

My Project

Generated by Doxygen 1.8.9.1

Mon Dec 5 2016 22:21:00

Contents

1	Module Index	1
1.1	Modules	1
2	Class Index	3
2.1	Class List	3
3	Module Documentation	5
3.1	Garbage collector	5
3.1.1	Detailed Description	5
3.1.2	Typedef Documentation	5
3.1.2.1	mem_item_t	5
3.1.2.2	mem_list_t	6
3.1.3	Function Documentation	6
3.1.3.1	free_memory	6
3.1.3.2	mem_alloc	6
3.1.3.3	mem_realloc	6
3.1.4	Variable Documentation	6
3.1.4.1	GARBAGE_COLLECTOR	6
3.2	Lexical analysis	7
3.2.1	Detailed Description	8
3.2.2	Macro Definition Documentation	8
3.2.2.1	reset_scanner	8
3.2.2.2	S_SIZE	8
3.2.2.3	SPEC_CHAR_FSEEK	8
3.2.3	Typedef Documentation	8
3.2.3.1	token	8
3.2.4	Enumeration Type Documentation	8
3.2.4.1	anonymous enum	8
3.2.5	Function Documentation	9
3.2.5.1	bin2dec	9
3.2.5.2	get_token	10
3.2.5.3	hex2dec_double	10

3.2.5.4	hex2dec_int	10
3.2.5.5	is_full_ident	10
3.2.5.6	is_keyword	11
3.2.5.7	is_num_literal	12
3.2.5.8	is_simple_ident	12
3.2.5.9	is_special_char	12
3.2.5.10	load_string	13
3.2.5.11	make_power	13
3.2.5.12	octal2dec	13
3.2.5.13	skip_comment	14
3.2.5.14	str2num	14
3.3	Structures	15
3.3.1	Detailed Description	15
3.3.2	Macro Definition Documentation	15
3.3.2.1	ARRAY_HTAB_INIT_SIZE	15
3.3.2.2	ARRAY_STRING_INIT_SIZE	16
3.3.2.3	STACK_HTAB_INIT_SIZE	16
3.3.3	Typedef Documentation	16
3.3.3.1	array_htab	16
3.3.3.2	array_string	16
3.3.3.3	stack_htab	16
3.3.4	Function Documentation	16
3.3.4.1	array_htab_destroy	16
3.3.4.2	array_htab_get_item	16
3.3.4.3	array_htab_init	17
3.3.4.4	array_htab_insert	18
3.3.4.5	array_string_destroy	18
3.3.4.6	array_string_find	18
3.3.4.7	array_string_init	18
3.3.4.8	array_string_insert	19
3.3.4.9	stack_htab_destroy	19
3.3.4.10	stack_htab_get_first	19
3.3.4.11	stack_htab_get_item	19
3.3.4.12	stack_htab_init	20
3.3.4.13	stack_htab_pop	20
3.3.4.14	stack_htab_push	20
4	Class Documentation	21
4.1	array_htab Struct Reference	21
4.1.1	Detailed Description	21

4.1.2	Member Data Documentation	21
4.1.2.1	data	21
4.1.2.2	idx	21
4.1.2.3	size	21
4.2	array_string Struct Reference	21
4.2.1	Detailed Description	22
4.2.2	Member Data Documentation	22
4.2.2.1	data	22
4.2.2.2	idx	22
4.2.2.3	size	22
4.3	mem_item_t Struct Reference	22
4.3.1	Detailed Description	22
4.3.2	Member Data Documentation	22
4.3.2.1	next	22
4.3.2.2	ptr	23
4.4	mem_list_t Struct Reference	23
4.4.1	Detailed Description	23
4.4.2	Member Data Documentation	23
4.4.2.1	first	23
4.4.2.2	last	23
4.5	stack_htab Struct Reference	23
4.5.1	Detailed Description	23
4.5.2	Member Data Documentation	24
4.5.2.1	data	24
4.5.2.2	size	24
4.5.2.3	top	24
4.6	token Struct Reference	24
4.6.1	Detailed Description	24
4.6.2	Member Data Documentation	24
4.6.2.1	id	24
4.6.2.2	ptr	24
	Index	25

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Garbage collector	5
Lexical analysis	7
Structures	15

Chapter 2

Class Index

2.1 Class List

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

array_htab	21
array_string	21
mem_item_t	22
mem_list_t	23
stack_htab	23
token	24

Chapter 3

Module Documentation

3.1 Garbage collector

Garbage collector is group of functions that allocate memory and store pointers into list to prevent memory leaks.

Classes

- struct [mem_list_t](#)
- struct [mem_item_t](#)

Typedefs

- typedef struct [mem_list_t](#) [mem_list_t](#)
- typedef struct [mem_item_t](#) [mem_item_t](#)

Functions

- void **mem_list_t_init** ()
- void * [mem_alloc](#) (size_t size)
- void [free_memory](#) ()
- void * [mem_realloc](#) (void *ptr, size_t size)

Variables

- [mem_list_t](#) [GARBAGE_COLLECTOR](#)

3.1.1 Detailed Description

Garbage collector is group of functions that allocate memory and store pointers into list to prevent memory leaks.

Author: Matejka Jiri Login: xmatej52 School: VUT FIT, BRNO Project: Interpret for IFJ16 gcc version: 5.4.0 (ubuntu 16.04.2) Date: 2016-12-03

3.1.2 Typedef Documentation

3.1.2.1 typedef struct [mem_item_t](#) [mem_item_t](#)

Item that holds one pointer to allocated memory

3.1.2.2 typedef struct mem_list_t mem_list_t

List of items, that holds allocated memory

3.1.3 Function Documentation

3.1.3.1 void free_memory ()

Free all memory allocated with this module

Precondition

Function mem_list_t_init was called before

3.1.3.2 void* mem_alloc (size_t size)

Allocate memory

Parameters

<i>size</i>	Number of memory that will be allocated in bytes
-------------	--

Returns

Pointer to allocated memory, NULL when allocation fails

Precondition

Function mem_list_t_init was called before

3.1.3.3 void* mem_realloc (void * ptr, size_t size)

Reallocate memory for new size

Parameters

<i>ptr</i>	Pointer to memory that will be reallocated
<i>size</i>	New size of memory for allocation

Returns

Pointer to allocated memory, NULL when allocation fails

Precondition

Function mem_list_t_init was called before

3.1.4 Variable Documentation

3.1.4.1 GARBAGE_COLLECTOR

Global list of items that holds allocated memory

Initialize

3.2 Lexical analysis

Lexical analysis analyse input source code and check, if it is subject of the language IFJ16. Also transfer input source code into tokens.

Classes

- struct `token`

Macros

- #define `S_SIZE` 32
- #define `reset_scanner()` (fseek(f, LINE_NUM = 0, SEEK_SET))
- #define `SPEC_CHAR_FSEEK(spec)` (((spec) == `S_EQUAL` || (spec) == `S_LESS_EQUAL` || (spec) == `S_GREATER_EQUAL` || (spec) == `S_NOT_EQUAL`)?-2:-1)

Typedefs

- typedef struct `token` `token`

Enumerations

- enum {
`S_BOOLEAN` = 1, `S_BREAK`, `S_CLASS`, `S_CONTINUE`,
`S_DO`, `S_DOUBLE`, `S_ELSE`, `S_FALSE`,
`S_FOR`, `S_IF`, `S_INT`, `S_RETURN`,
`S_STRING`, `S_STATIC`, `S_TRUE`, `S_VOID`,
`S_WHILE`, `TYPE_DOUBLE`, `TYPE_INT`, `TYPE_STRING`,
`TYPE_BOOLEAN`, `TYPE_INT_BIN`, `TYPE_INT_OCTAL`, `TYPE_INT_HEX`,
`TYPE_DOUBLE_HEX`, `BLOCK_COMMENT`, `LINE_COMMENT`, `S_SIMPLE_IDENT`,
`S_FULL_IDENT`, `S_EQUAL`, `S_LESS_EQUAL`, `S_GREATER_EQUAL`,
`S_LESS`, `S_GREATER`, `S_OR`, `S_AND`,
`S_NOT_EQUAL`, `S_LEFT_PARE`, `S_RIGHT_PARE`, `S_LEFT_BRACE`,
`S_RIGHT_BRACE`, `S_COMMA`, `S_SEMICOMMA`, `S_PLUS`,
`S_MINUS`, `S_DIV`, `S_MUL`, `S_ASSIGNMENT`,
`S_EOF` }

Functions

- int `is_keyword` (char *word)
- int `is_special_char` (char c)
- int `is_num_literal` (char *word, unsigned len)
- int `is_simple_ident` (char *word, unsigned len)
- int `is_full_ident` (char *word, unsigned len)
- int `skip_comment` (unsigned comment_type)
- char * `load_string` (char *word, int *max)
- double `make_power` (double x, long int exp)
- void `bin2dec` (char *str, int *result)
- void `octal2dec` (char *str, int *result)
- void `hex2dec_int` (char *str, int *result)
- void `hex2dec_double` (char *str, double *result)
- void * `str2num` (char *str, int type, int *valide)
- `token get_token` ()

Variables

- unsigned **LINE_NUM**
- FILE * **f**

3.2.1 Detailed Description

Lexical analysis analysi input source code and check, if itis subject of the language IFJ16. ALso transfer input source code into tokens.

Author: Matejka Jiri Login: xmatej52 School: VUT FIT, BRNO Project: Interpret for IFJ16 gcc version: 5.4.0 (ubuntu 16.04.2) Date: 2016-12-03

3.2.2 Macro Definition Documentation

3.2.2.1 `#define reset_scanner() (fseek(f, LINE_NUM = 0, SEEK_SET))`

Macro that set offset at the begining of file

3.2.2.2 `#define S_SIZE 32`

Default size for memory allocation

3.2.2.3 `#define SPEC_CHAR_FSEEK(spec) (((spec) == S_EQUAL || (spec) == S_LESS_EQUAL || (spec) == S_GREATER_EQUAL || (spec) == S_NOT_EQUAL)?-2:-1)`

Macro that return number how much will be offset returned

3.2.3 Typedef Documentation

3.2.3.1 `typedef struct token token`

Structure that represents token

3.2.4 Enumeration Type Documentation

3.2.4.1 anonymous enum

Enumerator

- S_BOOLEAN** Keyword boolean
- S_BREAK** Keyword break
- S_CLASS** Keyword class
- S_CONTINUE** Keyword continue
- S_DO** Keyword do
- S_DOUBLE** Keyword double
- S_ELSE** Keyword else
- S_FALSE** Keyword false
- S_FOR** Keyword for
- S_IF** Keyword if
- S_INT** Keyword int

S_RETURN Keyword return
S_STRING Keyword String
S_STATIC Keyword static
S_TRUE Keyword true
S_VOID Keyword void
S_WHILE Keyword while
TYPE_DOUBLE data type double
TYPE_INT data type int
TYPE_STRING data type String
TYPE_BOOLEAN data type boolean
TYPE_INT_BIN Integer written in binary
TYPE_INT_OCTAL Integer written in octal
TYPE_INT_HEX Integer written in hex
TYPE_DOUBLE_HEX Double written in hex
BLOCK_COMMENT identifikator of block comment
LINE_COMMENT identifikator of one line comment
S_SIMPLE_IDENT stands for simple identifikator
S_FULL_IDENT stands for full identifikator
S_EQUAL stands for ==
S_LESS_EQUAL stands for <=
S_GREATER_EQUAL stands for >=
S_LESS stands for <
S_GREATER stands for >
S_OR stands for ||
S_AND stands for &&
S_NOT_EQUAL stands for !=
S_LEFT_PARE stands for (
S_RIGHT_PARE stands for)
S_LEFT_BRACE stands for {
S_RIGHT_BRACE stands for }
S_COMMA stands for ,
S_SEMICOMMA stands for ;
S_PLUS stands for +
S_MINUS stands for -
S_DIV stands for /
S_MUL stands for *
S_ASSIGNMENT stands for =
S_EOF stands for EOF

3.2.5 Function Documentation

3.2.5.1 void bin2dec (char * str, int * result)

Convert string to decimal if string represents binnary integer number

Parameters

<i>str</i>	String for conversion
<i>result</i>	Converted number

3.2.5.2 token get_token ()

Retrive token from source code

Precondition

global variable *f* is already opened file

Postcondition

token.id > 0 (0 in case of lexical error, otherwise error while setting offset or allocating memory)

Returns

token, where *token.id* is identifikator and *token.ptr* is string (or poiter to NULL if string is not needed)

3.2.5.3 void hex2dec_double (char * *str*, double * *result*)

Convert string to decimal number if string represents hexadecimal floating point number

Parameters

<i>str</i>	String for conversion
<i>result</i>	Converted number

3.2.5.4 void hex2dec_int (char * *str*, int * *result*)

Convert string to decimal number if string represents hexadecimal integer number

Parameters

<i>str</i>	String for conversion
<i>result</i>	Converted number

3.2.5.5 int is_full_ident (char * *word*, unsigned *len*)

Detect if input string is full identifikator or not

Parameters

<i>word</i>	String (or array of chars) for detection
<i>len</i>	length of word (without '\0', if there is)

Precondition

size of allocated space for word is bigger or equal *len*

Returns

1 if word represents full identifikator, otherwise return 0

3.2.5.6 `int is_keyword (char * word)`

Detect if input String is key word or not

Parameters

<i>word</i>	String (or array of chars) for detection
-------------	--

Precondition

Word is ended by char '\0'

Returns

If word represents key word, return id of specific key word, otherwise return 0

3.2.5.7 int is_num_literal (char * word, unsigned len)

Detect if input string is numeric literal or not

Parameters

<i>word</i>	String (or array of chars) for detection
<i>len</i>	length of word (without '\0', if there is)

Precondition

size of allocated space for word is bigger or equal len

Returns

If word is numeric literal, return TYPE_INT for integer or TYPE_DOUBLE for double, otherwise return 0

3.2.5.8 int is_simple_ident (char * word, unsigned len)

Detect if input string is simple identifikator or not

Parameters

<i>word</i>	String (or array of chars) for detection
<i>len</i>	length of word (without '\0', if there is)

Precondition

size of allocated space for word is bigger or equal len

Returns

1 if word represents simple identifikator, otherwise return 0

3.2.5.9 int is_special_char (char c)

Detect if input char represents some of special chars like =, +, ,, ..., also detect if there is >=, ==, != or <= in file

Parameters

<i>c</i>	input char
----------	------------

Precondition

global variable *f* is already opened file

Returns

if input is special char, return its value (set by enum) otherwise return 0

3.2.5.10 `char* load_string (char * word, int * max)`

Load chars until function reach end of string

Parameters

<i>word</i>	pointer to allocated space for saving chars from stream
<i>max</i>	pointer to length of allocated space in bytes

Precondition

global variable *f* is already opened file
word points to already allocated space
**max* ≥ 1

Returns

Loaded string, returns NULL when function reach EOF or EOL and set *max* to -1 or returns NULL and set **max* to zero, if reallocation fails or return NULL and set *max* to -2 if there is invalid use of escape sekvence

3.2.5.11 `double make_power (double x, long int exp)`

Count power

Parameters

<i>x</i>	Cardinal number
<i>exp</i>	Exponent

Returns

Result of *x* to the exponent

3.2.5.12 `void octal2dec (char * str, int * result)`

Convert string to decimal number if string represents octal integer number

Parameters

<i>str</i>	String for conversion
------------	-----------------------

<i>result</i>	Converted number
---------------	------------------

3.2.5.13 int skip_comment (unsigned *comment_type*)

Ignore all chars until end of comment

Parameters

<i>comment_type</i>	Type of comment (LINE_COMMENT or BLOCK_COMMENT)
---------------------	---

Precondition

global variable *f* is already opened file or active stream

Returns

0 when skipped comment, return 1 when comment was ended by EOF or return -1, if end of BLOCK_COMMENT was not found

3.2.5.14 void* str2num (char * *str*, int *type*, int * *valide*)

Convert string into double or integer (depends on type variable) and store it into new allocated space

Parameters

<i>str</i>	String that represents number
<i>type</i>	Type of number that represent string (should be TYPE_INT or TYPE_DOUBLE)
<i>valide</i>	Variable that will be set into 0 in case of succes, into 1 in case of error while allocating memory, into 2 in case of invalide string or into 3 in case of invalide type

Returns

Pointer into value that is result of conversion

3.3 Structures

Structures is group of structures and functions upon them.

Classes

- struct [stack_htab](#)
- struct [array_htab](#)
- struct [array_string](#)

Macros

- #define [STACK_HTAB_INIT_SIZE](#) 16
- #define [ARRAY_HTAB_INIT_SIZE](#) 64
- #define [ARRAY_STRING_INIT_SIZE](#) 8

Typedefs

- typedef struct [stack_htab](#) [stack_htab](#)
- typedef struct [array_htab](#) [array_htab](#)
- typedef struct [array_string](#) [array_string](#)

Functions

- int [stack_htab_init](#) ([stack_htab](#) *stack)
- int [stack_htab_push](#) ([stack_htab](#) *stack, [htab_t](#) *table)
- [htab_t](#) * [stack_htab_pop](#) ([stack_htab](#) *stack)
- [htab_t](#) * [stack_htab_get_item](#) ([stack_htab](#) *stack, unsigned backtrack)
- [htab_t](#) * [stack_htab_get_first](#) ([stack_htab](#) *stack)
- void [stack_htab_destroy](#) ([stack_htab](#) *stack)
- int [array_htab_init](#) ([array_htab](#) *array)
- int [array_htab_insert](#) ([array_htab](#) *array, [htab_t](#) *htab)
- [htab_t](#) * [array_htab_get_item](#) ([array_htab](#) *array, unsigned idx)
- void [array_htab_destroy](#) ([array_htab](#) *array)
- int [array_string_init](#) ([array_string](#) *array)
- int [array_string_insert](#) ([array_string](#) *array, const char *str)
- char * [array_string_find](#) ([array_string](#) *array, const char *str)
- void [array_string_destroy](#) ([array_string](#) *array)

3.3.1 Detailed Description

Structures is group of structures and functions upon them.

Author: Matejka Jiri Login: xmatej52 School: VUT FIT, BRNO Project: Interpret for IFJ16 gcc version: 5.4.0 (ubuntu 16.04.2) Date: 2016-12-03

3.3.2 Macro Definition Documentation

3.3.2.1 #define ARRAY_HTAB_INIT_SIZE 64

Default size for allocation memory for array of hash tables

3.3.2.2 `#define ARRAY_STRING_INIT_SIZE 8`

Default size for allocation memory for array of strings

3.3.2.3 `#define STACK_HTAB_INIT_SIZE 16`

Default size for allocation memory for Stack of hash tables

3.3.3 Typedef Documentation

3.3.3.1 `typedef struct array_htab array_htab`

Array of hash tables

3.3.3.2 `typedef struct array_string array_string`

Array of strings

3.3.3.3 `typedef struct stack_htab stack_htab`

Stack of hash tables

3.3.4 Function Documentation

3.3.4.1 `void array_htab_destroy (array_htab * array)`

Free all memory allocated by array and all memory allocated by all hash tables in array

Parameters

<i>array</i>	Array that shall be freed
--------------	---------------------------

Precondition

Array was inicialized

3.3.4.2 `htab_t* array_htab_get_item (array_htab * array, unsigned idx)`

Retrive specific item from array

Parameters

<i>array</i>	Array with items
<i>idx</i>	Index in array

Returns

Pointer to specific item or NULL if item on index is not inicialized

Precondition

Array was inicialized

3.3.4.3 `int array_htab_init (array_htab * array)`

Initialize array

Parameters

<i>array</i>	array for initialization
--------------	--------------------------

Returns

0 on succes, 1 if memory allocation failed

3.3.4.4 int array_htab_insert (array_htab * array, htab_t * htab)

Insert item into array and also reallocate memory if array is full

Parameters

<i>array</i>	Array where item will be inserted
<i>htab</i>	Item (pointer to hash table) that will be inserted

Returns

0 on succes, 1 when reallocation failed

Precondition

Array was inicialized

3.3.4.5 void array_string_destroy (array_string * array)

Free all memory allocated by array

Parameters

<i>array</i>	Array that will be freed
--------------	--------------------------

Precondition

Array was inicialized

3.3.4.6 char* array_string_find (array_string * array, const char * str)

Find string in array

Parameters

<i>array</i>	Array where string will be seeked
<i>str</i>	String that will be seeked

Returns

NULL is string was not found, pointer to string if string was found

Precondition

Array was inicialized

3.3.4.7 int array_string_init (array_string * array)

Initalize new array of strings

Parameters

<i>array</i>	array that will be initialized
--------------	--------------------------------

Returns

0 in case of succes, 1 in case of error in memory allocation

Precondition

input pointer points to allocated space
Array was inicialized

3.3.4.8 int array_string_insert (array_string * array, const char * str)

Make deep copy of string and insert copy into array

Parameters

<i>array</i>	array where string will be inserted
<i>str</i>	string that will be copied

Returns

0 in case of succes, 1 in case of error while allocating memory

Precondition

Array was inicialized

3.3.4.9 void stack_htab_destroy (stack_htab * stack)

Free all memory allocated by stack

Parameters

<i>stack</i>	Stack that shall be freed
--------------	---------------------------

Precondition

Stack was inicialized

3.3.4.10 htab_t* stack_htab_get_first (stack_htab * stack)

Return item that is at the bottom of stack

Parameters

<i>stack</i>	Stack where item is stored
--------------	----------------------------

Returns

Item that is stored on the bottom, NULL if stack is empty

3.3.4.11 htab_t* stack_htab_get_item (stack_htab * stack, unsigned backtrack)

Retrive specific item from stack

Parameters

<i>stack</i>	Stack with items
<i>bactrack</i>	How far from top item is stored

Returns

Pointer to specific item or NULL if bactrack is too big

Precondition

Stack was inicialized

3.3.4.12 int stack_htab_init (stack_htab * stack)

Initialize stack

Parameters

<i>stack</i>	Stack for initialization
--------------	--------------------------

Returns

0 on succes, 1 when memory allocation failed

3.3.4.13 htab_t* stack_htab_pop (stack_htab * stack)

Delete item on top

Parameters

<i>stack</i>	Stack where item will be deleted
--------------	----------------------------------

Returns

pointer to popped table on succes if stack is already empty (before pop), return NULL

Precondition

Stack was inicialized

3.3.4.14 int stack_htab_push (stack_htab * stack, htab_t * table)

Push new item into stack. Reallocate itself if stack is full

Parameters

<i>stack</i>	Stack where item will be pushed
<i>table</i>	Pointer to hash table that will be pushed into stack

Returns

0 on succes 1 if reallocation failed (memory will not be freed)

Precondition

Stack was inicialized

Chapter 4

Class Documentation

4.1 array_htab Struct Reference

```
#include <structures.h>
```

Public Attributes

- unsigned [idx](#)
- [size_t](#) [size](#)
- [htab_t](#) ** [data](#)

4.1.1 Detailed Description

Array of hash tables

4.1.2 Member Data Documentation

4.1.2.1 [htab_t](#) ** [array_htab::data](#)

Array of hash tables

4.1.2.2 [unsigned](#) [array_htab::idx](#)

Index where will be added new hash table

4.1.2.3 [size_t](#) [array_htab::size](#)

Maximum number of items after last allocation

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

- [structures.h](#)

4.2 array_string Struct Reference

```
#include <structures.h>
```

Public Attributes

- unsigned [idx](#)
- [size_t](#) [size](#)
- [char](#) ** [data](#)

4.2.1 Detailed Description

Array of strings

4.2.2 Member Data Documentation

4.2.2.1 [char](#)** [array_string::data](#)

Array of hash tables

4.2.2.2 [unsigned](#) [array_string::idx](#)

Index where will be added new hash table

4.2.2.3 [size_t](#) [array_string::size](#)

Maximum number of items after last allocation

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

- [structures.h](#)

4.3 [mem_item_t](#) Struct Reference

```
#include <garbage_collector.h>
```

Public Attributes

- [void](#) * [ptr](#)
- [struct](#) [mem_item_t](#) * [next](#)

4.3.1 Detailed Description

Item that holds one pointer to allocated memory

4.3.2 Member Data Documentation

4.3.2.1 [struct](#) [mem_item_t](#)* [mem_item_t::next](#)

Pointer to nex item

4.3.2.2 void* mem_item_t::ptr

Pointer to allocated memory

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

- garbage_collector.h

4.4 mem_list_t Struct Reference

```
#include <garbage_collector.h>
```

Public Attributes

- struct [mem_item_t](#) * [first](#)
- struct [mem_item_t](#) * [last](#)

4.4.1 Detailed Description

List of items, that holds allocated memory

4.4.2 Member Data Documentation

4.4.2.1 struct mem_item_t* mem_list_t::first

Pointer to first item

4.4.2.2 struct mem_item_t* mem_list_t::last

Pointer to last item

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

- garbage_collector.h

4.5 stack_htab Struct Reference

```
#include <structures.h>
```

Public Attributes

- int [top](#)
- size_t [size](#)
- htab_t ** [data](#)

4.5.1 Detailed Description

Stack of hash tables

4.5.2 Member Data Documentation

4.5.2.1 `htab_t** stack_htab::data`

Array of hash tables

4.5.2.2 `size_t stack_htab::size`

Maximum number of items after last allocation

4.5.2.3 `int stack_htab::top`

Index of item on top of stack

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

- `structures.h`

4.6 token Struct Reference

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

Public Attributes

- `int id`
- `void * ptr`

4.6.1 Detailed Description

Structure that represents token

4.6.2 Member Data Documentation

4.6.2.1 `int token::id`

Id of token (Keyword, numeric constant, operator, ...)

4.6.2.2 `void* token::ptr`

Pointer into data (value of identifikator, name of identifikator...) or NULL if data are not needed.

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

- `scanner.h`

Index

- ARRAY_HTAB_INIT_SIZE
 - Structures, [15](#)
- ARRAY_STRING_INIT_SIZE
 - Structures, [15](#)
- array_htab, [21](#)
 - data, [21](#)
 - idx, [21](#)
 - size, [21](#)
 - Structures, [16](#)
- array_htab_destroy
 - Structures, [16](#)
- array_htab_get_item
 - Structures, [16](#)
- array_htab_init
 - Structures, [16](#)
- array_htab_insert
 - Structures, [18](#)
- array_string, [21](#)
 - data, [22](#)
 - idx, [22](#)
 - size, [22](#)
 - Structures, [16](#)
- array_string_destroy
 - Structures, [18](#)
- array_string_find
 - Structures, [18](#)
- array_string_init
 - Structures, [18](#)
- array_string_insert
 - Structures, [19](#)
- BLOCK_COMMENT
 - Lexical analysis, [9](#)
- bin2dec
 - Lexical analysis, [9](#)
- data
 - array_htab, [21](#)
 - array_string, [22](#)
 - stack_htab, [24](#)
- first
 - mem_list_t, [23](#)
- free_memory
 - Garbage collector, [6](#)
- GARBAGE_COLLECTOR
 - Garbage collector, [6](#)
- Garbage collector, [5](#)
 - free_memory, [6](#)
- GARBAGE_COLLECTOR, [6](#)
 - mem_alloc, [6](#)
 - mem_item_t, [5](#)
 - mem_list_t, [5](#)
 - mem_realloc, [6](#)
- get_token
 - Lexical analysis, [10](#)
- hex2dec_double
 - Lexical analysis, [10](#)
- hex2dec_int
 - Lexical analysis, [10](#)
- id
 - token, [24](#)
- idx
 - array_htab, [21](#)
 - array_string, [22](#)
- is_full_ident
 - Lexical analysis, [10](#)
- is_keyword
 - Lexical analysis, [10](#)
- is_num_literal
 - Lexical analysis, [12](#)
- is_simple_ident
 - Lexical analysis, [12](#)
- is_special_char
 - Lexical analysis, [12](#)
- LINE_COMMENT
 - Lexical analysis, [9](#)
- last
 - mem_list_t, [23](#)
- Lexical analysis, [7](#)
 - BLOCK_COMMENT, [9](#)
 - bin2dec, [9](#)
 - get_token, [10](#)
 - hex2dec_double, [10](#)
 - hex2dec_int, [10](#)
 - is_full_ident, [10](#)
 - is_keyword, [10](#)
 - is_num_literal, [12](#)
 - is_simple_ident, [12](#)
 - is_special_char, [12](#)
 - LINE_COMMENT, [9](#)
 - load_string, [13](#)
 - make_power, [13](#)
 - octal2dec, [13](#)
 - reset_scanner, [8](#)
 - S_AND, [9](#)

- S_ASSIGNMENT, 9
- S_BOOLEAN, 8
- S_BREAK, 8
- S_CLASS, 8
- S_COMMA, 9
- S_CONTINUE, 8
- S_DIV, 9
- S_DO, 8
- S_DOUBLE, 8
- S_ELSE, 8
- S_EOF, 9
- S_EQUAL, 9
- S_FALSE, 8
- S_FOR, 8
- S_FULL_IDENT, 9
- S_GREATER, 9
- S_GREATER_EQUAL, 9
- S_IF, 8
- S_INT, 8
- S_LEFT_BRACE, 9
- S_LEFT_PARE, 9
- S_LESS, 9
- S_LESS_EQUAL, 9
- S_MINUS, 9
- S_MUL, 9
- S_NOT_EQUAL, 9
- S_OR, 9
- S_PLUS, 9
- S_RETURN, 8
- S_RIGHT_BRACE, 9
- S_RIGHT_PARE, 9
- S_SEMICOMMA, 9
- S_SIMPLE_IDENT, 9
- S_SIZE, 8
- S_STATIC, 9
- S_STRING, 9
- S_TRUE, 9
- S_VOID, 9
- S_WHILE, 9
- SPEC_CHAR_FSEEK, 8
- skip_comment, 14
- str2num, 14
- TYPE_BOOLEAN, 9
- TYPE_DOUBLE, 9
- TYPE_DOUBLE_HEX, 9
- TYPE_INT, 9
- TYPE_INT_BIN, 9
- TYPE_INT_HEX, 9
- TYPE_INT_OCTAL, 9
- TYPE_STRING, 9
- token, 8
- load_string
 - Lexical analysis, 13
- make_power
 - Lexical analysis, 13
- mem_alloc
 - Garbage collector, 6
- mem_item_t, 22
 - Garbage collector, 5
 - next, 22
 - ptr, 22
- mem_list_t, 23
 - first, 23
 - Garbage collector, 5
 - last, 23
- mem_realloc
 - Garbage collector, 6
- next
 - mem_item_t, 22
- octal2dec
 - Lexical analysis, 13
- ptr
 - mem_item_t, 22
 - token, 24
- reset_scanner
 - Lexical analysis, 8
- S_AND
 - Lexical analysis, 9
- S_ASSIGNMENT
 - Lexical analysis, 9
- S_BOOLEAN
 - Lexical analysis, 8
- S_BREAK
 - Lexical analysis, 8
- S_CLASS
 - Lexical analysis, 8
- S_COMMA
 - Lexical analysis, 9
- S_CONTINUE
 - Lexical analysis, 8
- S_DIV
 - Lexical analysis, 9
- S_DO
 - Lexical analysis, 8
- S_DOUBLE
 - Lexical analysis, 8
- S_ELSE
 - Lexical analysis, 8
- S_EOF
 - Lexical analysis, 9
- S_EQUAL
 - Lexical analysis, 9
- S_FALSE
 - Lexical analysis, 8
- S_FOR
 - Lexical analysis, 8
- S_FULL_IDENT
 - Lexical analysis, 9
- S_GREATER
 - Lexical analysis, 9
- S_GREATER_EQUAL
 - Lexical analysis, 9

- S_IF
 - Lexical analysis, [8](#)
- S_INT
 - Lexical analysis, [8](#)
- S_LEFT_BRACE
 - Lexical analysis, [9](#)
- S_LEFT_PARE
 - Lexical analysis, [9](#)
- S_LESS
 - Lexical analysis, [9](#)
- S_LESS_EQUAL
 - Lexical analysis, [9](#)
- S_MINUS
 - Lexical analysis, [9](#)
- S_MUL
 - Lexical analysis, [9](#)
- S_NOT_EQUAL
 - Lexical analysis, [9](#)
- S_OR
 - Lexical analysis, [9](#)
- S_PLUS
 - Lexical analysis, [9](#)
- S_RETURN
 - Lexical analysis, [8](#)
- S_RIGHT_BRACE
 - Lexical analysis, [9](#)
- S_RIGHT_PARE
 - Lexical analysis, [9](#)
- S_SEMICOMMA
 - Lexical analysis, [9](#)
- S_SIMPLE_IDENT
 - Lexical analysis, [9](#)
- S_SIZE
 - Lexical analysis, [8](#)
- S_STATIC
 - Lexical analysis, [9](#)
- S_STRING
 - Lexical analysis, [9](#)
- S_TRUE
 - Lexical analysis, [9](#)
- S_VOID
 - Lexical analysis, [9](#)
- S_WHILE
 - Lexical analysis, [9](#)
- SPEC_CHAR_FSEEK
 - Lexical analysis, [8](#)
- STACK_HTAB_INIT_SIZE
 - Structures, [16](#)
- size
 - array_htab, [21](#)
 - array_string, [22](#)
 - stack_htab, [24](#)
- skip_comment
 - Lexical analysis, [14](#)
- stack_htab, [23](#)
 - data, [24](#)
 - size, [24](#)
 - Structures, [16](#)
 - top, [24](#)
- stack_htab_destroy
 - Structures, [19](#)
- stack_htab_get_first
 - Structures, [19](#)
- stack_htab_get_item
 - Structures, [19](#)
- stack_htab_init
 - Structures, [20](#)
- stack_htab_pop
 - Structures, [20](#)
- stack_htab_push
 - Structures, [20](#)
- str2num
 - Lexical analysis, [14](#)
- Structures, [15](#)
 - ARRAY_HTAB_INIT_SIZE, [15](#)
 - ARRAY_STRING_INIT_SIZE, [15](#)
 - array_htab, [16](#)
 - array_htab_destroy, [16](#)
 - array_htab_get_item, [16](#)
 - array_htab_init, [16](#)
 - array_htab_insert, [18](#)
 - array_string, [16](#)
 - array_string_destroy, [18](#)
 - array_string_find, [18](#)
 - array_string_init, [18](#)
 - array_string_insert, [19](#)
 - STACK_HTAB_INIT_SIZE, [16](#)
 - stack_htab, [16](#)
 - stack_htab_destroy, [19](#)
 - stack_htab_get_first, [19](#)
 - stack_htab_get_item, [19](#)
 - stack_htab_init, [20](#)
 - stack_htab_pop, [20](#)
 - stack_htab_push, [20](#)
- TYPE_BOOLEAN
 - Lexical analysis, [9](#)
- TYPE_DOUBLE
 - Lexical analysis, [9](#)
- TYPE_DOUBLE_HEX
 - Lexical analysis, [9](#)
- TYPE_INT
 - Lexical analysis, [9](#)
- TYPE_INT_BIN
 - Lexical analysis, [9](#)
- TYPE_INT_HEX
 - Lexical analysis, [9](#)
- TYPE_INT_OCTAL
 - Lexical analysis, [9](#)
- TYPE_STRING
 - Lexical analysis, [9](#)
- token, [24](#)
 - id, [24](#)
 - Lexical analysis, [8](#)
 - ptr, [24](#)
- top
 - stack_htab, [24](#)