

GraphEx

1.0

Generated by Doxygen 1.9.1

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 TokenData Struct Reference	5
3.1.1 Detailed Description	5
4 File Documentation	7
4.1 parser.c File Reference	7
4.1.1 Detailed Description	8
4.1.2 Function Documentation	8
4.1.2.1 is_compare_op()	8
4.1.2.2 is_expression()	8
4.1.2.3 is_instruction()	9
4.1.2.4 is_operation_param()	9
4.1.2.5 match()	9
4.1.2.6 operations_routine()	9
4.1.2.7 parse_declare()	10
4.1.2.8 parse_graph()	10
4.1.2.9 parse_graph_type()	10
4.1.2.10 parse_main()	10
4.1.2.11 parse_operation_call()	10
4.1.2.12 parse_operations()	11
4.1.2.13 parse_program()	11
4.1.2.14 parse_subgraph()	11
4.1.2.15 syntax_error()	11
4.1.3 Variable Documentation	12
4.1.3.1 token_error_map	12
4.2 parser.h File Reference	12
4.2.1 Detailed Description	12
4.2.2 Function Documentation	12
4.2.2.1 parse_program()	12
4.3 scanner.c File Reference	13
4.3.1 Detailed Description	13
4.3.2 Function Documentation	13
4.3.2.1 isColor()	14
4.3.2.2 isKeyword()	15
4.3.2.3 isSpace()	15
4.3.2.4 isTag()	15
4.3.2.5 next_token()	16

4.3.3 Variable Documentation	16
4.3.3.1 token_map	16
4.4 scanner.h File Reference	16
4.4.1 Detailed Description	18
4.4.2 Function Documentation	18
4.4.2.1 isColor()	18
4.4.2.2 isKeyword()	18
4.4.2.3 isSpace()	18
4.4.2.4 isTag()	19
4.4.2.5 next_token()	19
Index	21

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

TokenData	Defined type based a struct holding various informations on a token	5
---------------------------	-------------------------------------------------------------------------------	-------------------

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

parser.c	Parser source file	7
parser.h	Parser header file	12
scanner.c	Scanner source file	13
scanner.h	Scanner header file	16

Chapter 3

Data Structure Documentation

3.1 TokenData Struct Reference

Defined type based a struct holding various informations on a token.

```
#include <scanner.h>
```

Data Fields

- char * **token**
- [TokenType](#) **type**
- int **start_ln**
- int **start_col**

3.1.1 Detailed Description

Defined type based a struct holding various informations on a token.

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

- [scanner.h](#)

Chapter 4

File Documentation

4.1 parser.c File Reference

Parser source file.

```
#include <stdio.h>
#include <stdlib.h>
#include "scanner.h"
```

Functions

- int [parse_subgraph](#) ()
Parses subgraphs declarations if next token matches subgraph.
- int [parse_declare](#) ()
Parses nodes & edges declarations.
- void [parse_main](#) ()
Parses a main block.
- void [parse_graph](#) ()
Parses a graph declaration.
- void [syntax_error](#) (const [TokenType](#) expected_token)
Prints the syntax error corresponding to the expected type with the error line and column mention.
- int [match](#) (const [TokenType](#) type_to_match)
Prints the syntax error corresponding to the expected type with the error line and column mention.
- int [is_instruction](#) ()
Checks if the current token is a valid instruction.
- int [is_operation_param](#) ()
Checks if the current token is a valid operation parameter.
- int [is_expression](#) ()
Checks if the current token is a valid condition expression.
- int [is_compare_op](#) ()
Checks if the current token is a valid comparison operator.
- void [parse_program](#) ()
Parses the next token and calls [parse_graph\(\)](#) or [parse_main\(\)](#) correspondingly.
- int [parse_graph_type](#) ()

- Parses the graph type (type) declaration and calls the `parse_subgraph()` function.*
- `int parse_operation_call ()`
Parses a single operation call, stopping at the closing parenthesis token.
- `int operations_routine ()`
Parses successive valid instructions (operation call, if clause or traverse clause).
- `int parse_operations ()`
Parses an operations block.

Variables

- `const char *const token_error_map []`
Constant char array for mapping the token type to the corresponding error name.*

4.1.1 Detailed Description

Parser source file.

4.1.2 Function Documentation

4.1.2.1 `is_compare_op()`

```
int is_compare_op ( )
```

Checks if the current token is a valid comparison operator.

Returns

1 if valid comparison operator, 0 if not.

4.1.2.2 `is_expression()`

```
int is_expression ( )
```

Checks if the current token is a valid condition expression.

Returns

1 if valid condition expression, 0 if not.

4.1.2.3 is_instruction()

```
int is_instruction ( )
```

Checks if the current token is a valid instruction.

Returns

1 if valid instruction, 0 if not.

4.1.2.4 is_operation_param()

```
int is_operation_param ( )
```

Checks if the current token is a valid operation parameter.

Returns

1 if valid operation parameter, 0 if not.

4.1.2.5 match()

```
int match (
    const TokenType type_to_match )
```

Prints the syntax error corresponding to the expected type with the error line and column mention.

Parameters

<i>expected_token</i>	The expected token type that was failed to match.
-----------------------	---------------------------------------------------

Returns

1 if the current token type matches the expected type, 0 if not.

4.1.2.6 operations_routine()

```
int operations_routine ( )
```

Parses successive valid instructions (operation call, if clause or traverse clause).

Returns

0 if a syntax error is found, 1 if not.

4.1.2.7 `parse_declare()`

```
int parse_declare ( )
```

Parses nodes & edges declarations.

Returns

0 if a syntax error is found, 1 if not.

4.1.2.8 `parse_graph()`

```
void parse_graph ( )
```

Parses a graph declaration.

Calls [parse_program\(\)](#) at the end.

4.1.2.9 `parse_graph_type()`

```
int parse_graph_type ( )
```

Parses the graph type (type) declaration and calls the [parse_subgraph\(\)](#) function.

Returns

0 if a syntax error is found, else returns [parse_subgraph\(\)](#) value.

4.1.2.10 `parse_main()`

```
void parse_main ( )
```

Parses a main block.

Calls [parse_operations\(\)](#) to parse the operations block.

4.1.2.11 `parse_operation_call()`

```
int parse_operation_call ( )
```

Parses a single operation call, stopping at the closing parenthesis token.

Recursively calls itself if one of the operation parameters is also an operation.

Returns

0 if a syntax error is found, 1 if not.

4.1.2.12 `parse_operations()`

```
int parse_operations ( )
```

Parses an operations block.

Returns

0 if a syntax error is found, 1 if not.

4.1.2.13 `parse_program()`

```
void parse_program ( )
```

Parses the next token and calls [parse_graph\(\)](#) or [parse_main\(\)](#) correspondingly.

If the parsed token is neither an identifier nor a main token, an error is printed and the parser halts.

4.1.2.14 `parse_subgraph()`

```
int parse_subgraph ( )
```

Parses subgraphs declarations if next token matches subgraph.

Calls [parse_declare\(\)](#) at the end.

Returns

0 if a syntax error is found, else returns [parse_declare\(\)](#) value.

4.1.2.15 `syntax_error()`

```
void syntax_error (
    const TokenType expected_token )
```

Prints the syntax error corresponding to the expected type with the error line and column mention.

Parameters

<i>expected_token</i>	The expected token type that was failed to match.
-----------------------	---------------------------------------------------

4.1.3 Variable Documentation

4.1.3.1 token_error_map

```
const char* const token_error_map[]
```

Initial value:

```
= {
    " ", "(", ")", "=", "<>", ">", "<", "<=", ">=", "{", "}", "main", "%type", "%declare", "%subgraph",
    "%operations",
    "directed or token undirected", "->", ",", ";", "color", "if", "traverse", "operation", "=>", "dfs or
    token bfs", ":", "EOF"
}
```

Constant char* array for mapping the token type to the corresponding error name.

4.2 parser.h File Reference

Parser header file.

Functions

- void [parse_program\(\)](#)
Parses the next token and calls [parse_graph\(\)](#) or [parse_main\(\)](#) correspondingly.

Variables

- const char *const **keywords** []

4.2.1 Detailed Description

Parser header file.

4.2.2 Function Documentation

4.2.2.1 parse_program()

```
void parse_program ( )
```

Parses the next token and calls [parse_graph\(\)](#) or [parse_main\(\)](#) correspondingly.

If the parsed token is neither an identifier nor a main token, an error is printed and the parser halts.

4.3 scanner.c File Reference

Scanner source file.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "scanner.h"
```

Functions

- void `next_token` ()
Decides the next token type and calls the appropriate function.
- void `readWord` ()
Reads the next word token in the file and stores its data in the `current_token` variable.
- int `isKeyword` (char *token)
Checks if the given token is a keyword or an identifier.
- void `readNum` ()
Reads the next number token in the file and stores its data in the `current_token` variable.
- void `readTag` ()
Reads the next tag token (%...) in the file and stores its data in the `current_token` variable.
- int `isTag` (char *token)
Checks if the given token is a valid tag token.
- void `readColor` ()
Reads the next color token in the file and stores its data in the `current_token` variable.
- int `isColor` (char *token)
Checks if the given token is a valid color token.
- int `isSpace` ()
Checks if the current character is a space and increments `CURRENT_COLUMN` and `CURRENT_ROW` accordingly.
- void `readSpecialChar` ()
Reads the next special character and stores the data in the `current_token` variable.
- void `generateError` ()
Prints the lexical error with the error line and column mention.

Variables

- const char *const `token_map` []
Constant char array for mapping the token type to its string.*

4.3.1 Detailed Description

Scanner source file.

4.3.2 Function Documentation

4.3.2.1 isColor()

```
int isColor (
    char * token )
```

Checks if the given token is a valid color token.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type if the token is a color tag, -1 if not.

4.3.2.2 isKeyword()

```
int isKeyword (
    char * token )
```

Checks if the given token is a keyword or an identifier.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type value.

4.3.2.3 isSpace()

```
int isSpace ( )
```

Checks if the current character is a space and increments CURRENT_COLUMN and CURRENT_ROW accordingly.

Returns

1 if the current character is a space character, 0 if not.

4.3.2.4 isTag()

```
int isTag (
    char * token )
```

Checks if the given token is a valid tag token.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type if the token is a valid tag, -1 if not.

4.3.2.5 next_token()

```
void next_token ( )
```

Decides the next token type and calls the appropriate function.

Keeps count of the character in the current line.

4.3.3 Variable Documentation**4.3.3.1 token_map**

```
const char* const token_map[]
```

Initial value:

```
= {
    "ID_TOKEN", "NUM_TOKEN", "OP_TOKEN", "CP_TOKEN", "EQ_TOKEN", "NEQ_TOKEN", "GT_TOKEN", "LT_TOKEN",
    "LEQ_TOKEN", "BEQ_TOKEN",
    "OB_TOKEN", "CB_TOKEN", "MAIN_TOKEN", "PTYPE_TOKEN", "PDECLARE_TOKEN", "PSUBGRAPH_TOKEN",
    "POPERATIONS_TOKEN", "GTYPE_TOKEN",
    "EDGE_TOKEN", "COMMA_TOKEN", "SEMICOLON_TOKEN", "COLOR_TOKEN", "IF_TOKEN", "LOOP_TOKEN",
    "OPERATION_TOKEN", "ARROW_TOKEN",
    "GSEARCH_TOKEN", "COLON_TOKEN", "EOF_TOKEN"
}
```

Constant char* array for mapping the token type to its string.

4.4 scanner.h File Reference

Scanner header file.

Data Structures

- struct [TokenData](#)

Defined type based a struct holding various informations on a token.

Macros

- `#define TOKEN_COUNT 29`

Enumerations

- enum `TokenType` {
`ID_TOKEN` , `NUM_TOKEN` , `OP_TOKEN` , `CP_TOKEN` ,
`EQ_TOKEN` , `NEQ_TOKEN` , `GT_TOKEN` , `LT_TOKEN` ,
`LEQ_TOKEN` , `BEQ_TOKEN` , `OB_TOKEN` , `CB_TOKEN` ,
`MAIN_TOKEN` , `PTYPE_TOKEN` , `PDECLARE_TOKEN` , `PSUBGRAPH_TOKEN` ,
`POPERATIONS_TOKEN` , `GTYPE_TOKEN` , `EDGE_TOKEN` , `COMMA_TOKEN` ,
`SEMICOLON_TOKEN` , `COLOR_TOKEN` , `IF_TOKEN` , `LOOP_TOKEN` ,
`OPERATION_TOKEN` , `ARROW_TOKEN` , `GSEARCH_TOKEN` , `COLON_TOKEN` ,
`EOF_TOKEN` }

Enumeration of the different tokens that constitute the GraphEx grammar.

Functions

- void `next_token` ()
Keeps count of the character in the current line.
- void `readWord` ()
Reads the next word token in the file and stores its data in the `current_token` variable.
- int `isKeyword` (char *)
Checks if the given token is a keyword or an identifier.
- void `readNum` ()
Reads the next number token in the file and stores its data in the `current_token` variable.
- void `readTag` ()
Reads the next tag token (%...) in the file and stores its data in the `current_token` variable.
- int `isTag` (char *)
Checks if the given token is a valid tag token.
- void `readColor` ()
Reads the next color token in the file and stores its data in the `current_token` variable.
- int `isColor` (char *)
Checks if the given token is a valid color token.
- int `isSpace` ()
Checks if the current character is a space and increments `CURRENT_COLUMN` and `CURRENT_ROW` accordingly.
- void `readSpecialChar` ()
Reads the next special character and stores the data in the `current_token` variable.
- void `generateError` ()
Prints the lexical error with the error line and column mention.

Variables

- `TokenData` * `current_token`
- FILE * `PROGRAM_File`
Pointer on the current token.
- char `CURRENT_CHAR`
Pointer on the current file.
- int `CURRENT_ROW`
Current character in the buffer.
- int `CURRENT_COLUMN`
Keeps count of the current line in the file.

4.4.1 Detailed Description

Scanner header file.

4.4.2 Function Documentation

4.4.2.1 isColor()

```
int isColor (
    char * token )
```

Checks if the given token is a valid color token.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type if the token is a color tag, -1 if not.

4.4.2.2 isKeyword()

```
int isKeyword (
    char * token )
```

Checks if the given token is a keyword or an identifier.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type value.

4.4.2.3 isSpace()

```
int isSpace ( )
```

Checks if the current character is a space and increments CURRENT_COLUMN and CURRENT_ROW accordingly.

Returns

1 if the current character is a space character, 0 if not.

4.4.2.4 isTag()

```
int isTag (
    char * token )
```

Checks if the given token is a valid tag token.

Parameters

<i>token</i>	The token that has been read.
--------------	-------------------------------

Returns

The corresponding token type if the token is a valid tag, -1 if not.

4.4.2.5 next_token()

```
void next_token ( )
```

Keeps count of the character in the current line.

Keeps count of the character in the current line.

Index

- is_compare_op
 - parser.c, [8](#)
- is_expression
 - parser.c, [8](#)
- is_instruction
 - parser.c, [8](#)
- is_operation_param
 - parser.c, [9](#)
- isColor
 - scanner.c, [13](#)
 - scanner.h, [18](#)
- isKeyword
 - scanner.c, [15](#)
 - scanner.h, [18](#)
- isSpace
 - scanner.c, [15](#)
 - scanner.h, [18](#)
- isTag
 - scanner.c, [15](#)
 - scanner.h, [19](#)
- match
 - parser.c, [9](#)
- next_token
 - scanner.c, [16](#)
 - scanner.h, [19](#)
- operations_routine
 - parser.c, [9](#)
- parse_declare
 - parser.c, [9](#)
- parse_graph
 - parser.c, [10](#)
- parse_graph_type
 - parser.c, [10](#)
- parse_main
 - parser.c, [10](#)
- parse_operation_call
 - parser.c, [10](#)
- parse_operations
 - parser.c, [10](#)
- parse_program
 - parser.c, [11](#)
 - parser.h, [12](#)
- parse_subgraph
 - parser.c, [11](#)
- parser.c, [7](#)
 - is_compare_op, [8](#)
- is_expression, [8](#)
- is_instruction, [8](#)
- is_operation_param, [9](#)
- match, [9](#)
- operations_routine, [9](#)
- parse_declare, [9](#)
- parse_graph, [10](#)
- parse_graph_type, [10](#)
- parse_main, [10](#)
- parse_operation_call, [10](#)
- parse_operations, [10](#)
- parse_program, [11](#)
- parse_subgraph, [11](#)
- syntax_error, [11](#)
- token_error_map, [12](#)
- parser.h, [12](#)
 - parse_program, [12](#)
- scanner.c, [13](#)
 - isColor, [13](#)
 - isKeyword, [15](#)
 - isSpace, [15](#)
 - isTag, [15](#)
 - next_token, [16](#)
 - token_map, [16](#)
- scanner.h, [16](#)
 - isColor, [18](#)
 - isKeyword, [18](#)
 - isSpace, [18](#)
 - isTag, [19](#)
 - next_token, [19](#)
- syntax_error
 - parser.c, [11](#)
- token_error_map
 - parser.c, [12](#)
- token_map
 - scanner.c, [16](#)
- TokenData, [5](#)