator!=** ([Iterator](#) other) const
- const reference **operator*** () const
- const pointer **operator->** () const
- size_t **index** () const

Public Attributes

- const typedef KeyType * **pointer**
- const typedef KeyType & **reference**

Private Attributes

- const KeyType * **data_begin_**
- const State * **state_begin_**
- size_t **index_**
- size_t **max_**

Friends

- class **Set**

1.3.1 Detailed Description

```
template<class KeyType, class Hash = std::hash<KeyType>, class Equals = std::equal_to<KeyType>>
class Set< KeyType, Hash, Equals >::iterator
```

Const iterator for a [Set](#). May become invalid/give errors after inserting data into the set; to check if an insertion would render the iterators invalid, use [willInvalidateIteratorsOnInsert\(\)](#)

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

- Set.hpp

1.4 SymbolTable Class Reference

Symbol table data structure. Features iterators that will sometimes get updated on insert, so that they are never invalid

Classes

- class [Position](#)

Symbol table const iterator. Will still be valid after inserts in the parent [SymbolTable](#)

Public Member Functions

- `std::pair< Position, bool > insert (const std::string &symbol)`
Inserts a symbol into the symbol table
- `Position find (const std::string &symbol)`
Searches for an entry in the [SymbolTable](#)
- `size_t size () const`
- `Position begin ()`
- `Position end ()`

Private Member Functions

- void **subscribe** ([Position::Data](#) *subscriber)
- void **unsubscribe** ([Position::Data](#) *subscriber)

Private Attributes

- bool **destructed_** = false
- [Set](#)< std::string > **data_**
- std::unordered_set< [Position::Data](#) * > **positions_**

1.4.1 Detailed Description

Symbol table data structure. Features iterators that will sometimes get updated on insert, so that they are never invalid

1.4.2 Member Function Documentation

1.4.2.1 insert() `std::pair< typename SymbolTable::Position, bool > SymbolTable::insert (const std::string & symbol)`

Inserts a symbol into the symbol table

Parameters

<i>symbol</i>	The symbol to insert
---------------	----------------------

Returns

A [Position](#) of where the symbol is and whether or not the insertion happened; if a similar symbol already exists in the table, the [Position](#) points to that

1.4.2.2 find() `SymbolTable::Position SymbolTable::find (const std::string & symbol)`

Searches for an entry in the [SymbolTable](#)

Parameters

<i>symbol</i>	The symbol to look for
---------------	------------------------

Returns

[Position](#) to the symbol or end() if it was not found

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

- SymbolTable.h
- SymbolTable.cpp

1.5 SymbolTable::Position Class Reference

Symbol table const iterator. Will still be valid after inserts in the parent [SymbolTable](#)

Classes

- struct [Data](#)

Public Types

- typedef std::input_iterator_tag **iterator_category**
- typedef std::ptrdiff_t **difference_type**

Public Member Functions

- **Position** ([Position](#) &&other)
- **Position** (const [Position](#) &other)
- **Position** ([SymbolTable](#) *parent, [Set](#)< std::string >::Iterator iterator)
- [SymbolTable](#) & **symbolTable** () const
- [Position](#) & **operator=** (const [Position](#) &other)
- [Position](#) & **operator=** ([Position](#) &&other)
- [Position](#) & **operator++** ()
- [Position](#) **operator++** (int)
- bool **operator==** ([Position](#) other) const
- bool **operator!=** ([Position](#) other) const
- reference **operator*** () const
- pointer **operator->** () const
- size_t **index** () const

Public Attributes

- const typedef std::string **value_type**
- const typedef std::string * **pointer**
- const typedef std::string & **reference**

Private Attributes

- [Data](#) * **data_** = nullptr

Friends

- class **SymbolTable**

1.5.1 Detailed Description

Symbol table const iterator. Will still be valid after inserts in the parent [SymbolTable](#)

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

- SymbolTable.h
- SymbolTable.cpp

1.6 SymbolTable::Position::Data Struct Reference

Public Attributes

- [SymbolTable](#) * **parent_**
- [Set](#)< std::string >::iterator **iterator_**

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

- SymbolTable.h

1.7 Token Struct Reference

Public Attributes

- std::string **content**
- size_t **line**

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

- Token.h

1.8 Tokenizer Class Reference

Classes

- class [Error](#)

Static Public Member Functions

- static std::vector< [Token](#) > [tokenize](#) (std::string filename)
Reads input from a file and splits it into tokens

Static Private Member Functions

- static std::vector< [Token](#) > [splitTokensFromSeparators](#) (std::string filename)
Reads a file and (partially) splits it into tokens
- static std::vector< [Token](#) > [splitSeparators](#) ([Token](#) input)
Splits the strings that are made of separators into individual tokens
- static std::vector< [Token](#) > [removeComments](#) (std::vector< [Token](#) > tokenizedWithSplitSeparators)
Removes sequences of tokens that begin with '/' and end with '
,
- static std::vector< [Token](#) > [combineStringLiterals](#) (std::vector< [Token](#) > tokenizedWithoutComments)
Combines token sequences that are on a single line and begin and end with "
- static std::vector< [Token](#) > [removeWhitespaces](#) (std::vector< [Token](#) > tokenizedWithStringLiterals)
Removes entries that are whitespaces

1.8.1 Member Function Documentation

1.8.1.1 tokenize() `vector< Token > Tokenizer::tokenize (std::string filename) [static]`

Reads input from a file and splits it into tokens

Parameters

<i>filename</i>	The name of the file to read
-----------------	------------------------------

Exceptions

<i>Error</i>	Thrown when the tokenizer cannot read a file or cannot split the content into tokens correctly
--------------	--

Returns

A vector of tokens

1.8.1.2 splitTokensFromSeparators() `vector< Token > Tokenizer::splitTokensFromSeparators (std::string filename) [static], [private]`

Reads a file and (partially) splits it into tokens

Parameters

<i>filename</i>	The name of the file to read
-----------------	------------------------------

Returns

Tokens that may be either strings of separators or strings of not separators

1.8.1.3 splitSeparators() `vector< Token > Tokenizer::splitSeparators (Token input) [static], [private]`

Splits the strings that are made of separators into individual tokens

Parameters

<i>input</i>	<code>Token</code> returned by <code>splitTokensFromSeparators</code>
--------------	---

Returns

Tokens where all separators are a separate token (taking into account 2-character operators, e.g. >=)

1.8.1.4 removeComments() `vector< Token > Tokenizer::removeComments (std::vector< Token > tokenizedWithSplitSeparators) [static], [private]`

Removes sequences of tokens that begin with `/*` and end with `*/`

Parameters

<i>tokenizedWithSplitSeparators</i>	Output of splitSeparators
-------------------------------------	---

Returns

The modified tokens

1.8.1.5 combineStringLiterals() `vector< Token > Tokenizer::combineStringLiterals (`
`std::vector< Token > tokenizedWithoutComments) [static], [private]`

Combines token sequences that are on a single line and begin and end with "

Parameters

<i>tokenizedWithoutComments</i>	Output of removeComments
---------------------------------	--

Exceptions

Error	Thrown when a string constant doesn't end on the same line or doesn't end at all
-----------------------	--

Returns

Vector of tokens where string literals are single tokens

1.8.1.6 removeWhitespaces() `vector< Token > Tokenizer::removeWhitespaces (`
`std::vector< Token > tokenizedWithStringLiterals) [static], [private]`

Removes entries that are whitespaces

Parameters

<i>tokenizedWithStringLiterals</i>	Vector of tokens that contains spaces, tabs or newlines as tokens
------------------------------------	---

Returns

The tokens that are not whitespaces

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

- Tokenizer.h
- Tokenizer.cpp

1.9 Tokenizer::Error Class Reference

Public Member Functions

- **Error** (std::string what, [Token](#) token)
- **Error** (std::string what)
- bool **hasToken** ()
- [Token](#) **token** ()

Private Attributes

- [Token](#) **token_**
- bool **hasToken_** = false

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

- Tokenizer.h

Index

- Analyze
 - LexicalAnalyzer, [2](#)
- capacity
 - Set< KeyType, Hash, Equals >, [6](#)
- combineStringLiterals
 - Tokenizer, [13](#)
- erase
 - Set< KeyType, Hash, Equals >, [6](#)
- find
 - Set< KeyType, Hash, Equals >, [5](#)
 - SymbolTable, [9](#)
- firstEmpty_
 - Set< KeyType, Hash, Equals >, [6](#)
- insert
 - Set< KeyType, Hash, Equals >, [5](#)
 - SymbolTable, [9](#)
- LexicalAnalyzer, [1](#)
 - Analyze, [2](#)
 - TOKEN_CLASSES, [2](#)
- next_
 - Set< KeyType, Hash, Equals >, [6](#)
- removeComments
 - Tokenizer, [12](#)
- removeWhitespaces
 - Tokenizer, [13](#)
- Set
 - Set< KeyType, Hash, Equals >, [4](#)
- Set< KeyType, Hash, Equals >, [2](#)
 - capacity, [6](#)
 - erase, [6](#)
 - find, [5](#)
 - firstEmpty_, [6](#)
 - insert, [5](#)
 - next_, [6](#)
 - Set, [4](#)
 - size_, [6](#)
- Set< KeyType, Hash, Equals >::Iterator, [7](#)
- size_
 - Set< KeyType, Hash, Equals >, [6](#)
- splitSeparators
 - Tokenizer, [12](#)
- splitTokensFromSeparators
 - Tokenizer, [12](#)
- SymbolTable, [8](#)
 - find, [9](#)
 - insert, [9](#)
- SymbolTable::Position, [9](#)
- SymbolTable::Position::Data, [11](#)

- Token, [11](#)
- TOKEN_CLASSES
 - LexicalAnalyzer, [2](#)
- tokenize
 - Tokenizer, [11](#)
- Tokenizer, [11](#)
 - combineStringLiterals, [13](#)
 - removeComments, [12](#)
 - removeWhitespaces, [13](#)
 - splitSeparators, [12](#)
 - splitTokensFromSeparators, [12](#)
 - tokenize, [11](#)
- Tokenizer::Error, [14](#)