

Ramiel

0.1

Generated by Doxygen 1.8.14

Contents

1	Ramiel	1
2	Data Structure Index	3
2.1	Data Structures	3
3	File Index	5
3.1	File List	5
4	Data Structure Documentation	7
4.1	hashtable Struct Reference	7
4.1.1	Detailed Description	7
4.1.2	Field Documentation	7
4.1.2.1	count	7
4.1.2.2	items	7
4.1.2.3	size	8
4.2	ht_item Struct Reference	8
4.2.1	Detailed Description	8
4.2.2	Field Documentation	8
4.2.2.1	identifier	8
4.2.2.2	next	8
4.2.2.3	size	9
4.2.2.4	type	9
4.3	stack Struct Reference	9
4.3.1	Detailed Description	9
4.3.2	Field Documentation	9

4.3.2.1	count	9
4.3.2.2	top	10
4.4	stack_item Struct Reference	10
4.4.1	Detailed Description	10
4.4.2	Field Documentation	10
4.4.2.1	next	10
4.4.2.2	size	10
4.4.2.3	type	11
4.5	state Struct Reference	11
4.5.1	Detailed Description	11
4.5.2	Field Documentation	11
4.5.2.1	column	11
4.5.2.2	current	11
4.5.2.3	errors	12
4.5.2.4	input	12
4.5.2.5	line	12
4.5.2.6	list	12
4.5.2.7	output	12
4.5.2.8	stack	12
4.5.2.9	variables	13
4.5.2.10	warnings	13
4.6	token Struct Reference	13
4.6.1	Detailed Description	13
4.6.2	Field Documentation	13
4.6.2.1	content	13
4.6.2.2	type	14
4.7	token_info Struct Reference	14
4.7.1	Detailed Description	14
4.7.2	Field Documentation	14
4.7.2.1	"@1	14
4.7.2.2	column	14
4.7.2.3	line	15
4.7.2.4	negative	15
4.7.2.5	next	15
4.7.2.6	tok	15
4.7.2.7	val_c	15
4.7.2.8	val_d	15
4.7.2.9	val_i	15
4.7.2.10	val_s	15

5 File Documentation	17
5.1 console.c File Reference	17
5.1.1 Macro Definition Documentation	17
5.1.1.1 C_B	17
5.1.1.2 C_G	18
5.1.1.3 C_R	18
5.1.1.4 C_RST	18
5.1.1.5 C_Y	18
5.1.2 Function Documentation	18
5.1.2.1 error_m()	18
5.1.2.2 standard_m()	19
5.1.2.3 warning_m()	19
5.2 hashtable.c File Reference	19
5.2.1 Function Documentation	19
5.2.1.1 hash()	19
5.2.1.2 hash_item()	20
5.2.1.3 lookup_item()	20
5.2.1.4 new_ht()	20
5.2.1.5 new_ht_item()	20
5.2.1.6 rm_ht()	20
5.2.1.7 rm_ht_helper()	21
5.2.1.8 rm_ht_item()	21
5.3 hashtable.h File Reference	21
5.3.1 Enumeration Type Documentation	21
5.3.1.1 data_t	21
5.3.2 Function Documentation	22
5.3.2.1 hash()	22
5.3.2.2 hash_item()	22
5.3.2.3 lookup_item()	22
5.3.2.4 new_ht()	22

5.3.2.5	new_ht_item()	23
5.3.2.6	rm_ht()	23
5.3.2.7	rm_ht_helper()	23
5.3.2.8	rm_ht_item()	23
5.3.3	Variable Documentation	23
5.3.3.1	data_t_str	23
5.4	lexer.c File Reference	24
5.4.1	Function Documentation	24
5.4.1.1	make_tokens()	24
5.4.1.2	next_char()	24
5.4.1.3	parse()	24
5.4.1.4	parse_char()	25
5.4.1.5	parse_number()	25
5.4.1.6	parse_string()	25
5.4.1.7	parse_symbol_or_operand()	25
5.4.1.8	parse_word()	25
5.4.1.9	parse_word_or_number()	26
5.4.1.10	return_char()	26
5.4.1.11	skip_line()	26
5.5	lexer.h File Reference	26
5.5.1	Function Documentation	27
5.5.1.1	make_tokens()	27
5.5.1.2	next_char()	27
5.5.1.3	parse()	27
5.5.1.4	parse_char()	27
5.5.1.5	parse_number()	27
5.5.1.6	parse_string()	28
5.5.1.7	parse_symbol_or_operand()	28
5.5.1.8	parse_word()	28
5.5.1.9	parse_word_or_number()	28

5.5.1.10	return_char()	28
5.5.1.11	skip_line()	29
5.6	parser.c File Reference	29
5.6.1	Function Documentation	29
5.6.1.1	analyze()	29
5.6.1.2	consume_body_token()	30
5.6.1.3	consume_token()	30
5.6.1.4	expect()	30
5.6.1.5	expect_asgn()	30
5.6.1.6	expect_body()	30
5.6.1.7	expect_data_type()	30
5.6.1.8	expect_decl()	31
5.6.1.9	expect_eoi()	31
5.6.1.10	expect_for()	31
5.6.1.11	expect_iden()	31
5.6.1.12	expect_if()	31
5.6.1.13	expect_lit()	32
5.6.1.14	expect_lit_or_iden()	32
5.6.1.15	expect_operation()	32
5.6.1.16	expect_operator()	32
5.6.1.17	expect_print()	32
5.6.1.18	expect_read()	32
5.6.1.19	expect_while()	33
5.6.1.20	rewind_token()	33
5.6.1.21	skip_body_tokens()	33
5.6.1.22	skip_tokens()	33
5.7	parser.h File Reference	33
5.7.1	Function Documentation	34
5.7.1.1	analyze()	34
5.7.1.2	consume_body_token()	34

5.7.1.3	<code>consume_token()</code>	35
5.7.1.4	<code>expect()</code>	35
5.7.1.5	<code>expect_asgn()</code>	35
5.7.1.6	<code>expect_body()</code>	35
5.7.1.7	<code>expect_data_type()</code>	35
5.7.1.8	<code>expect_decl()</code>	35
5.7.1.9	<code>expect_eoi()</code>	36
5.7.1.10	<code>expect_for()</code>	36
5.7.1.11	<code>expect_iden()</code>	36
5.7.1.12	<code>expect_if()</code>	36
5.7.1.13	<code>expect_lit()</code>	36
5.7.1.14	<code>expect_lit_or_iden()</code>	37
5.7.1.15	<code>expect_operation()</code>	37
5.7.1.16	<code>expect_operator()</code>	37
5.7.1.17	<code>expect_print()</code>	37
5.7.1.18	<code>expect_read()</code>	37
5.7.1.19	<code>expect_while()</code>	37
5.7.1.20	<code>rewind_token()</code>	38
5.7.1.21	<code>skip_body_tokens()</code>	38
5.7.1.22	<code>skip_tokens()</code>	38
5.7.2	Variable Documentation	38
5.7.2.1	<code>iden</code>	38
5.7.2.2	<code>ope</code>	38
5.8	<code>ramiel.c</code> File Reference	38
5.8.1	Function Documentation	39
5.8.1.1	<code>main()</code>	39
5.9	<code>README.md</code> File Reference	39
5.10	<code>stack.c</code> File Reference	39
5.10.1	Function Documentation	39
5.10.1.1	<code>new_stack()</code>	39

5.10.1.2	new_stack_item()	40
5.10.1.3	pop()	40
5.10.1.4	push()	40
5.10.1.5	stack_operation()	40
5.11	stack.h File Reference	40
5.11.1	Function Documentation	41
5.11.1.1	new_stack()	41
5.11.1.2	new_stack_item()	41
5.11.1.3	pop()	41
5.11.1.4	push()	41
5.11.1.5	stack_operation()	41
5.12	state.c File Reference	42
5.12.1	Function Documentation	42
5.12.1.1	append_token()	42
5.12.2	Variable Documentation	42
5.12.2.1	lex_state	42
5.13	tokens.c File Reference	42
5.13.1	Macro Definition Documentation	43
5.13.1.1	D_TYPES	44
5.13.1.2	L_OPERATORS	44
5.13.1.3	LITERALS	44
5.13.1.4	OPERATORS	44
5.13.1.5	R_OPERATORS	44
5.13.2	Enumeration Type Documentation	44
5.13.2.1	token_type	44
5.13.3	Function Documentation	46
5.13.3.1	new_token_info()	46
5.13.4	Variable Documentation	46
5.13.4.1	d_types	46
5.13.4.2	d_types_code	46

5.13.4.3	d_types_mnemonic	46
5.13.4.4	l_operator_code	47
5.13.4.5	l_operators	47
5.13.4.6	literals	47
5.13.4.7	operator_code	47
5.13.4.8	operators	47
5.13.4.9	r_operator_code	48
5.13.4.10	r_operators	48
5.13.4.11	token_names	48
5.14	utils.c File Reference	48
5.14.1	Macro Definition Documentation	48
5.14.1.1	arr_len	49
5.14.1.2	max	49
5.14.2	Function Documentation	49
5.14.2.1	append()	49
Index		51

Chapter 1

Ramiel

Compiler for the AUTO programming language

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

hashtable	7
ht_item	8
stack	9
stack_item	10
state	11
token	13
token_info	14

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

console.c	17
hashtable.c	19
hashtable.h	21
lexer.c	24
lexer.h	26
parser.c	29
parser.h	33
ramiel.c	38
stack.c	39
stack.h	40
state.c	42
tokens.c	42
utils.c	48

Chapter 4

Data Structure Documentation

4.1 hashtable Struct Reference

```
#include <hashtable.h>
```

Data Fields

- `size_t` [size](#)
- `char` [count](#)
- `struct ht_item` [** items](#)

4.1.1 Detailed Description

Definition at line 15 of file hashtable.h.

4.1.2 Field Documentation

4.1.2.1 count

```
char count
```

Definition at line 17 of file hashtable.h.

4.1.2.2 items

```
struct ht_item** items
```

Definition at line 18 of file hashtable.h.

4.1.2.3 size

`size_t size`

Definition at line 16 of file hashtable.h.

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

- [hashtable.h](#)

4.2 ht_item Struct Reference

```
#include <hashtable.h>
```

Data Fields

- `char *` [identifier](#)
- `size_t` [size](#)
- `enum` [data_t](#) `type`
- `void *` [next](#)

4.2.1 Detailed Description

Definition at line 8 of file hashtable.h.

4.2.2 Field Documentation

4.2.2.1 identifier

`char* identifier`

Definition at line 9 of file hashtable.h.

4.2.2.2 next

`void* next`

Definition at line 12 of file hashtable.h.

4.2.2.3 size

```
size_t size
```

Definition at line 10 of file hashtable.h.

4.2.2.4 type

```
enum data_t type
```

Definition at line 11 of file hashtable.h.

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

- [hashtable.h](#)

4.3 stack Struct Reference

```
#include <stack.h>
```

Data Fields

- struct [stack_item](#) * [top](#)
- size_t [count](#)

4.3.1 Detailed Description

Definition at line 7 of file stack.h.

4.3.2 Field Documentation

4.3.2.1 count

```
size_t count
```

Definition at line 9 of file stack.h.

4.3.2.2 top

```
struct stack\_item* top
```

Definition at line 8 of file stack.h.

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

- [stack.h](#)

4.4 stack_item Struct Reference

```
#include <stack.h>
```

Data Fields

- enum [data_t](#) type
- size_t [size](#)
- struct [stack_item](#) * [next](#)

4.4.1 Detailed Description

Definition at line 1 of file stack.h.

4.4.2 Field Documentation

4.4.2.1 next

```
struct stack\_item* next
```

Definition at line 4 of file stack.h.

4.4.2.2 size

```
size_t size
```

Definition at line 3 of file stack.h.

4.4.2.3 type

```
enum data_t type
```

Definition at line 2 of file stack.h.

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

- [stack.h](#)

4.5 state Struct Reference

Data Fields

- FILE * [input](#)
- FILE * [output](#)
- struct [token_info](#) * [current](#)
- struct [token_info](#) * [list](#)
- struct [hashtable](#) * [variables](#)
- struct [stack](#) * [stack](#)
- unsigned short [errors](#)
- unsigned short [warnings](#)
- unsigned short [line](#)
- unsigned short [column](#)

4.5.1 Detailed Description

Definition at line 1 of file state.c.

4.5.2 Field Documentation

4.5.2.1 column

```
unsigned short column
```

Definition at line 11 of file state.c.

4.5.2.2 current

```
struct token_info* current
```

Definition at line 4 of file state.c.

4.5.2.3 errors

`unsigned short errors`

Definition at line 8 of file state.c.

4.5.2.4 input

`FILE* input`

Definition at line 2 of file state.c.

4.5.2.5 line

`unsigned short line`

Definition at line 10 of file state.c.

4.5.2.6 list

`struct token_info* list`

Definition at line 5 of file state.c.

4.5.2.7 output

`FILE* output`

Definition at line 3 of file state.c.

4.5.2.8 stack

`struct stack* stack`

Definition at line 7 of file state.c.

4.5.2.9 variables

```
struct hashtable* variables
```

Definition at line 6 of file state.c.

4.5.2.10 warnings

```
unsigned short warnings
```

Definition at line 9 of file state.c.

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

- [state.c](#)

4.6 token Struct Reference

Data Fields

- enum [token_type](#) type
- char * [content](#)

4.6.1 Detailed Description

Definition at line 40 of file tokens.c.

4.6.2 Field Documentation

4.6.2.1 content

```
char* content
```

Definition at line 42 of file tokens.c.

4.6.2.2 type

```
enum token_type type
```

Definition at line 41 of file tokens.c.

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

- [tokens.c](#)

4.7 token_info Struct Reference

Data Fields

- struct [token](#) * [tok](#)
- struct [token_info](#) * [next](#)
- unsigned short [line](#)
- unsigned short [column](#)
- union {
 - int [val_i](#)
 - double [val_d](#)
 - char [val_c](#)
 - char * [val_s](#)
- };
- unsigned char [negative](#)

4.7.1 Detailed Description

Definition at line 45 of file tokens.c.

4.7.2 Field Documentation

4.7.2.1 "@1

```
union { ... }
```

4.7.2.2 column

```
unsigned short column
```

Definition at line 48 of file tokens.c.

4.7.2.3 line

unsigned short line

Definition at line 48 of file tokens.c.

4.7.2.4 negative

unsigned char negative

Definition at line 55 of file tokens.c.

4.7.2.5 next

struct [token_info](#)* next

Definition at line 47 of file tokens.c.

4.7.2.6 tok

struct [token](#)* tok

Definition at line 46 of file tokens.c.

4.7.2.7 val_c

char val_c

Definition at line 52 of file tokens.c.

4.7.2.8 val_d

double val_d

Definition at line 51 of file tokens.c.

4.7.2.9 val_i

int val_i

Definition at line 50 of file tokens.c.

4.7.2.10 val_s

char* val_s

Definition at line 53 of file tokens.c.

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

- [tokens.c](#)

Chapter 5

File Documentation

5.1 console.c File Reference

```
#include <stdio.h>
```

Macros

- `#define C_R "\e[1;31m"`
- `#define C_G "\e[1;32m"`
- `#define C_B "\e[1;34m"`
- `#define C_Y "\e[1;33m"`
- `#define C_RST "\e[0m"`

Functions

- void `error_m` (char *message, ushort line, ushort column)
- void `warning_m` (char *message, ushort line, ushort column)
- void `standard_m` (char *message)

5.1.1 Macro Definition Documentation

5.1.1.1 C_B

```
#define C_B "\e[1;34m"
```

Definition at line 3 of file console.c.

5.1.1.2 C_G

```
#define C_G "\e[1;32m"
```

Definition at line 2 of file console.c.

5.1.1.3 C_R

```
#define C_R "\e[1;31m"
```

Definition at line 1 of file console.c.

5.1.1.4 C_RST

```
#define C_RST "\e[0m"
```

Definition at line 5 of file console.c.

5.1.1.5 C_Y

```
#define C_Y "\e[1;33m"
```

Definition at line 4 of file console.c.

5.1.2 Function Documentation

5.1.2.1 error_m()

```
void error_m (  
    char * message,  
    ushort line,  
    ushort column )
```

Definition at line 9 of file console.c.

5.1.2.2 standard_m()

```
void standard_m (  
    char * message )
```

Definition at line 21 of file console.c.

5.1.2.3 warning_m()

```
void warning_m (  
    char * message,  
    ushort line,  
    ushort column )
```

Definition at line 15 of file console.c.

5.2 hashtable.c File Reference

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include "hashtable.h"
```

Functions

- struct [hashtable](#) * [new_ht](#) (size_t s)
- void [rm_ht](#) (struct [hashtable](#) *ht)
- void [rm_ht_helper](#) (struct [ht_item](#) *item)
- struct [ht_item](#) * [new_ht_item](#) (const char *identifier, size_t size, enum [data_t](#) type)
- char [rm_ht_item](#) (struct [hashtable](#) *ht, char *str)
- unsigned long [hash](#) (const char *str)
- void [hash_item](#) (struct [hashtable](#) *ht, struct [ht_item](#) *item)
- struct [ht_item](#) * [lookup_item](#) (struct [hashtable](#) *ht, char *identifier)

5.2.1 Function Documentation

5.2.1.1 hash()

```
unsigned long hash (  
    const char * str )
```

Definition at line 73 of file hashtable.c.

5.2.1.2 hash_item()

```
void hash_item (
    struct hashtable * ht,
    struct ht_item * item )
```

Definition at line 83 of file hashtable.c.

5.2.1.3 lookup_item()

```
struct ht_item* lookup_item (
    struct hashtable * ht,
    char * identifier )
```

Definition at line 99 of file hashtable.c.

5.2.1.4 new_ht()

```
struct hashtable* new_ht (
    size_t s )
```

Definition at line 6 of file hashtable.c.

5.2.1.5 new_ht_item()

```
struct ht_item* new_ht_item (
    const char * identifier,
    size_t size,
    enum data_t type )
```

Definition at line 34 of file hashtable.c.

5.2.1.6 rm_ht()

```
void rm_ht (
    struct hashtable * ht )
```

Definition at line 15 of file hashtable.c.

5.2.1.7 rm_ht_helper()

```
void rm_ht_helper (
    struct ht_item * item )
```

Definition at line 27 of file hashtable.c.

5.2.1.8 rm_ht_item()

```
char rm_ht_item (
    struct hashtable * ht,
    char * str )
```

Definition at line 44 of file hashtable.c.

5.3 hashtable.h File Reference

Data Structures

- struct [ht_item](#)
- struct [hashtable](#)

Enumerations

- enum [data_t](#) { [DATA_CHR](#), [DATA_INT](#), [DATA_FLT](#), [DATA_STR](#) }

Functions

- struct [hashtable](#) * [new_ht](#) (size_t s)
- void [rm_ht](#) (struct [hashtable](#) *ht)
- void [rm_ht_helper](#) (struct [ht_item](#) *item)
- struct [ht_item](#) * [new_ht_item](#) (const char *identifier, size_t size, enum [data_t](#) type)
- char [rm_ht_item](#) (struct [hashtable](#) *ht, char *str)
- unsigned long [hash](#) (const char *str)
- void [hash_item](#) (struct [hashtable](#) *ht, struct [ht_item](#) *item)
- struct [ht_item](#) * [lookup_item](#) (struct [hashtable](#) *ht, char *label)

Variables

- char * [data_t_str](#) = "DATA_CHRDATA_INTDATA_FLTDATA_STR"

5.3.1 Enumeration Type Documentation

5.3.1.1 data_t

```
enum data_t
```

Enumerator

DATA_CHR	
DATA_INT	
DATA_FLT	
DATA_STR	

Definition at line 1 of file hashtable.h.

5.3.2 Function Documentation

5.3.2.1 hash()

```
unsigned long hash (  
    const char * str )
```

Definition at line 73 of file hashtable.c.

5.3.2.2 hash_item()

```
void hash_item (  
    struct hashtable * ht,  
    struct ht_item * item )
```

Definition at line 83 of file hashtable.c.

5.3.2.3 lookup_item()

```
struct ht_item* lookup_item (  
    struct hashtable * ht,  
    char * label )
```

Definition at line 99 of file hashtable.c.

5.3.2.4 new_ht()

```
struct hashtable* new_ht (  
    size_t s )
```

Definition at line 6 of file hashtable.c.

5.3.2.5 new_ht_item()

```
struct ht_item* new_ht_item (
    const char * identifier,
    size_t size,
    enum data_t type )
```

Definition at line 34 of file hashtable.c.

5.3.2.6 rm_ht()

```
void rm_ht (
    struct hashtable * ht )
```

Definition at line 15 of file hashtable.c.

5.3.2.7 rm_ht_helper()

```
void rm_ht_helper (
    struct ht_item * item )
```

Definition at line 27 of file hashtable.c.

5.3.2.8 rm_ht_item()

```
char rm_ht_item (
    struct hashtable * ht,
    char * str )
```

Definition at line 44 of file hashtable.c.

5.3.3 Variable Documentation

5.3.3.1 data_t_str

```
char* data_t_str = "DATA_CHRDATA_INTDATA_FLTDATA_STR"
```

Definition at line 6 of file hashtable.h.

5.4 lexer.c File Reference

```
#include "lexer.h"
```

Functions

- char [make_tokens](#) ()
- char [next_char](#) ()
- void [return_char](#) (char ret)
- void [skip_line](#) ()
- char [parse](#) (struct [token_info](#) *n_info, char current)
- char [parse_char](#) (struct [token_info](#) *n_info)
- char [parse_string](#) (struct [token_info](#) *n_info)
- char [parse_word_or_number](#) (struct [token_info](#) *n_info, char first)
- char [parse_word](#) (struct [token_info](#) *n_info, char first)
- char [parse_number](#) (struct [token_info](#) *n_info, char first)
- char [parse_symbol_or_operand](#) (struct [token_info](#) *n_info, char *symbol)

5.4.1 Function Documentation

5.4.1.1 [make_tokens\(\)](#)

```
char make_tokens ( )
```

Definition at line 3 of file `lexer.c`.

5.4.1.2 [next_char\(\)](#)

```
char next_char ( )
```

Definition at line 39 of file `lexer.c`.

5.4.1.3 [parse\(\)](#)

```
char parse (  
    struct token\_info * n_info,  
    char current )
```

Definition at line 64 of file `lexer.c`.

5.4.1.4 parse_char()

```
char parse_char (
    struct token_info * n_info )
```

Definition at line 190 of file lexer.c.

5.4.1.5 parse_number()

```
char parse_number (
    struct token_info * n_info,
    char first )
```

Definition at line 290 of file lexer.c.

5.4.1.6 parse_string()

```
char parse_string (
    struct token_info * n_info )
```

Definition at line 211 of file lexer.c.

5.4.1.7 parse_symbol_or_operand()

```
char parse_symbol_or_operand (
    struct token_info * n_info,
    char * symbol )
```

Definition at line 326 of file lexer.c.

5.4.1.8 parse_word()

```
char parse_word (
    struct token_info * n_info,
    char first )
```

Definition at line 243 of file lexer.c.

5.4.1.9 parse_word_or_number()

```
char parse_word_or_number (
    struct token_info * n_info,
    char first )
```

Definition at line 234 of file lexer.c.

5.4.1.10 return_char()

```
void return_char (
    char ret )
```

Definition at line 54 of file lexer.c.

5.4.1.11 skip_line()

```
void skip_line ( )
```

Definition at line 59 of file lexer.c.

5.5 lexer.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "utils.c"
#include "tokens.c"
#include "state.c"
#include "console.c"
```

Functions

- char [make_tokens](#) ()
- char [next_char](#) ()
- void [return_char](#) (char ret)
- void [skip_line](#) ()
- char [parse](#) (struct [token_info](#) *n_info, char current)
- char [parse_char](#) (struct [token_info](#) *n_info)
- char [parse_string](#) (struct [token_info](#) *n_info)
- char [parse_word_or_number](#) (struct [token_info](#) *n_info, char first)
- char [parse_word](#) (struct [token_info](#) *n_info, char first)
- char [parse_symbol_or_operand](#) (struct [token_info](#) *n_info, char *symbol)
- char [parse_number](#) (struct [token_info](#) *n_info, char first)

5.5.1 Function Documentation

5.5.1.1 make_tokens()

```
char make_tokens ( )
```

Definition at line 3 of file lexer.c.

5.5.1.2 next_char()

```
char next_char ( )
```

Definition at line 39 of file lexer.c.

5.5.1.3 parse()

```
char parse (
    struct token_info * n_info,
    char current )
```

Definition at line 64 of file lexer.c.

5.5.1.4 parse_char()

```
char parse_char (
    struct token_info * n_info )
```

Definition at line 190 of file lexer.c.

5.5.1.5 parse_number()

```
char parse_number (
    struct token_info * n_info,
    char first )
```

Definition at line 290 of file lexer.c.

5.5.1.6 parse_string()

```
char parse_string (
    struct token_info * n_info )
```

Definition at line 211 of file lexer.c.

5.5.1.7 parse_symbol_or_operand()

```
char parse_symbol_or_operand (
    struct token_info * n_info,
    char * symbol )
```

Definition at line 326 of file lexer.c.

5.5.1.8 parse_word()

```
char parse_word (
    struct token_info * n_info,
    char first )
```

Definition at line 243 of file lexer.c.

5.5.1.9 parse_word_or_number()

```
char parse_word_or_number (
    struct token_info * n_info,
    char first )
```

Definition at line 234 of file lexer.c.

5.5.1.10 return_char()

```
void return_char (
    char ret )
```

Definition at line 54 of file lexer.c.

5.5.1.11 skip_line()

```
void skip_line ( )
```

Definition at line 59 of file lexer.c.

5.6 parser.c File Reference

```
#include "parser.h"
```

Functions

- char [analyze](#) ()
- void [rewind_token](#) ()
- void [skip_tokens](#) ()
- char [consume_token](#) ()
- char [expect](#) (enum [token_type](#) type, char required)
- char [expect_data_type](#) (char required)
- char [expect_lit_or_iden](#) (char [stack](#))
- char [expect_lit](#) ()
- char [expect_iden](#) (char left, char right)
- char [expect_operator](#) (char required)
- char [expect_operation](#) ()
- char [expect_decl](#) (char multiple, enum [token_type](#) type)
- char [expect_asgn](#) (struct [ht_item](#) *item)
- char [expect_if](#) (unsigned short line, unsigned short column)
- char [expect_while](#) (unsigned short line, unsigned short column)
- char [expect_for](#) (unsigned short line, unsigned short column)
- char [expect_print](#) ()
- char [expect_read](#) ()
- char [expect_body](#) ()
- void [skip_body_tokens](#) ()
- char [consume_body_token](#) ()
- char [expect_eoi](#) ()

5.6.1 Function Documentation

5.6.1.1 analyze()

```
char analyze ( )
```

Definition at line 3 of file parser.c.

5.6.1.2 consume_body_token()

```
char consume_body_token ( )
```

Definition at line 460 of file parser.c.

5.6.1.3 consume_token()

```
char consume_token ( )
```

Definition at line 33 of file parser.c.

5.6.1.4 expect()

```
char expect (
    enum token_type type,
    char required )
```

Definition at line 89 of file parser.c.

5.6.1.5 expect_asgn()

```
char expect_asgn (
    struct ht_item * item )
```

Definition at line 316 of file parser.c.

5.6.1.6 expect_body()

```
char expect_body ( )
```

Definition at line 432 of file parser.c.

5.6.1.7 expect_data_type()

```
char expect_data_type (
    char required )
```

Definition at line 107 of file parser.c.

5.6.1.8 expect_decl()

```
char expect_decl (
    char multiple,
    enum token_type type )
```

Definition at line 259 of file parser.c.

5.6.1.9 expect_eoi()

```
char expect_eoi ( )
```

Definition at line 518 of file parser.c.

5.6.1.10 expect_for()

```
char expect_for (
    unsigned short line,
    unsigned short column )
```

Definition at line 353 of file parser.c.

5.6.1.11 expect_iden()

```
char expect_iden (
    char left,
    char right )
```

Definition at line 167 of file parser.c.

5.6.1.12 expect_if()

```
char expect_if (
    unsigned short line,
    unsigned short column )
```

Definition at line 345 of file parser.c.

5.6.1.13 expect_lit()

```
char expect_lit ( )
```

Definition at line 153 of file parser.c.

5.6.1.14 expect_lit_or_iden()

```
char expect_lit_or_iden (
    char stack )
```

Definition at line 120 of file parser.c.

5.6.1.15 expect_operation()

```
char expect_operation ( )
```

Definition at line 235 of file parser.c.

5.6.1.16 expect_operator()

```
char expect_operator (
    char required )
```

Definition at line 222 of file parser.c.

5.6.1.17 expect_print()

```
char expect_print ( )
```

Definition at line 424 of file parser.c.

5.6.1.18 expect_read()

```
char expect_read ( )
```

Definition at line 428 of file parser.c.

5.6.1.19 expect_while()

```
char expect_while (
    unsigned short line,
    unsigned short column )
```

Definition at line 349 of file parser.c.

5.6.1.20 rewind_token()

```
void rewind_token ( )
```

Definition at line 14 of file parser.c.

5.6.1.21 skip_body_tokens()

```
void skip_body_tokens ( )
```

Definition at line 448 of file parser.c.

5.6.1.22 skip_tokens()

```
void skip_tokens ( )
```

Definition at line 23 of file parser.c.

5.7 parser.h File Reference

```
#include "hashtable.c"
#include "stack.c"
```

Functions

- char [analyze](#) ()
- void [rewind_token](#) ()
- void [skip_tokens](#) ()
- char [consume_token](#) ()
- char [expect](#) (enum [token_type](#) type, char required)
- char [expect_data_type](#) (char required)
- char [expect_lit_or_iden](#) (char [stack](#))
- char [expect_lit](#) ()
- char [expect_iden](#) (char left, char right)
- char [expect_operator](#) (char required)
- char [expect_operation](#) ()
- char [expect_decl](#) (char multiple, enum [token_type](#) type)
- char [expect_asgn](#) (struct [ht_item](#) *item)
- char [expect_if](#) (unsigned short line, unsigned short column)
- char [expect_while](#) (unsigned short line, unsigned short column)
- char [expect_for](#) (unsigned short line, unsigned short column)
- char [expect_print](#) ()
- char [expect_read](#) ()
- char [expect_body](#) ()
- void [skip_body_tokens](#) ()
- char [consume_body_token](#) ()
- char [expect_eoi](#) ()

Variables

- char * [iden](#)
- int [ope](#)

5.7.1 Function Documentation

5.7.1.1 [analyze](#)()

```
char analyze ( )
```

Definition at line 3 of file parser.c.

5.7.1.2 [consume_body_token](#)()

```
char consume_body_token ( )
```

Definition at line 460 of file parser.c.

5.7.1.3 consume_token()

```
char consume_token ( )
```

Definition at line 33 of file parser.c.

5.7.1.4 expect()

```
char expect (
    enum token_type type,
    char required )
```

Definition at line 89 of file parser.c.

5.7.1.5 expect_asgn()

```
char expect_asgn (
    struct ht_item * item )
```

Definition at line 316 of file parser.c.

5.7.1.6 expect_body()

```
char expect_body ( )
```

Definition at line 432 of file parser.c.

5.7.1.7 expect_data_type()

```
char expect_data_type (
    char required )
```

Definition at line 107 of file parser.c.

5.7.1.8 expect_decl()

```
char expect_decl (
    char multiple,
    enum token_type type )
```

Definition at line 259 of file parser.c.

5.7.1.9 expect_eoi()

```
char expect_eoi ( )
```

Definition at line 518 of file parser.c.

5.7.1.10 expect_for()

```
char expect_for (
    unsigned short line,
    unsigned short column )
```

Definition at line 353 of file parser.c.

5.7.1.11 expect_iden()

```
char expect_iden (
    char left,
    char right )
```

Definition at line 167 of file parser.c.

5.7.1.12 expect_if()

```
char expect_if (
    unsigned short line,
    unsigned short column )
```

Definition at line 345 of file parser.c.

5.7.1.13 expect_lit()

```
char expect_lit ( )
```

Definition at line 153 of file parser.c.

5.7.1.14 expect_lit_or_iden()

```
char expect_lit_or_iden (
    char stack )
```

Definition at line 120 of file parser.c.

5.7.1.15 expect_operation()

```
char expect_operation ( )
```

Definition at line 235 of file parser.c.

5.7.1.16 expect_operator()

```
char expect_operator (
    char required )
```

Definition at line 222 of file parser.c.

5.7.1.17 expect_print()

```
char expect_print ( )
```

Definition at line 424 of file parser.c.

5.7.1.18 expect_read()

```
char expect_read ( )
```

Definition at line 428 of file parser.c.

5.7.1.19 expect_while()

```
char expect_while (
    unsigned short line,
    unsigned short column )
```

Definition at line 349 of file parser.c.

5.7.1.20 `rewind_token()`

```
void rewind_token ( )
```

Definition at line 14 of file parser.c.

5.7.1.21 `skip_body_tokens()`

```
void skip_body_tokens ( )
```

Definition at line 448 of file parser.c.

5.7.1.22 `skip_tokens()`

```
void skip_tokens ( )
```

Definition at line 23 of file parser.c.

5.7.2 Variable Documentation

5.7.2.1 `iden`

```
char* iden
```

Definition at line 4 of file parser.h.

5.7.2.2 `ope`

```
int ope
```

Definition at line 5 of file parser.h.

5.8 `ramiel.c` File Reference

```
#include "lexer.c"  
#include "parser.c"
```


Functions

- int [main](#) (int argc, char *args[])

5.8.1 Function Documentation

5.8.1.1 main()

```
int main (
    int argc,
    char * args[] )
```

Definition at line 4 of file ramiel.c.

5.9 README.md File Reference

5.10 stack.c File Reference

```
#include "stack.h"
```

Functions

- struct [stack](#) * [new_stack](#) ()
- struct [stack_item](#) * [new_stack_item](#) (enum [data_t](#) type, size_t size)
- void [push](#) (struct [stack](#) *[stack](#), struct [stack_item](#) *item)
- struct [stack_item](#) * [pop](#) (struct [stack](#) *[stack](#))
- char [stack_operation](#) (struct [stack](#) *[stack](#), char *operator)

5.10.1 Function Documentation

5.10.1.1 new_stack()

```
struct stack* new_stack ( )
```

Definition at line 3 of file stack.c.

5.10.1.2 new_stack_item()

```
struct stack\_item* new_stack_item (
    enum data\_t type,
    size_t size )
```

Definition at line 11 of file stack.c.

5.10.1.3 pop()

```
struct stack\_item* pop (
    struct stack * stack )
```

Definition at line 26 of file stack.c.

5.10.1.4 push()

```
void push (
    struct stack * stack,
    struct stack\_item * item )
```

Definition at line 20 of file stack.c.

5.10.1.5 stack_operation()

```
char stack_operation (
    struct stack * stack,
    char * operator )
```

Definition at line 38 of file stack.c.

5.11 stack.h File Reference

Data Structures

- struct [stack_item](#)
- struct [stack](#)

Functions

- struct [stack](#) * [new_stack](#) ()
- struct [stack_item](#) * [new_stack_item](#) (enum [data_t](#) type, size_t size)
- void [push](#) (struct [stack](#) *[stack](#), struct [stack_item](#) *item)
- struct [stack_item](#) * [pop](#) (struct [stack](#) *[stack](#))
- char [stack_operation](#) (struct [stack](#) *[stack](#), char *operator)

5.11.1 Function Documentation

5.11.1.1 new_stack()

```
struct stack* new_stack ( )
```

Definition at line 3 of file stack.c.

5.11.1.2 new_stack_item()

```
struct stack\_item* new_stack_item (
    enum data\_t type,
    size_t size )
```

Definition at line 11 of file stack.c.

5.11.1.3 pop()

```
struct stack\_item* pop (
    struct stack * stack )
```

Definition at line 26 of file stack.c.

5.11.1.4 push()

```
void push (
    struct stack * stack,
    struct stack\_item * item )
```

Definition at line 20 of file stack.c.

5.11.1.5 stack_operation()

```
char stack_operation (
    struct stack * stack,
    char * operator )
```

Definition at line 38 of file stack.c.

5.12 state.c File Reference

Data Structures

- struct [state](#)

Functions

- void [append_token](#) (struct [token_info](#) *n_info)

Variables

- struct [state](#) [lex_state](#) = (struct [state](#)) { NULL, NULL, NULL, NULL, NULL, NULL, 0, 0, 1, 0 }

5.12.1 Function Documentation

5.12.1.1 [append_token\(\)](#)

```
void append_token (  
    struct token\_info * n_info )
```

Definition at line 16 of file state.c.

5.12.2 Variable Documentation

5.12.2.1 [lex_state](#)

```
struct state lex_state = (struct state) { NULL, NULL, NULL, NULL, NULL, NULL, 0, 0, 1, 0 }
```

Definition at line 14 of file state.c.

5.13 tokens.c File Reference

Data Structures

- struct [token](#)
- struct [token_info](#)

Macros

- `#define D_TYPES 4`
- `#define LITERALS 4`
- `#define OPERATORS 13`
- `#define L_OPERATORS 2`
- `#define R_OPERATORS 2`

Enumerations

- enum `token_type` {
`O_SUM, O_INC, O_SUB, O_DEC,`
`O_MUL, O_DIV, O_MOD, O_NOT,`
`O_NEQ, O_LES, O_LEEQ, O_GRE,`
`O_GREQ, S_ASGN, O_EQ, O_AND,`
`O_OR, K_CHR, K_INT, K_FLT,`
`K_STR, K_IF, K_EIF, K_ELSE,`
`K_WHLE, K_FOR, K_PRNT, K_READ,`
`K_BRK, K_CONT, S_LPAR, S_RPAR,`
`S_LCBR, S_RCBR, S_LSBR, S_RSBR,`
`S_SCLN, S_PNT, S_CMA, T_IDEN,`
`T_CHR, T_INT, T_FLT, T_STR,`
`T_EOF` }

Functions

- struct `token_info` * `new_token_info` (struct `token` *n_token)

Variables

- char * `token_names` = "O_SUM O_INC O_SUB O_DEC O_MUL O_DIV O_MOD O_NOT O_NEQ O_LES O_LEEQ O_GRE O_GREQ S_ASGN O_EQ O_AND O_OR K_CHR K_INT K_FLT K_STR K_IF K_EIF K_ELSE K_WHLE K_FOR K_PRNT K_READ K_BRK K_CONT S_LPAR S_RPAR S_LCBR S_RCBR S_LSBR S_RSBR S_SCLN S_PNT S_CMA T_IDEN T_CHR T_INT T_FLT T_STR T_EOF"
- enum `token_type` `d_types` [D_TYPES] = { `K_CHR, K_INT, K_FLT, K_STR` }
- enum `token_type` `literals` [LITERALS] = { `T_CHR, T_INT, T_FLT, T_STR` }
- enum `token_type` `operators` [OPERATORS] = { `O_SUM, O_SUB, O_MUL, O_DIV, O_MOD, O_EQ, O_NEQ, O_GRE, O_GREQ, O_LES, O_LEEQ, O_AND, O_OR` }
- enum `token_type` `l_operators` [L_OPERATORS] = { `O_INC, O_DEC` }
- enum `token_type` `r_operators` [R_OPERATORS] = { `O_INC, O_DEC` }
- char * `d_types_code` = "CIDS"
- char * `d_types_mnemonic` [D_TYPES]
- char * `operator_code` = "ADDSUBMULDIVMODCEQCNECGTCGECLTCLE"
- char * `l_operator_code` [L_OPERATORS]
- char * `r_operator_code` [R_OPERATORS]

5.13.1 Macro Definition Documentation

5.13.1.1 D_TYPES

```
#define D_TYPES 4
```

Definition at line 11 of file tokens.c.

5.13.1.2 L_OPERATORS

```
#define L_OPERATORS 2
```

Definition at line 14 of file tokens.c.

5.13.1.3 LITERALS

```
#define LITERALS 4
```

Definition at line 12 of file tokens.c.

5.13.1.4 OPERATORS

```
#define OPERATORS 13
```

Definition at line 13 of file tokens.c.

5.13.1.5 R_OPERATORS

```
#define R_OPERATORS 2
```

Definition at line 15 of file tokens.c.

5.13.2 Enumeration Type Documentation

5.13.2.1 token_type

```
enum token_type
```

Enumerator

O_SUM	
O_INC	
O_SUB	
O_DEC	
O_MUL	
O_DIV	
O_MOD	
O_NOT	
O_NEQ	
O_LES	
O_LEEQ	
O_GRE	
O_GREQ	
S_ASGN	
O_EQ	
O_AND	
O_OR	
K_CHR	
K_INT	
K_FLT	
K_STR	
K_IF	
K_EIF	
K_ELSE	
K_WHLE	
K_FOR	
K_PRNT	
K_READ	
K_BRK	
K_CONT	
S_LPAR	
S_RPAR	
S_LCBR	
S_RCBR	
S_LSBR	
S_RSBR	
S_SCLN	
S_PNT	
S_CMA	
T_IDEN	
T_CHR	
T_INT	
T_FLT	
T_STR	
T_EOF	

Definition at line 3 of file tokens.c.

5.13.3 Function Documentation

5.13.3.1 new_token_info()

```
struct token_info* new_token_info (
    struct token * n_token )
```

Definition at line 58 of file tokens.c.

5.13.4 Variable Documentation

5.13.4.1 d_types

```
enum token_type d_types[D_TYPES] = { K_CHR, K_INT, K_FLT, K_STR }
```

Definition at line 17 of file tokens.c.

5.13.4.2 d_types_code

```
char* d_types_code = "CIDS"
```

Definition at line 23 of file tokens.c.

5.13.4.3 d_types_mnemonic

```
char* d_types_mnemonic[D_TYPES]
```

Initial value:

```
= {
    "char",
    "integer",
    "float",
    "string"
}
```

Definition at line 24 of file tokens.c.

5.13.4.4 l_operator_code

```
char* l_operator_code[L_OPERATORS]
```

Initial value:

```
= {  
    "PUSH %s\nPUSHKI 1\nADD\nPOP%c %s\n",  
    "PUSH %s\nPUSHKI 1\nSUB\nPOP%c %s\n"  
}
```

Definition at line 31 of file tokens.c.

5.13.4.5 l_operators

```
enum token_type l_operators[L_OPERATORS] = { O_INC, O_DEC }
```

Definition at line 20 of file tokens.c.

5.13.4.6 literals

```
enum token_type literals[LITERALS] = { T_CHR, T_INT, T_FLT, T_STR }
```

Definition at line 18 of file tokens.c.

5.13.4.7 operator_code

```
char* operator_code = "ADDSUBMULDIVMODCEQCNECGTCGECLTCL"
```

Definition at line 30 of file tokens.c.

5.13.4.8 operators

```
enum token_type operators[OPERATORS] = { O_SUM, O_SUB, O_MUL, O_DIV, O_MOD, O_EQ, O_NEQ, O_GRE,  
O_GREQ, O_LES, O_LEEQ, O_AND, O_OR }
```

Definition at line 19 of file tokens.c.

5.13.4.9 r_operator_code

```
char* r_operator_code[R_OPERATORS]
```

Initial value:

```
= {
    "PUSH %s\nPUSHKI 1\nADD\nPOP%c %s\n",
    "PUSH %s\nPUSHKI 1\nSUB\nPOP%c %s\n"
}
```

Definition at line 35 of file tokens.c.

5.13.4.10 r_operators

```
enum token_type r_operators[R_OPERATORS] = { O_INC, O_DEC }
```

Definition at line 21 of file tokens.c.

5.13.4.11 token_names

```
char* token_names = "O_SUM O_INC O_SUB O_DEC O_MUL O_DIV O_MOD O_NOT O_NEQ O_LES O_LEEQ O_GRE
O_GREQ S_ASGN O_EQ O_AND O_OR K_CHR K_INT K_FLT K_STR K_IF K_EIF K_ELSE K_WHLE K_FOR K_PRNT
K_READ K_BRK K_CONT S_LPAR S_RPAR S_LCBR S_RCBR S_LSBR S_RSBR S_SCLN S_PNT S_CMA T_IDEN T_CHR
T_INT T_FLT T_STR T_EOF"
```

Definition at line 1 of file tokens.c.

5.14 utils.c File Reference

Macros

- #define `max(a, b)` ((a) > (b) ? (a) : (b))
- #define `arr_len(arr)` (sizeof(arr) / sizeof(arr[0]))

Functions

- char * `append` (char *array, char a)

5.14.1 Macro Definition Documentation

5.14.1.1 arr_len

```
#define arr_len(  
    arr ) (sizeof(arr) / sizeof(arr[0]))
```

Definition at line 3 of file utils.c.

5.14.1.2 max

```
#define max(  
    a,  
    b ) ((a) > (b) ? (a) : (b))
```

Definition at line 1 of file utils.c.

5.14.2 Function Documentation

5.14.2.1 append()

```
char* append (  
    char * array,  
    char a )
```

Definition at line 5 of file utils.c.

Index

- analyze
 - parser.c, [29](#)
 - parser.h, [34](#)
- append
 - utils.c, [49](#)
- append_token
 - state.c, [42](#)
- arr_len
 - utils.c, [48](#)
- C_RST
 - console.c, [18](#)
- C_B
 - console.c, [17](#)
- C_G
 - console.c, [17](#)
- C_R
 - console.c, [18](#)
- C_Y
 - console.c, [18](#)
- column
 - state, [11](#)
 - token_info, [14](#)
- console.c, [17](#)
 - C_RST, [18](#)
 - C_B, [17](#)
 - C_G, [17](#)
 - C_R, [18](#)
 - C_Y, [18](#)
 - error_m, [18](#)
 - standard_m, [18](#)
 - warning_m, [19](#)
- consume_body_token
 - parser.c, [29](#)
 - parser.h, [34](#)
- consume_token
 - parser.c, [30](#)
 - parser.h, [34](#)
- content
 - token, [13](#)
- count
 - hashtable, [7](#)
 - stack, [9](#)
- current
 - state, [11](#)
- D_TYPES
 - tokens.c, [43](#)
- d_types
 - tokens.c, [46](#)
- d_types_code
 - tokens.c, [46](#)
- d_types_mnemonic
 - tokens.c, [46](#)
- data_t
 - hashtable.h, [21](#)
- data_t_str
 - hashtable.h, [23](#)
- error_m
 - console.c, [18](#)
- errors
 - state, [11](#)
- expect
 - parser.c, [30](#)
 - parser.h, [35](#)
- expect_asgn
 - parser.c, [30](#)
 - parser.h, [35](#)
- expect_body
 - parser.c, [30](#)
 - parser.h, [35](#)
- expect_data_type
 - parser.c, [30](#)
 - parser.h, [35](#)
- expect_decl
 - parser.c, [30](#)
 - parser.h, [35](#)
- expect_eoi
 - parser.c, [31](#)
 - parser.h, [35](#)
- expect_for
 - parser.c, [31](#)
 - parser.h, [36](#)
- expect_iden
 - parser.c, [31](#)
 - parser.h, [36](#)
- expect_if
 - parser.c, [31](#)
 - parser.h, [36](#)
- expect_lit
 - parser.c, [31](#)
 - parser.h, [36](#)
- expect_lit_or_iden
 - parser.c, [32](#)
 - parser.h, [36](#)
- expect_operation
 - parser.c, [32](#)
 - parser.h, [37](#)
- expect_operator

- parser.c, [32](#)
 - parser.h, [37](#)
- expect_print
 - parser.c, [32](#)
 - parser.h, [37](#)
- expect_read
 - parser.c, [32](#)
 - parser.h, [37](#)
- expect_while
 - parser.c, [32](#)
 - parser.h, [37](#)
- hash
 - hashtable.c, [19](#)
 - hashtable.h, [22](#)
- hash_item
 - hashtable.c, [19](#)
 - hashtable.h, [22](#)
- hashtable, [7](#)
 - count, [7](#)
 - items, [7](#)
 - size, [7](#)
- hashtable.c, [19](#)
 - hash, [19](#)
 - hash_item, [19](#)
 - lookup_item, [20](#)
 - new_ht, [20](#)
 - new_ht_item, [20](#)
 - rm_ht, [20](#)
 - rm_ht_helper, [20](#)
 - rm_ht_item, [21](#)
- hashtable.h, [21](#)
 - data_t, [21](#)
 - data_t_str, [23](#)
 - hash, [22](#)
 - hash_item, [22](#)
 - lookup_item, [22](#)
 - new_ht, [22](#)
 - new_ht_item, [22](#)
 - rm_ht, [23](#)
 - rm_ht_helper, [23](#)
 - rm_ht_item, [23](#)
- ht_item, [8](#)
 - identifier, [8](#)
 - next, [8](#)
 - size, [8](#)
 - type, [9](#)
- iden
 - parser.h, [38](#)
- identifier
 - ht_item, [8](#)
- input
 - state, [12](#)
- items
 - hashtable, [7](#)
- L_OPERATORS
 - tokens.c, [44](#)
- l_operator_code
 - tokens.c, [46](#)
- l_operators
 - tokens.c, [47](#)
- LITERALS
 - tokens.c, [44](#)
- lex_state
 - state.c, [42](#)
- lexer.c, [24](#)
 - make_tokens, [24](#)
 - next_char, [24](#)
 - parse, [24](#)
 - parse_char, [24](#)
 - parse_number, [25](#)
 - parse_string, [25](#)
 - parse_symbol_or_operand, [25](#)
 - parse_word, [25](#)
 - parse_word_or_number, [25](#)
 - return_char, [26](#)
 - skip_line, [26](#)
- lexer.h, [26](#)
 - make_tokens, [27](#)
 - next_char, [27](#)
 - parse, [27](#)
 - parse_char, [27](#)
 - parse_number, [27](#)
 - parse_string, [27](#)
 - parse_symbol_or_operand, [28](#)
 - parse_word, [28](#)
 - parse_word_or_number, [28](#)
 - return_char, [28](#)
 - skip_line, [28](#)
- line
 - state, [12](#)
 - token_info, [14](#)
- list
 - state, [12](#)
- literals
 - tokens.c, [47](#)
- lookup_item
 - hashtable.c, [20](#)
 - hashtable.h, [22](#)
- main
 - ramiel.c, [39](#)
- make_tokens
 - lexer.c, [24](#)
 - lexer.h, [27](#)
- max
 - utils.c, [49](#)
- negative
 - token_info, [15](#)
- new_ht
 - hashtable.c, [20](#)
 - hashtable.h, [22](#)
- new_ht_item
 - hashtable.c, [20](#)
 - hashtable.h, [22](#)

- new_stack
 - stack.c, [39](#)
 - stack.h, [41](#)
- new_stack_item
 - stack.c, [39](#)
 - stack.h, [41](#)
- new_token_info
 - tokens.c, [46](#)
- next
 - ht_item, [8](#)
 - stack_item, [10](#)
 - token_info, [15](#)
- next_char
 - lexer.c, [24](#)
 - lexer.h, [27](#)
- OPERATORS
 - tokens.c, [44](#)
- ope
 - parser.h, [38](#)
- operator_code
 - tokens.c, [47](#)
- operators
 - tokens.c, [47](#)
- output
 - state, [12](#)
- parse
 - lexer.c, [24](#)
 - lexer.h, [27](#)
- parse_char
 - lexer.c, [24](#)
 - lexer.h, [27](#)
- parse_number
 - lexer.c, [25](#)
 - lexer.h, [27](#)
- parse_string
 - lexer.c, [25](#)
 - lexer.h, [27](#)
- parse_symbol_or_operand
 - lexer.c, [25](#)
 - lexer.h, [28](#)
- parse_word
 - lexer.c, [25](#)
 - lexer.h, [28](#)
- parse_word_or_number
 - lexer.c, [25](#)
 - lexer.h, [28](#)
- parser.c, [29](#)
 - analyze, [29](#)
 - consume_body_token, [29](#)
 - consume_token, [30](#)
 - expect, [30](#)
 - expect_asgn, [30](#)
 - expect_body, [30](#)
 - expect_data_type, [30](#)
 - expect_decl, [30](#)
 - expect_eoi, [31](#)
 - expect_for, [31](#)
 - expect_iden, [31](#)
 - expect_if, [31](#)
 - expect_lit, [31](#)
 - expect_lit_or_iden, [32](#)
 - expect_operation, [32](#)
 - expect_operator, [32](#)
 - expect_print, [32](#)
 - expect_read, [32](#)
 - expect_while, [32](#)
 - rewind_token, [33](#)
 - skip_body_tokens, [33](#)
 - skip_tokens, [33](#)
- parser.h, [33](#)
 - analyze, [34](#)
 - consume_body_token, [34](#)
 - consume_token, [34](#)
 - expect, [35](#)
 - expect_asgn, [35](#)
 - expect_body, [35](#)
 - expect_data_type, [35](#)
 - expect_decl, [35](#)
 - expect_eoi, [35](#)
 - expect_for, [36](#)
 - expect_iden, [36](#)
 - expect_if, [36](#)
 - expect_lit, [36](#)
 - expect_lit_or_iden, [36](#)
 - expect_operation, [37](#)
 - expect_operator, [37](#)
 - expect_print, [37](#)
 - expect_read, [37](#)
 - expect_while, [37](#)
 - iden, [38](#)
 - ope, [38](#)
 - rewind_token, [37](#)
 - skip_body_tokens, [38](#)
 - skip_tokens, [38](#)
- pop
 - stack.c, [40](#)
 - stack.h, [41](#)
- push
 - stack.c, [40](#)
 - stack.h, [41](#)
- R_OPERATORS
 - tokens.c, [44](#)
- r_operator_code
 - tokens.c, [47](#)
- r_operators
 - tokens.c, [48](#)
- README.md, [39](#)
- ramiel.c, [38](#)
 - main, [39](#)
- return_char
 - lexer.c, [26](#)
 - lexer.h, [28](#)
- rewind_token
 - parser.c, [33](#)
 - parser.h, [37](#)

- rm_ht
 - hashtable.c, 20
 - hashtable.h, 23
- rm_ht_helper
 - hashtable.c, 20
 - hashtable.h, 23
- rm_ht_item
 - hashtable.c, 21
 - hashtable.h, 23
- size
 - hashtable, 7
 - ht_item, 8
 - stack_item, 10
- skip_body_tokens
 - parser.c, 33
 - parser.h, 38
- skip_line
 - lexer.c, 26
 - lexer.h, 28
- skip_tokens
 - parser.c, 33
 - parser.h, 38
- stack, 9
 - count, 9
 - state, 12
 - top, 9
- stack.c, 39
 - new_stack, 39
 - new_stack_item, 39
 - pop, 40
 - push, 40
 - stack_operation, 40
- stack.h, 40
 - new_stack, 41
 - new_stack_item, 41
 - pop, 41
 - push, 41
 - stack_operation, 41
- stack_item, 10
 - next, 10
 - size, 10
 - type, 10
- stack_operation
 - stack.c, 40
 - stack.h, 41
- standard_m
 - console.c, 18
- state, 11
 - column, 11
 - current, 11
 - errors, 11
 - input, 12
 - line, 12
 - list, 12
 - output, 12
 - stack, 12
 - variables, 12
 - warnings, 13
- state.c, 42
 - append_token, 42
 - lex_state, 42
- tok
 - token_info, 15
- token, 13
 - content, 13
 - type, 13
- token_info, 14
 - column, 14
 - line, 14
 - negative, 15
 - next, 15
 - tok, 15
 - val_c, 15
 - val_d, 15
 - val_i, 15
 - val_s, 15
- token_names
 - tokens.c, 48
- token_type
 - tokens.c, 44
- tokens.c, 42
 - D_TYPES, 43
 - d_types, 46
 - d_types_code, 46
 - d_types_mnemonic, 46
 - L_OPERATORS, 44
 - l_operator_code, 46
 - l_operators, 47
 - LITERALS, 44
 - literals, 47
 - new_token_info, 46
 - OPERATORS, 44
 - operator_code, 47
 - operators, 47
 - R_OPERATORS, 44
 - r_operator_code, 47
 - r_operators, 48
 - token_names, 48
 - token_type, 44
- top
 - stack, 9
- type
 - ht_item, 9
 - stack_item, 10
 - token, 13
- utils.c, 48
 - append, 49
 - arr_len, 48
 - max, 49
- val_c
 - token_info, 15
- val_d
 - token_info, 15
- val_i

- token_info, [15](#)
- val_s
 - token_info, [15](#)
- variables
 - state, [12](#)
- warning_m
 - console.c, [19](#)
- warnings
 - state, [13](#)