

My Project

Generated by Doxygen 1.8.9.1

Sun Dec 11 2016 21:45:38

Contents

1	Module Index	1
1.1	Modules	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Module Documentation	7
4.1	Embedded functions	7
4.1.1	Detailed Description	7
4.1.2	Function Documentation	7
4.1.2.1	readDouble	7
4.1.2.2	readInt	7
4.1.2.3	readString	8
4.2	Expression processing	9
4.2.1	Detailed Description	10
4.2.2	Macro Definition Documentation	10
4.2.2.1	ERR_WARNING	10
4.2.2.2	FATAL_ERROR	10
4.2.2.3	STRDUP	10
4.2.3	Function Documentation	10
4.2.3.1	expr_analyze	10
4.2.3.2	operator_priority	11
4.2.3.3	print_token	11
4.2.3.4	print_token_array	12
4.2.3.5	stack_expression_destroy	12
4.2.3.6	stack_expression_empty	12
4.2.3.7	stack_expression_full	12
4.2.3.8	stack_expression_init	13
4.2.3.9	stack_expression_pop	13

4.2.3.10	stack_expression_push	14
4.2.3.11	stack_expression_top	15
4.2.3.12	type_name_conversion	15
4.2.3.13	type_priority	15
4.3	Garbage collector	17
4.3.1	Detailed Description	17
4.3.2	Typedef Documentation	17
4.3.2.1	mem_item_t	17
4.3.2.2	mem_list_t	17
4.3.3	Function Documentation	17
4.3.3.1	free_memory	17
4.3.3.2	mem_alloc	18
4.3.3.3	mem_realloc	19
4.3.4	Variable Documentation	19
4.3.4.1	GARBAGE_COLLECTOR	19
4.4	Functions for string processing	20
4.4.1	Detailed Description	20
4.4.2	Macro Definition Documentation	20
4.4.2.1	ALPHABET_ARRAY	20
4.4.2.2	MAX	20
4.4.3	Function Documentation	20
4.4.3.1	computeJumps	20
4.4.3.2	find	20
4.4.3.3	length	21
4.4.3.4	shellsort	21
4.4.3.5	substring	21
4.5	Stack of integers	23
4.5.1	Detailed Description	23
4.5.2	Function Documentation	23
4.5.2.1	stack_int_clean	23
4.5.2.2	stack_int_create	23
4.5.2.3	stack_int_destroy	24
4.5.2.4	stack_int_is_empty	24
4.5.2.5	stack_int_is_full	24
4.5.2.6	stack_int_pop	24
4.5.2.7	stack_int_push	25
4.5.2.8	stack_int_top	25
4.6	Hash table	26
4.6.1	Detailed Description	26
4.6.2	Typedef Documentation	26

4.6.2.1	htab_item	26
4.6.2.2	htab_t	26
4.6.3	Function Documentation	26
4.6.3.1	htab_clear_items	26
4.6.3.2	htab_copy	26
4.6.3.3	htab_find_item	27
4.6.3.4	htab_find_item_by_argument_index	27
4.6.3.5	htab_free_all	27
4.6.3.6	htab_init	27
4.6.3.7	htab_init2	28
4.6.3.8	htab_insert_item	29
4.7	Interpret processing	30
4.7.1	Detailed Description	30
4.7.2	Enumeration Type Documentation	30
4.7.2.1	anonymous enum	30
4.7.3	Function Documentation	30
4.7.3.1	Add_Instr	30
4.7.3.2	Conc_Str	31
4.7.3.3	do_expression	31
4.7.3.4	DoubleToString	31
4.7.3.5	inter	32
4.7.3.6	inter_arm_op	32
4.7.3.7	inter_bool_op	32
4.7.3.8	inter_plus	33
4.7.3.9	IntToString	33
4.7.3.10	stack_htab_find_htab_item	33
4.8	Lexical analysis	35
4.8.1	Detailed Description	36
4.8.2	Macro Definition Documentation	36
4.8.2.1	reset_scanner	36
4.8.2.2	S_SIZE	36
4.8.2.3	SPEC_CHAR_FSEEK	36
4.8.3	Typedef Documentation	36
4.8.3.1	token	36
4.8.4	Enumeration Type Documentation	36
4.8.4.1	anonymous enum	36
4.8.5	Function Documentation	37
4.8.5.1	bin2dec	37
4.8.5.2	get_token	38
4.8.5.3	hex2dec_double	38

4.8.5.4	hex2dec_int	38
4.8.5.5	is_full_ident	38
4.8.5.6	is_keyword	39
4.8.5.7	is_num_literal	40
4.8.5.8	is_simple_ident	40
4.8.5.9	is_special_char	40
4.8.5.10	load_string	41
4.8.5.11	make_power	41
4.8.5.12	octal2dec	41
4.8.5.13	repair_num	42
4.8.5.14	skip_comment	42
4.8.5.15	str2num	42
4.9	Structures	43
4.9.1	Detailed Description	44
4.9.2	Macro Definition Documentation	44
4.9.2.1	ARRAY_HTAB_INIT_SIZE	44
4.9.2.2	ARRAY_STRING_INIT_SIZE	44
4.9.2.3	STACK_HTAB_INIT_SIZE	44
4.9.2.4	STACK_INSTR_INIT_SIZE	44
4.9.3	Typedef Documentation	44
4.9.3.1	array_htab	44
4.9.3.2	array_string	44
4.9.3.3	stack_htab	44
4.9.3.4	stack_instr	44
4.9.4	Function Documentation	44
4.9.4.1	array_htab_destroy	45
4.9.4.2	array_htab_get_item	46
4.9.4.3	array_htab_init	46
4.9.4.4	array_htab_insert	46
4.9.4.5	array_string_destroy	46
4.9.4.6	array_string_find	47
4.9.4.7	array_string_init	47
4.9.4.8	array_string_insert	47
4.9.4.9	stack_htab_destroy	48
4.9.4.10	stack_htab_get_first	49
4.9.4.11	stack_htab_get_item	49
4.9.4.12	stack_htab_init	49
4.9.4.13	stack_htab_pop	49
4.9.4.14	stack_htab_push	50

5	Class Documentation	51
5.1	array_htab Struct Reference	51
5.1.1	Detailed Description	51
5.1.2	Member Data Documentation	51
5.1.2.1	data	51
5.1.2.2	idx	51
5.1.2.3	size	51
5.2	array_string Struct Reference	51
5.2.1	Detailed Description	52
5.2.2	Member Data Documentation	52
5.2.2.1	data	52
5.2.2.2	idx	52
5.2.2.3	size	52
5.3	htab_item Struct Reference	52
5.3.1	Detailed Description	52
5.3.2	Member Data Documentation	53
5.3.2.1	argument_index	53
5.3.2.2	data	53
5.3.2.3	data_type	53
5.3.2.4	func_or_var	53
5.3.2.5	initialized	53
5.3.2.6	instruction_tape	53
5.3.2.7	key	53
5.3.2.8	local_table	53
5.3.2.9	next_item	53
5.3.2.10	number_of_arguments	53
5.4	htab_t Struct Reference	53
5.4.1	Detailed Description	54
5.4.2	Member Data Documentation	54
5.4.2.1	hash_fun_ptr	54
5.4.2.2	htab_size	54
5.4.2.3	number_items	54
5.4.2.4	ptr	54
5.5	I_Instr Struct Reference	54
5.6	Instr_List Struct Reference	55
5.7	mem_item_t Struct Reference	55
5.7.1	Detailed Description	55
5.7.2	Member Data Documentation	55
5.7.2.1	next	55
5.7.2.2	ptr	55

5.8	mem_list_t Struct Reference	55
5.8.1	Detailed Description	55
5.8.2	Member Data Documentation	56
5.8.2.1	first	56
5.8.2.2	last	56
5.9	stack_expression Struct Reference	56
5.9.1	Detailed Description	56
5.10	stack_htab Struct Reference	56
5.10.1	Detailed Description	56
5.10.2	Member Data Documentation	57
5.10.2.1	data	57
5.10.2.2	size	57
5.10.2.3	top	57
5.11	stack_instr Struct Reference	57
5.11.1	Detailed Description	57
5.11.2	Member Data Documentation	57
5.11.2.1	data	57
5.11.2.2	size	57
5.11.2.3	top	57
5.12	t_stack_int Struct Reference	58
5.12.1	Detailed Description	58
5.12.2	Member Data Documentation	58
5.12.2.1	data	58
5.12.2.2	size	58
5.12.2.3	top	58
5.13	token Struct Reference	58
5.13.1	Detailed Description	58
5.13.2	Member Data Documentation	59
5.13.2.1	id	59
5.13.2.2	ptr	59
6	File Documentation	61
6.1	embedded_functions.h File Reference	61
6.1.1	Detailed Description	61
6.2	expression.h File Reference	61
6.2.1	Detailed Description	63
6.3	interpret.h File Reference	63
6.3.1	Detailed Description	64
	Index	65

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Embedded functions	7
Expression processing	9
Garbage collector	17
Functions for string processing	20
Stack of integers	23
Hash table	26
Interpret processing	30
Lexical analysis	35
Structures	43

Chapter 2

Class Index

2.1 Class List

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

array_htab	51
array_string	51
htab_item	52
htab_t	53
I_Instr	54
Instr_List	55
mem_item_t	55
mem_list_t	55
stack_expresion	
Structure for stack of tokens	56
stack_htab	56
stack_instr	57
t_stack_int	
Structure for stack of integers	58
token	58

Chapter 3

File Index

3.1 File List

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

embedded_functions.h	
Documentation for embeded functions	61
expression.h	
Documentation for expression processing	61
garbage_collector.h	??
ial.h	??
interpret.h	
Documentation for interpret processing	63
parser.h	??
scanner.h	??
structures.h	??

Chapter 4

Module Documentation

4.1 Embeded functions

Embeded functions of the language IFJ16.

Functions

- char * `readString` ()
Function reads a string from STDIN.
- int `readInt` ()
Function reads an integer value from STDIN.
- double `readDouble` ()
Function reads a number in double format from STDIN.
- void `print` (char *string)
Function prints string to STDOUT.

4.1.1 Detailed Description

Embeded functions of the language IFJ16.

4.1.2 Function Documentation

4.1.2.1 double readDouble ()

Function reads a number in double format from STDIN.

Returns

number in double format read from STDIN.

4.1.2.2 int readInt ()

Function reads an integer value from STDIN.

Returns

integer value read from STDIN.

4.1.2.3 `char* readString ()`

Function reads a string from STDIN.

Returns

array of chars (string) read from STDIN.

4.2 Expression processing

Precedence analysis and expression processing of the language IFJ16.

Classes

- struct [stack_expression](#)
Structure for stack of tokens.

Macros

- #define [ERR_WARNING](#) 0
- #define [ERR_LEXICAL_ANALYSIS](#) 1
- #define [ERR_SYNTACTIC_ANALYSIS](#) 2
- #define [ERR_SEM_NDEF_REDEF](#) 3
- #define [ERR_SEM_COMPATIBILITY](#) 4
- #define [ERR_SEM_OTHERS](#) 6
- #define [ERR_INPUT_NUMBER](#) 7
- #define [ERR_UNINIALIZED_VAR](#) 8
- #define [ERR_DIVISION_ZERO](#) 9
- #define [ERR_OTHERS](#) 10
- #define [ERR_INTERN_FAULT](#) 99
- #define [FATAL_ERROR](#)(message, error_code)
- #define [STRDUP](#)(l, s)

Functions

- int [expr_analyze](#) (token t_in, token *t_out, char *class_name, int error_6_flag, token **postfix_token_array, int *token_count, int *expr_data_type, [htab_t](#) *global_table, [htab_t](#) *local_table,...)
Function analyzes precedence and converts expression to postfix format.
- int [stack_expression_init](#) (struct [stack_expression](#) *s, int size)
Function initializes a stack, allocates required memory and sets its variables.
- int [stack_expression_destroy](#) (struct [stack_expression](#) *s)
Function destroys a stack, frees its memory and sets its variables.
- int [stack_expression_empty](#) (const struct [stack_expression](#) *s)
Function checks whether stack is empty.
- int [stack_expression_full](#) (const struct [stack_expression](#) *s)
Function checks whether stack is full.
- int [stack_expression_top](#) (struct [stack_expression](#) *s, token *t)
Function gives back top element from the stack.
- int [stack_expression_pop](#) (struct [stack_expression](#) *s, token *t)
Function pops and gives back top element from the stack.
- int [stack_expression_push](#) (struct [stack_expression](#) *s, token t)
Function pushes given element to the stack.
- int [operator_priority](#) (int op)
Function tells the priority of a given operator.
- int [type_priority](#) (int type)
Function tells the priority of a given data type, which is later used for determining data type of the whole expression.
- int [type_name_conversion](#) (int type)
Function converts names of the given data type, so it could be understood by the function [expr_analyze\(\)](#).
- void [print_token](#) (token t, int id_flag)

Function prints token value (and its id) to STDERR. This function is only used for debugging.

- void `print_token_array` (token *arr, int id_flag)

Function prints array of tokens (and their ids) to STDERR. This function is only used for debugging.

4.2.1 Detailed Description

Precedence analysis and expression processing of the language IFJ16.

4.2.2 Macro Definition Documentation

4.2.2.1 #define ERR_WARNING 0

Error constants

4.2.2.2 #define FATAL_ERROR(message, error_code)

Value:

```
do {      if (mal[0] != NULL) free(mal[0]); if (mal[1] != NULL) free(mal[1]); \
                                                fputs((message), stderr);
\
                                                return (error_code); } while(0)
```

Macro frees allocated memory, prints error message and returns with given error code

4.2.2.3 #define STRDUP(l, s)

Value:

```
do {      char *tmp = (char *)mem_alloc( sizeof(char) * ( strlen((char *) (s)) + 1 ) );
\
          if (tmp == NULL)
\
          FATAL_ERROR("EXPRESSION: Memory could not be allocated.
11\n", ERR_INTERN_FAULT);
          else
\
          {
\
          strcpy(tmp, (char *) (s));
\
          (l) = tmp;
\
          }
\
          } while(0)
```

Macro that duuplicates given string

4.2.3 Function Documentation

4.2.3.1 int expr_analyze(token t_in, token * t_out, char * class_name, int error_6_flag, token ** postfix_token_array, int * token_count, int * expr_data_type, htab_t * global_table, htab_t * local_table, ...)

Function analyzes precedence and converts expression to postfix format.

Parameters

in	<i>t_in</i>	first token in an expression.
out	<i>t_out</i>	last token red by the function, which is not contained in a final expression.
in	<i>class_name</i>	name of a class, which contains processed expression.
in	<i>error_6_flag</i>	flag which determines whether error number 6 could be returned.
out	<i>postfix_token_↔ array</i>	final postfix expression.
out	<i>token_count</i>	number of tokens in a retuned expression array.
out	<i>expr_data_type</i>	data type of the retuned expression.
in	<i>global_table</i>	global table of symbols.
in	<i>local_table</i>	local table of symbols.

Precondition

```

class_name!=NULL
global_table!=NULL
local_table!=NULL
error_6_flag==1 || error_6_flag==0

```

Postcondition

```

postfix_token_array!=NULL
t_out!=NULL
token_count!=0
postfix_token_array[token_count-1]==END_EXPR

```

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.2 int operator_priority (int op)

Function tells the priority of a given operator.

Parameters

in	<i>op</i>	operator symbol.
----	-----------	------------------

Precondition

```

op>=0

```

Returns

integer value which tells the priority of a given operator.

4.2.3.3 void print_token (token t, int id_flag)

Function prints token value (and its id) to STDERR. This function is only used for debugging.

Parameters

<i>in</i>	<i>t</i>	token which should be printed.
<i>in</i>	<i>id_flag</i>	value which determines whether the id of a token should be printed as well.

Precondition

`id_flag == 0 || id_flag == 1`

4.2.3.4 void print_token_array (token * arr, int id_flag)

Function prints array of tokens (and their ids) to STDERR. This function is only used for debugging.

Parameters

<i>in</i>	<i>arr</i>	array of tokens which should be printed.
<i>in</i>	<i>id_flag</i>	value which determines whether the id of a tokens should be printed as well.

Precondition

`id_flag == 0 || id_flag == 1`

4.2.3.5 int stack_expression_destroy (struct stack_expresion * s)

Function destroys a stack, frees its memory and sets its variables.

Parameters

<i>in, out</i>	<i>s</i>	pointer to a stack.
----------------	----------	---------------------

Precondition

`s!=NULL`

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.6 int stack_expression_empty (const struct stack_expresion * s)

Function checks whether stack is empty.

Parameters

<i>in</i>	<i>s</i>	pointer to a stack.
-----------	----------	---------------------

Precondition

`s!=NULL`

Returns

integer value which tells, if the stack is empty, 0 -> not empty, !=0 -> empty.

4.2.3.7 int stack_expression_full (const struct stack_expresion * s)

Function checks whether stack is full.

Parameters

<i>in</i>	<i>s</i>	pointer to a stack.
-----------	----------	---------------------

Precondition

s!=NULL

Returns

integer value which tells, if the stack is full, 0 -> not full, !=0 -> full.

4.2.3.8 int stack_expression_init (struct stack_expression * *s*, int *size*)

Function initializes a stack, allocates required memory and sets its variables.

Parameters

<i>in, out</i>	<i>s</i>	pointer to a stack.
<i>in</i>	<i>size</i>	max. number of tokens which stack could contain.

Precondition

s!=NULL

size!=0

Postcondition

s!=NULL

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.9 int stack_expression_pop (struct stack_expression * *s*, token * *t*)

Function pops and gives back top element from the stack.

Parameters

<i>in</i>	<i>s</i>	pointer to a stack.
<i>out</i>	<i>t</i>	top element from the stack.

Precondition

s!=NULL

t!=NULL

s is not empty

Postcondition

t!=NULL

s->size == *s*->size-1

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.10 `int stack_expression_push (struct stack_expresion * s, token t)`

Function pushes given element to the stack.

Parameters

in	<i>s</i>	pointer to a stack.
out	<i>t</i>	token which should be pushed to the stack.

Precondition

s!=NULL
s is not full

Postcondition

s->size == *s*->size+1
s->arr[*s*->size] == *t*

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.11 int stack_expression_top (struct stack_expression * *s*, token * *t*)

Function gives back top element from the stack.

Parameters

in	<i>s</i>	pointer to a stack.
out	<i>t</i>	top element from the stack.

Precondition

s!=NULL
t!=NULL
s is not empty

Postcondition

t!=NULL

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.2.3.12 int type_name_conversion (int *type*)

Function converts names of the given data type, so it could be understood by the function [expr_analyze\(\)](#).

Parameters

in	<i>type</i>	data type symbol.
----	-------------	-------------------

Returns

converted integer value which tells the what is the given data type.

4.2.3.13 int type_priority (int *type*)

Function tells the priority of a given data type, which is later used for determining data type of the whole expression.

Parameters

<i>in</i>	<i>type</i>	data type symbol.
-----------	-------------	-------------------

Precondition

$type \geq 0$

Returns

integer value which tells the priority of a given data type.

4.3 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](#)

4.3.1 Detailed Description

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

4.3.2 Typedef Documentation

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

Item that holds one pointer to allocated memory

4.3.2.2 typedef struct [mem_list_t](#) [mem_list_t](#)

List of items, that holds allocated memory

4.3.3 Function Documentation

4.3.3.1 void [free_memory](#) ()

Free all memory allocated with this module

Precondition

Function [mem_list_t_init](#) was called before

4.3.3.2 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

4.3.3.3 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

4.3.4 Variable Documentation**4.3.4.1 GARBAGE_COLLECTOR**

Global list of items that holds allocated memory

Initialize

4.4 Functions for string processing

Functions for string processing.

Macros

- `#define MAX(a, b) ((a) > (b) ? (a) : (b))`
- `#define ALPHABET_ARRAY 256`

Functions

- `int length (char *string)`
- `char * substring (char *s, int i, int n)`
- `char * shellsort (char *str)`
- `int find (char *s, char *search)`
- `void computeJumps (char *string, int badchar[ALPHABET_ARRAY])`

4.4.1 Detailed Description

Functions for string processing.

Authors: Miroslava Misova, Nemanja Vasiljevic, Jiri Matejka, Sava Nedeljkovic School: VUT FIT, BRNO Project: Interpret for IFJ16 gcc version: 5.4.0 (ubuntu 16.04.2)

4.4.2 Macro Definition Documentation

4.4.2.1 `#define ALPHABET_ARRAY 256`

Represents number of chars in alphabet

4.4.2.2 `#define MAX(a, b) ((a) > (b) ? (a) : (b))`

Finds maximum

4.4.3 Function Documentation

4.4.3.1 `void computeJumps (char * string, int badchar[ALPHABET_ARRAY])`

Preprocessing for find. Fill the bad character array by given pattern

Parameters

<i>string</i>	pattern
<i>badchar[ALPHABET_ARRAY]</i>	alphabet array of integers

4.4.3.2 `int find (char * s, char * search)`

Finds substring in given string. Function uses Boyer-Moore string algorithm.

Parameters

<i>s</i>	text
<i>search</i>	patent

Returns

index where substring is found, -1 if substring is not found.

4.4.3.3 int length (char * *string*)

Function returns length of 0 terminated string.

Parameters

<i>string</i>	array of chars
---------------	----------------

Precondition

string!=NULL

Returns

integer value, length of string.

4.4.3.4 char* shellsort (char * *str*)

Returns sorted array of a chars. Function uses Shell sort algorithm. Function allocates new memory for a return array of chars.

Parameters

<i>str</i>	input string
------------	--------------

Precondition

s != NULL

Returns

sorted array of a chars.

4.4.3.5 char* substring (char * *s*, int *i*, int *n*)

Returns substring of a given length, beginning at a given position of a given string Function allocates new memory for a substring

Parameters

<i>s</i>	input string
<i>i</i>	index where the substring begins

n	length of a substring
-----	-----------------------

Precondition

$s \neq \text{NULL}$
 $n > 0$
 $i \geq 0$
 $\text{strlen}(s) \geq n+i$

Returns

array of chars containing found substring

4.5 Stack of integers

Stack of integers.

Classes

- struct [t_stack_int](#)
Structure for stack of integers.

Typedefs

- typedef struct [t_stack_int](#) **stack_int_t**

Functions

- int [stack_int_create](#) (struct [t_stack_int](#) *stack, int n)
Function initializes a stack, allocates required memory and sets its variables.
- void [stack_int_destroy](#) (struct [t_stack_int](#) *stack)
Function destroys a stack, frees its memory and sets its variables.
- int [stack_int_push](#) (struct [t_stack_int](#) *stack, int num,...)
- int [stack_int_pop](#) (struct [t_stack_int](#) *stack)
- int [stack_int_top](#) (struct [t_stack_int](#) *stack, int *var)
- int [stack_int_clean](#) (struct [t_stack_int](#) *stack, int n)
- int [stack_int_is_empty](#) (struct [t_stack_int](#) *stack)
- int [stack_int_is_full](#) (struct [t_stack_int](#) *stack)

4.5.1 Detailed Description

Stack of integers.

4.5.2 Function Documentation

4.5.2.1 int stack_int_clean (struct [t_stack_int](#) * *stack*, int *n*)

Decrements stack pointer by given number of elements.

Parameters

<i>stack</i>	pointer to a stack.
<i>n</i>	number of elements which should be removed from stack.

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.5.2.2 int stack_int_create (struct [t_stack_int](#) * *stack*, int *n*)

Function initializes a stack, allocates required memory and sets its variables.

Parameters

<i>in, out</i>	<i>stack</i>	pointer to a stack.
<i>in</i>	<i>n</i>	max. number of elements which stack could contain.

Precondition

stack!=NULL
n!=0

Postcondition

stack!=NULL

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.5.2.3 void stack_int_destroy (struct t_stack_int * *stack*)

Function destroys a stack, frees its memory and sets its variables.

Parameters

<i>in, out</i>	<i>s</i>	pointer to a stack.
----------------	----------	---------------------

Precondition

stack!=NULL

4.5.2.4 int stack_int_is_empty (struct t_stack_int * *stack*)

Function checks whether stack is empty.

Parameters

<i>stack</i>	pointer to a stack.
--------------	---------------------

Returns

integer value which tells, if the stack is empty, 0 -> not empty, !=0 -> empty.

4.5.2.5 int stack_int_is_full (struct t_stack_int * *stack*)

Function checks whether stack is full.

Parameters

<i>stack</i>	pointer to a stack.
--------------	---------------------

Returns

integer value which tells, if the stack is full, 0 -> not full, !=0 -> full.

4.5.2.6 int stack_int_pop (struct t_stack_int * *stack*)

Function pops and gives back top element from the stack.

Parameters

<i>stack</i>	pointer to a stack.
--------------	---------------------

Returns

integer value, top element from the stack.

4.5.2.7 `int stack_int_push (struct t_stack_int * stack, int num, ...)`

Function pushes given elements to the stack.

Parameters

<i>stack</i>	pointer to a stack.
<i>num</i>	number of elements which should be pushed to the stack
<i>VARARGS</i>	integer values which should be pushed to the stack

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.5.2.8 `int stack_int_top (struct t_stack_int * stack, int * var)`

Function gives back top element from the stack.

Parameters

<i>stack</i>	pointer to a stack.
<i>var</i>	top element from the stack.

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.6 Hash table

Group of structure hash table where is stored variables, functions and classes and functions that operate upon it.

Classes

- struct [htab_item](#)
- struct [htab_t](#)

Typedefs

- typedef struct [htab_item](#) [htab_item](#)
- typedef struct [htab_t](#) [htab_t](#)

Functions

- [htab_t](#) * [htab_init](#) (unsigned size)
- [htab_t](#) * [htab_init2](#) (unsigned size, unsigned(*hash_fun)(const char *str, unsigned htab_size))
- [htab_item](#) * [htab_find_item](#) ([htab_t](#) *T, const char *key)
- [htab_item](#) * [htab_insert_item](#) ([htab_t](#) *T, const char *key)
- [htab_t](#) * [htab_copy](#) ([htab_t](#) *T)
- [htab_item](#) * [htab_find_item_by_argument_index](#) ([htab_t](#) *T, int index)
- void [htab_clear_items](#) ([htab_t](#) *T)
- void [htab_free_all](#) ([htab_t](#) *T)

4.6.1 Detailed Description

Group of structure hash table where is stored variables, functions and classes and functions that operate upon it.

4.6.2 Typedef Documentation

4.6.2.1 typedef struct [htab_item](#) [htab_item](#)

Abstract data type that represents item of hash table

4.6.2.2 typedef struct [htab_t](#) [htab_t](#)

Abstract data type that represents hash table

4.6.3 Function Documentation

4.6.3.1 void [htab_clear_items](#) ([htab_t](#) * *T*)

Free memory allocated when inserting items but since we use garbage collector, it only sets pointers to NULL

Parameters

<i>T</i>	Table where items will be freed
----------	---------------------------------

4.6.3.2 [htab_t](#)* [htab_copy](#) ([htab_t](#) * *T*)

Makes copy of table (without pointers)

Parameters

<i>T</i>	Table that will be copied
----------	---------------------------

Returns

Copy of table

4.6.3.3 `htab_item* htab_find_item (htab_t * T, const char * key)`

Finds item by key in table

Parameters

<i>T</i>	Table where item will be sought
<i>key</i>	Key of searched item

Returns

Pointer to searched item, if item will not be find, returns NULL

4.6.3.4 `htab_item* htab_find_item_by_argument_index (htab_t * T, int index)`

Finds arguments of functions

Parameters

<i>T</i>	Table where function is stored
<i>index</i>	Index of argument

Returns

Item where is argument stored

Precondition

`index >= 0`

4.6.3.5 `void htab_free_all (htab_t * T)`

Free allocated memory but since we use garbage collector, it only sets pointers to NULL

Parameters

<i>T</i>	Table that will be freed
----------	--------------------------

4.6.3.6 `htab_t* htab_init (unsigned size)`

Creates new table with default hash function

Parameters

<i>size</i>	Size of new table
-------------	-------------------

Returns

Pointer to new table

4.6.3.7 `htab_t* htab_init2 (unsigned size, unsigned(*)(const char *str, unsigned htab_size) hash_fun)`

Creates new table with specific hash function

Parameters

<i>size</i>	Size of new table
<i>hash_fun</i>	Pointer to hash function

Returns

Pointer to new table

4.6.3.8 `htab_item* htab_insert_item (htab_t * T, const char * key)`

Creates new item and insert it into table

Parameters

<i>T</i>	Table where item will be inserted
<i>key</i>	Key of new item (key will be copied)

Returns

Pointer to new item

4.7 Interpret processing

Interpret processing instruction tape made for language IFJ16.

Enumerations

- enum {
I_ASSIGNMENT = 100+1, **I_IF**, **I_ELSE**, **I_WHILE**,
I_END, **I_FCE**, **I_RETURN**, **I_PRINT**,
I_ENDIF, **I_ENDWHILE**, **I_ENDEELSE** }

Functions

- int **Add_Instr** (**Instr_List** *L, **I_Instr** *new)
Function add instruction at end of instruction tape.
- token** * **do_expression** (**token** *postfix_array, **stack_htab** *I_Htable, struct **stack_expression** *S, **Instr_List** *L, int void_flag)
Interpret calls this function when instruction has expression. Function process postfix array, manage all arithmetic and bool operation and calls interpret.
- token** * **inter_plus** (**token** a, **token** b)
Function adds tokens data or concatenate tokens data if one of operands is string.
- token** * **inter_arm_op** (**token** tmp1, **token** tmp2, int i)
Function does arithmetic operation between two tokens data.
- token** * **inter_bool_op** (**token** tmp1, **token** tmp2, int i)
Function does boolean operation between two tokens data.
- int **inter** (**Instr_List** *L, **stack_htab** *I_Htable, **token** *return_token, int void_flag)
Function does boolean operation between two tokens data.
- char * **IntToString** (int x)
Function translates integer to string.
- char * **DoubleToString** (double x)
Function translates double to string.
- char * **Conc_Str** (char *s1, char *s2)
Function concatenates two strings.
- htab_item** * **stack_htab_find_htab_item** (**stack_htab** *stack, char *key)
- htab_t** * **stack_htab_get_first** (**stack_htab** *stack)
- void **I_Instr_null_elements** (**I_Instr** *Instruction)

4.7.1 Detailed Description

Interpret processing instruction tape made for language IFJ16.

4.7.2 Enumeration Type Documentation

4.7.2.1 anonymous enum

Enumator for type of instructions

4.7.3 Function Documentation

4.7.3.1 int Add_Instr (Instr_List * L, I_Instr * new)

Function add instruction at end of instruction tape.

Parameters

<i>in, out</i>	<i>L</i>	pointer to instruction tape.
<i>in</i>	<i>new</i>	pointer to instruction.

Precondition

new!=NULL
size!=0

Returns

integer value which tells, how the whole process has been executed, 0 -> no error, -1 -> error.

4.7.3.2 char* Conc_Str (char * *s1*, char * *s2*)

Function concatenates two strings.

Parameters

<i>in</i>	<i>s1</i>	first operand.
<i>in</i>	<i>s2</i>	second operand.

Returns

concatenated string, NULL for error.

4.7.3.3 token* do_expression (token * *postfix_array*, stack_htab * *I_Htable*, struct stack_expression * *S*, Instr_List * *L*, int *void_flag*)

Interpret calls this function when instruction has expression. Function process postfix array, manage all arithmetic and bool operation and calls interpret.

Parameters

<i>in</i>	<i>postfix_array</i>	array of postfix tokens.
<i>in</i>	<i>I_Htable</i>	pointer to stack of tables.
<i>in</i>	<i>S</i>	pointer to stack for tokens.
<i>in</i>	<i>L</i>	pointer to instruction tape.
<i>in</i>	<i>void_flag</i>	flag for void function, 1 represent that interpret is in void function, 0 for non-void function.

Precondition

I_Htable!=NULL
L!=NULL
S!=NULL

Returns

token as result of expression, token->id tells how the whole process has been executed, token->id >= 0 -> no error, <0 or NULL -> error.

4.7.3.4 char* DoubleToString (double *x*)

Function translates double to string.

Parameters

in	x	double that is going to be converted.
----	---	---------------------------------------

Returns

converted string, NULL for error.

4.7.3.5 `int inter (Instr_List * L, stack_htab * l_Htable, token * return_token, int void_flag)`

Function does boolean operation between two tokens data.

Parameters

in	L	pointer to instruction tape.
in	l_Htable	pointer to stack of tables.
out	return_token	is result of function on instruction tape. Used if function is called by indirect recursion from do_expression.
in	void_flag	flag for void function, 1 represent that interpret is in void function, 0 for non-void function.

Precondition

l_Htable!=NULL;
return_token!=NULL;

Returns

integer value which tells, how the whole processed has been executed, 0 -> no error, !=0 -> error.

4.7.3.6 `token* inter_arm_op (token tmp1, token tmp2, int i)`

Function does arithmetic operation between two tokens data.

Parameters

in	tmp1	is first operand.
in	tmp2	is second operand.
in	i	stand for type of arithmetic operation. 1 minus, 2 multiply, 3 div

Precondition

$0 < i < 4$

Returns

token, token->ptr is pointer to result of arithmetic operation of two tokens data. token->id >= 0 -> no error, <0 or NULL -> error.

4.7.3.7 `token* inter_bool_op (token tmp1, token tmp2, int i)`

Function does boolean operation between two tokens data.

Parameters

in	<i>tmp1</i>	is first operand.
in	<i>tmp2</i>	is second operand.

Precondition

$$0 < i < 8$$

Parameters

in	<i>i</i>	stand for type of arithmetic operation. 1-> ==, 2-> !=, 3-> >=, 4-> >, 5-> <=, 6-> <, 7-> !(only for boolean)
----	----------	---

Returns

token, token->ptr is pointer to result of boolean operation of two tokens data. token->id >= 0 -> no error, <0 or NULL -> error.

4.7.3.8 token* inter_plus (token a, token b)

Function adds tokens data or concatenate tokens data if one of operands is string.

Parameters

in	<i>a</i>	is first operand.
in	<i>b</i>	is second operand.

Returns

token, token->ptr is pointer to result of addition / concatenate of two tokens data. token->id >= 0 -> no error, <0 or NULL -> error.

4.7.3.9 char* IntToString (int x)

Function translates integer to string.

Parameters

in	<i>x</i>	integer that is going to be converted.
----	----------	--

Returns

converted string, NULL for error.

4.7.3.10 htab_item* stack_htab_find_htab_item (stack_htab * stack, char * key)

Search for item of local or global hash table in the stack

Parameters

<i>stack</i>	stack where item will be searched
--------------	-----------------------------------

<i>key</i>	name of variable or function which will be searched
------------	---

Returns

pointer to item of hash. table where searched thing is, or NULL if the search was not successful

Precondition

stack has been inicialized

4.8 Lexical analysis

Lexical analyse analyse input source code and check, if it is subset 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_NOT](#), [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 [repair_num](#) (char *str)
- void * [str2num](#) (char *str, int type, int *valid)
- [token](#) [get_token](#) ()

Variables

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

4.8.1 Detailed Description

Lexical analyse analyse input source code and check, if it is subset 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

4.8.2 Macro Definition Documentation

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

Macro that set offset at the beginning of file

4.8.2.2 `#define S_SIZE 32`

Default size for memory allocation

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

Macro that tells how much will be offset returned

4.8.3 Typedef Documentation

4.8.3.1 `typedef struct token token`

Structure that represents token

4.8.4 Enumeration Type Documentation

4.8.4.1 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_NOT 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

4.8.5 Function Documentation

4.8.5.1 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

4.8.5.2 token get_token ()

Retrieve 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 pointer to NULL if string is not needed)

4.8.5.3 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

4.8.5.4 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

4.8.5.5 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

4.8.5.6 `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

4.8.5.7 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

4.8.5.8 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

4.8.5.9 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

4.8.5.10 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 sequence

4.8.5.11 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

4.8.5.12 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
---------------	------------------

4.8.5.13 void repair_num (char * *str*)

Remove '_' from string

Parameters

<i>str</i>	String what will be changed
------------	-----------------------------

4.8.5.14 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

4.8.5.15 void* str2num (char * *str*, int *type*, int * *valid*)

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>valid</i>	Variable that will be set into 0 in case of success, into 1 in case of error while allocating memory, into 2 in case of invalid string or into 3 in case of invalid type

Returns

Pointer into value that is result of conversion

4.9 Structures

Structures is group of structures and functions upon them.

Classes

- struct [I_Instr](#)
- struct [Instr_List](#)
- struct [stack_htab](#)
- struct [array_htab](#)
- struct [array_string](#)
- struct [stack_instr](#)

Macros

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

Typedefs

- typedef struct [I_Instr](#) **I_Instr**
- typedef struct [Instr_List](#) **Instr_List**
- typedef struct [stack_htab](#) **stack_htab**
- typedef struct [array_htab](#) **array_htab**
- typedef struct [array_string](#) **array_string**
- typedef struct [stack_instr](#) **stack_instr**

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)
- int **stack_instr_init** ([stack_instr](#) *stack)
- int **stack_instr_push** ([stack_instr](#) *stack, [I_Instr](#) *instr)
- [I_Instr](#) * **stack_instr_pop** ([stack_instr](#) *stack)
- void **stack_instr_destroy** ([stack_instr](#) *stack)

4.9.1 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

4.9.2 Macro Definition Documentation

4.9.2.1 `#define ARRAY_HTAB_INIT_SIZE 64`

Default size for allocation memory for array of hash tables

4.9.2.2 `#define ARRAY_STRING_INIT_SIZE 8`

Default size for allocation memory for array of strings

4.9.2.3 `#define STACK_HTAB_INIT_SIZE 16`

Default size for allocation memory for Stack of hash tables

4.9.2.4 `#define STACK_INSTR_INIT_SIZE 8`

Default size for allocation memory of stack of instructions

4.9.3 Typedef Documentation

4.9.3.1 `typedef struct array_htab array_htab`

Array of hash tables

4.9.3.2 `typedef struct array_string array_string`

Array of strings

4.9.3.3 `typedef struct stack_htab stack_htab`

Stack of hash tables

4.9.3.4 `typedef struct stack_instr stack_instr`

Stack of instructions

4.9.4 Function Documentation

4.9.4.1 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 initialized

4.9.4.2 `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 initialized

Precondition

Array was initialized

4.9.4.3 `int array_htab_init (array_htab * array)`

Initialize array

Parameters

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

Returns

0 on success, 1 if memory allocation failed

4.9.4.4 `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 success, 1 when reallocation failed

Precondition

Array was initialized

4.9.4.5 `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 initialized

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

Find string in array

Parameters

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

Returns

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

Precondition

Array was initialized

4.9.4.7 int array_string_init (array_string * array)

Initialize new array of strings

Parameters

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

Returns

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

Precondition

input pointer points to allocated space
Array was initialized

4.9.4.8 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 success, 1 in case of error while allocating memory

Precondition

Array was initialized

4.9.4.9 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 initialized

4.9.4.10 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

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

Retrive specific item from stack

Parameters

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

Returns

Pointer to specific item or NULL if backtrack is too big

Precondition

Stack was initialized

4.9.4.12 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

4.9.4.13 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 success if stack is already empty (before pop), return NULL

Precondition

Stack was initialized

4.9.4.14 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 initialized

Chapter 5

Class Documentation

5.1 array_htab Struct Reference

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

Public Attributes

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

5.1.1 Detailed Description

Array of hash tables

5.1.2 Member Data Documentation

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

Array of hash tables

5.1.2.2 [unsigned](#) array_htab::idx

Index where will be added new hash table

5.1.2.3 [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

5.2 array_string Struct Reference

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

Public Attributes

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

5.2.1 Detailed Description

Array of strings

5.2.2 Member Data Documentation

5.2.2.1 char** array_string::data

Array of hash tables

5.2.2.2 unsigned array_string::idx

Index where will be added new hash table

5.2.2.3 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

5.3 htab_item Struct Reference

```
#include <ial.h>
```

Public Attributes

- char * [key](#)
- unsigned [data_type](#)
- unsigned [func_or_var](#)
- void * [data](#)
- unsigned [initialized](#)
- unsigned [number_of_arguments](#)
- void * [local_table](#)
- void * [instruction_tape](#)
- int [argument_index](#)
- struct [htab_item](#) * [next_item](#)

5.3.1 Detailed Description

Abstract data type that represents item of hash table

5.3.2 Member Data Documentation

5.3.2.1 int htab_item::argument_index

If item is argument of function, it tells which it is argument

5.3.2.2 void* htab_item::data

pointer to the place with data, for function it is int* (int array of data_types of parametres)

5.3.2.3 unsigned htab_item::data_type

Data type for variable or returns type of function

5.3.2.4 unsigned htab_item::func_or_var

Tells if item represents function or variable (0 - not defined, 1 - variable, 2 - function)

5.3.2.5 unsigned htab_item::initialized

0 - not initialized, 1 - initialized

5.3.2.6 void* htab_item::instruction_tape

Pointer to instruction tape of function

5.3.2.7 char* htab_item::key

String ID

5.3.2.8 void* htab_item::local_table

Pointer to local symbol table

5.3.2.9 struct htab_item* htab_item::next_item

Pointer to next item

5.3.2.10 unsigned htab_item::number_of_arguments

Number of arguments in function

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

- ial.h

5.4 htab_t Struct Reference

```
#include <ial.h>
```

Public Attributes

- unsigned(* [hash_fun_ptr](#))(const char *str, unsigned [htab_size](#))
- unsigned [htab_size](#)
- unsigned [number_items](#)
- [htab_item](#) ** ptr

5.4.1 Detailed Description

Abstract data type that represents hash table

5.4.2 Member Data Documentation

5.4.2.1 unsigned(* [htab_t::hash_fun_ptr](#))(const char *str, unsigned [htab_size](#))

Pointer to hash function, &hash_function by default

5.4.2.2 unsigned [htab_t::htab_size](#)

Size of table (number of lines)

5.4.2.3 unsigned [htab_t::number_items](#)

Real number of items

5.4.2.4 [htab_item](#)** [htab_t::ptr](#)

Array of pointers to items

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

- [ial.h](#)

5.5 I_Instr Struct Reference

Public Attributes

- int **type_instr**
- void * **adr1**
- void * **adr2**
- void * **adr3**
- struct [I_Instr](#) * **next_instr**

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

- [structures.h](#)

5.6 Instr_List Struct Reference

Public Attributes

- [l_instr](#) * **Active**
- [l_instr](#) * **Last**

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

- structures.h

5.7 mem_item_t Struct Reference

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

Public Attributes

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

5.7.1 Detailed Description

Item that holds one pointer to allocated memory

5.7.2 Member Data Documentation

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

Pointer to nex item

5.7.2.2 void* [mem_item_t::ptr](#)

Pointer to allocated memory

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

- garbage_collector.h

5.8 mem_list_t Struct Reference

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

Public Attributes

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

5.8.1 Detailed Description

List of items, that holds allocated memory

5.8.2 Member Data Documentation

5.8.2.1 struct mem_item_t* mem_list_t::first

Pointer to first item

5.8.2.2 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](#)

5.9 stack_expresion Struct Reference

Structure for stack of tokens.

```
#include <expression.h>
```

Public Attributes

- [token](#) * [arr](#)
Pointer to an array of tokens.
- int [size](#)
Size of stack (array length)
- int [top](#)
Index of a top element in a stack.

5.9.1 Detailed Description

Structure for stack of tokens.

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

- [expression.h](#)

5.10 stack_hstab Struct Reference

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

Public Attributes

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

5.10.1 Detailed Description

Stack of hash tables

5.10.2 Member Data Documentation

5.10.2.1 `htab_t** stack_htab::data`

Array of hash tables

5.10.2.2 `size_t stack_htab::size`

Maximum number of items after last allocation

5.10.2.3 `int stack_htab::top`

Index of item on top of stack

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

- `structures.h`

5.11 `stack_instr` Struct Reference

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

Public Attributes

- `int top`
- `size_t size`
- `I_Instr** data`

5.11.1 Detailed Description

Stack of instructions

5.11.2 Member Data Documentation

5.11.2.1 `I_Instr** stack_instr::data`

Array of hash tables

5.11.2.2 `size_t stack_instr::size`

Maximum number of items after last allocation

5.11.2.3 `int stack_instr::top`

Index of item on top of stack

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

- `structures.h`

5.12 t_stack_int Struct Reference

Structure for stack of integers.

```
#include <ial.h>
```

Public Attributes

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

5.12.1 Detailed Description

Structure for stack of integers.

5.12.2 Member Data Documentation

5.12.2.1 int* t_stack_int::data

Pointer to an array of integers

5.12.2.2 int t_stack_int::size

Size of stack (array length)

5.12.2.3 int t_stack_int::top

Index of a top element in a stack

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

- ial.h

5.13 token Struct Reference

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

Public Attributes

- int [id](#)
- void * [ptr](#)

5.13.1 Detailed Description

Structure that represents token

5.13.2 Member Data Documentation

5.13.2.1 int token::id

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

5.13.2.2 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

Chapter 6

File Documentation

6.1 `embedded_functions.h` File Reference

Documentation for embeded functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include "garbage_collector.h"
```

Functions

- `char * readString ()`
Function reads a string from STDIN.
- `int readInt ()`
Function reads an integer value from STDIN.
- `double readDouble ()`
Function reads a number in double format from STDIN.
- `void print (char *string)`
Function prints string to STDOUT.

6.1.1 Detailed Description

Documentation for embeded functions.

Author

Sava Nedeljkovic

Date

11.12.2016

6.2 `expression.h` File Reference

Documentation for expression processing.

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

Classes

- struct [stack_expresion](#)
Structure for stack of tokens.

Macros

- #define [ERR_WARNING](#) 0
- #define [ERR_LEXICAL_ANALYSIS](#) 1
- #define [ERR_SYNTACTIC_ANALYSIS](#) 2
- #define [ERR_SEM_NDEF_REDEF](#) 3
- #define [ERR_SEM_COMPATIBILITY](#) 4
- #define [ERR_SEM_OTHERS](#) 6
- #define [ERR_INPUT_NUMBER](#) 7
- #define [ERR_UNINIALIZED_VAR](#) 8
- #define [ERR_DIVISION_ZERO](#) 9
- #define [ERR_OTHERS](#) 10
- #define [ERR_INTERN_FAULT](#) 99
- #define [FATAL_ERROR](#)(message, error_code)
- #define [STRDUP](#)(l, s)

Functions

- int [expr_analyze](#) (token t_in, token *t_out, char *class_name, int error_6_flag, token **postfix_token_array, int *token_count, int *expr_data_type, [htab_t](#) *global_table, [htab_t](#) *local_table,...)
Function analyzes precedence and converts expression to postfix format.
- int [stack_expression_init](#) (struct [stack_expresion](#) *s, int size)
Function initializes a stack, allocates required memory and sets its variables.
- int [stack_expression_destroy](#) (struct [stack_expresion](#) *s)
Function destroys a stack, frees its memory and sets its variables.
- int [stack_expression_empty](#) (const struct [stack_expresion](#) *s)
Function checks whether stack is empty.
- int [stack_expression_full](#) (const struct [stack_expresion](#) *s)
Function checks whether stack is full.
- int [stack_expression_top](#) (struct [stack_expresion](#) *s, token *t)
Function gives back top element from the stack.
- int [stack_expression_pop](#) (struct [stack_expresion](#) *s, token *t)
Function pops and gives back top element from the stack.
- int [stack_expression_push](#) (struct [stack_expresion](#) *s, token t)
Function pushes given element to the stack.
- int [operator_priority](#) (int op)
Function tells the priority of a given operator.
- int [type_priority](#) (int type)
Function tells the priority of a given data type, which is later used for determining data type of the whole expression.

- int [type_name_conversion](#) (int type)
Function converts names of the given data type, so it could be understood by the function [expr_analyze\(\)](#).
- void [print_token](#) (token t, int id_flag)
Function prints token value (and its id) to STDERR. This function is only used for debugging.
- void [print_token_array](#) (token *arr, int id_flag)
Function prints array of tokens (and their ids) to STDERR. This function is only used for debugging.

6.2.1 Detailed Description

Documentation for expression processing.

Author

Sava Nedeljkovic, xnedel08

Date

11.12.2016

This module is used for processing expressions. It checks whether expressions follows allowed rules. Final expression is converted to postfix format.

6.3 interpret.h File Reference

Documentation for interpret processing.

```
#include <string.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include "scanner.h"
#include "structures.h"
#include "expression.h"
```

Enumerations

- enum {
 I_ASSIGNMENT = 100+1, I_IF, I_ELSE, I_WHILE,
 I_END, I_FCE, I_RETURN, I_PRINT,
 I_ENDIF, I_ENDWHILE, I_ENDEELSE }

Functions

- int [Add_Instr](#) (Instr_List *L, I_Instr *new)
Function add instruction at end of instruction tape.
- token * [do_expression](#) (token *postfix_array, stack_hab *I_Htable, struct [stack_expression](#) *S, Instr_List *L, int void_flag)
Interpret calls this function when instruction has expression. Function process postfix array, manage all arithmetic and bool operation and calls interpret.
- token * [inter_plus](#) (token a, token b)
Function adds tokens data or concatenate tokens data if one of operands is string.
- token * [inter_arm_op](#) (token tmp1, token tmp2, int i)

- Function does arithmetic operation between two tokens data.*
- `token * inter_bool_op (token tmp1, token tmp2, int i)`
- Function does boolean operation between two tokens data.*
- `int inter (Instr_List *L, stack_htab *l_Htable, token *return_token, int void_flag)`
- Function does boolean operation between two tokens data.*
- `char * IntToString (int x)`
- Function translates integer to string.*
- `char * DoubleToString (double x)`
- Function translates double to string.*
- `char * Conc_Str (char *s1, char *s2)`
- Function concatenates two strings.*
- `htab_item * stack_htab_find_htab_item (stack_htab *stack, char *key)`
- `htab_t * stack_htab_get_first (stack_htab *stack)`
- `void I_Instr_null_elements (I_Instr *Instruction)`

6.3.1 Detailed Description

Documentation for interpret processing.

Author

Nemanja Vasiljevic, xvasil03

Date

11.12.2016

This module is used for processing Instruction list made of 3AC.

Index

ALPHABET_ARRAY

Functions for string processing, [20](#)

ARRAY_HTAB_INIT_SIZE

Structures, [44](#)

ARRAY_STRING_INIT_SIZE

Structures, [44](#)

Add_Instr

Interpret processing, [30](#)

argument_index

htab_item, [53](#)

array_htab, [51](#)

data, [51](#)

idx, [51](#)

size, [51](#)

Structures, [44](#)

array_htab_destroy

Structures, [44](#)

array_htab_get_item

Structures, [46](#)

array_htab_init

Structures, [46](#)

array_htab_insert

Structures, [46](#)

array_string, [51](#)

data, [52](#)

idx, [52](#)

size, [52](#)

Structures, [44](#)

array_string_destroy

Structures, [46](#)

array_string_find

Structures, [47](#)

array_string_init

Structures, [47](#)

array_string_insert

Structures, [47](#)

BLOCK_COMMENT

Lexical analysis, [37](#)

bin2dec

Lexical analysis, [37](#)

computeJumps

Functions for string processing, [20](#)

Conc_Str

Interpret processing, [31](#)

data

array_htab, [51](#)

array_string, [52](#)

htab_item, [53](#)

stack_htab, [57](#)

stack_instr, [57](#)

t_stack_int, [58](#)

data_type

htab_item, [53](#)

do_expression

Interpret processing, [31](#)

DoubleToString

Interpret processing, [31](#)

ERR_WARNING

Expression processing, [10](#)

embedded_functions.h, [61](#)

Embeded functions, [7](#)

readDouble, [7](#)

readInt, [7](#)

readString, [7](#)

expr_analyze

Expression processing, [10](#)

Expression processing, [9](#)

ERR_WARNING, [10](#)

expr_analyze, [10](#)

FATAL_ERROR, [10](#)

operator_priority, [11](#)

print_token, [11](#)

print_token_array, [12](#)

STRDUP, [10](#)

stack_expression_destroy, [12](#)

stack_expression_empty, [12](#)

stack_expression_full, [12](#)

stack_expression_init, [13](#)

stack_expression_pop, [13](#)

stack_expression_push, [13](#)

stack_expression_top, [15](#)

type_name_conversion, [15](#)

type_priority, [15](#)

expression.h, [61](#)

FATAL_ERROR

Expression processing, [10](#)

find

Functions for string processing, [20](#)

first

mem_list_t, [56](#)

free_memory

Garbage collector, [17](#)

func_or_var

htab_item, [53](#)

Functions for string processing, [20](#)

- ALPHABET_ARRAY, 20
- computeJumps, 20
- find, 20
- length, 21
- MAX, 20
- shellsort, 21
- substring, 21
- GARBAGE_COLLECTOR
 - Garbage collector, 19
- Garbage collector, 17
 - free_memory, 17
 - GARBAGE_COLLECTOR, 19
 - mem_alloc, 17
 - mem_item_t, 17
 - mem_list_t, 17
 - mem_realloc, 19
- get_token
 - Lexical analysis, 38
- Hash table, 26
 - htab_clear_items, 26
 - htab_copy, 26
 - htab_find_item, 27
 - htab_find_item_by_argument_index, 27
 - htab_free_all, 27
 - htab_init, 27
 - htab_init2, 27
 - htab_insert_item, 29
 - htab_item, 26
 - htab_t, 26
- hash_fun_ptr
 - htab_t, 54
- hex2dec_double
 - Lexical analysis, 38
- hex2dec_int
 - Lexical analysis, 38
- htab_clear_items
 - Hash table, 26
- htab_copy
 - Hash table, 26
- htab_find_item
 - Hash table, 27
- htab_find_item_by_argument_index
 - Hash table, 27
- htab_free_all
 - Hash table, 27
- htab_init
 - Hash table, 27
- htab_init2
 - Hash table, 27
- htab_insert_item
 - Hash table, 29
- htab_item, 52
 - argument_index, 53
 - data, 53
 - data_type, 53
 - func_or_var, 53
 - Hash table, 26
 - initialized, 53
 - instruction_tape, 53
 - key, 53
 - local_table, 53
 - next_item, 53
 - number_of_arguments, 53
- htab_size
 - htab_t, 54
- htab_t, 53
 - Hash table, 26
 - hash_fun_ptr, 54
 - htab_size, 54
 - number_items, 54
 - ptr, 54
- I_Instr, 54
- id
 - token, 59
- idx
 - array_htab, 51
 - array_string, 52
- initialized
 - htab_item, 53
- Instr_List, 55
- instruction_tape
 - htab_item, 53
- IntToString
 - Interpret processing, 33
- inter
 - Interpret processing, 32
- inter_arm_op
 - Interpret processing, 32
- inter_bool_op
 - Interpret processing, 32
- inter_plus
 - Interpret processing, 33
- Interpret processing, 30
 - Add_Instr, 30
 - Conc_Str, 31
 - do_expression, 31
 - DoubleToString, 31
 - IntToString, 33
 - inter, 32
 - inter_arm_op, 32
 - inter_bool_op, 32
 - inter_plus, 33
 - stack_htab_find_htab_item, 33
- interpret.h, 63
- is_full_ident
 - Lexical analysis, 38
- is_keyword
 - Lexical analysis, 38
- is_num_literal
 - Lexical analysis, 40
- is_simple_ident
 - Lexical analysis, 40
- is_special_char
 - Lexical analysis, 40

- key
 - htab_item, 53
- LINE_COMMENT
 - Lexical analysis, 37
- last
 - mem_list_t, 56
- length
 - Functions for string processing, 21
- Lexical analysis, 35
 - BLOCK_COMMENT, 37
 - bin2dec, 37
 - get_token, 38
 - hex2dec_double, 38
 - hex2dec_int, 38
 - is_full_ident, 38
 - is_keyword, 38
 - is_num_literal, 40
 - is_simple_ident, 40
 - is_special_char, 40
 - LINE_COMMENT, 37
 - load_string, 41
 - make_power, 41
 - octal2dec, 41
 - repair_num, 42
 - reset_scanner, 36
 - S_AND, 37
 - S_ASSIGNMENT, 37
 - S_BOOLEAN, 36
 - S_BREAK, 36
 - S_CLASS, 36
 - S_COMMA, 37
 - S_CONTINUE, 36
 - S_DIV, 37
 - S_DO, 36
 - S_DOUBLE, 36
 - S_ELSE, 36
 - S_EOF, 37
 - S_EQUAL, 37
 - S_FALSE, 36
 - S_FOR, 36
 - S_FULL_IDENT, 37
 - S_GREATER, 37
 - S_GREATER_EQUAL, 37
 - S_IF, 37
 - S_INT, 37
 - S_LEFT_BRACE, 37
 - S_LEFT_PARE, 37
 - S_LESS, 37
 - S_LESS_EQUAL, 37
 - S_MINUS, 37
 - S_MUL, 37
 - S_NOT, 37
 - S_NOT_EQUAL, 37
 - S_OR, 37
 - S_PLUS, 37
 - S_RETURN, 37
 - S_RIGHT_BRACE, 37
 - S_RIGHT_PARE, 37
 - S_SEMICOMMA, 37
 - S_SIMPLE_IDENT, 37
 - S_SIZE, 36
 - S_STATIC, 37
 - S_STRING, 37
 - S_TRUE, 37
 - S_VOID, 37
 - S_WHILE, 37
 - SPEC_CHAR_FSEEK, 36
 - skip_comment, 42
 - str2num, 42
 - TYPE_BOOLEAN, 37
 - TYPE_DOUBLE, 37
 - TYPE_DOUBLE_HEX, 37
 - TYPE_INT, 37
 - TYPE_INT_BIN, 37
 - TYPE_INT_HEX, 37
 - TYPE_INT_OCTAL, 37
 - TYPE_STRING, 37
 - token, 36
- load_string
 - Lexical analysis, 41
- local_table
 - htab_item, 53
- MAX
 - Functions for string processing, 20
- make_power
 - Lexical analysis, 41
- mem_alloc
 - Garbage collector, 17
- mem_item_t, 55
 - Garbage collector, 17
 - next, 55
 - ptr, 55
- mem_list_t, 55
 - first, 56
 - Garbage collector, 17
 - last, 56
- mem_realloc
 - Garbage collector, 19
- next
 - mem_item_t, 55
- next_item
 - htab_item, 53
- number_items
 - htab_t, 54
- number_of_arguments
 - htab_item, 53
- octal2dec
 - Lexical analysis, 41
- operator_priority
 - Expression processing, 11
- print_token
 - Expression processing, 11
- print_token_array

- Expression processing, [12](#)
- ptr
 - htab_t, [54](#)
 - mem_item_t, [55](#)
 - token, [59](#)
- readDouble
 - Embedded functions, [7](#)
- readInt
 - Embedded functions, [7](#)
- readString
 - Embedded functions, [7](#)
- repair_num
 - Lexical analysis, [42](#)
- reset_scanner
 - Lexical analysis, [36](#)
- S_AND
 - Lexical analysis, [37](#)
- S_ASSIGNMENT
 - Lexical analysis, [37](#)
- S_BOOLEAN
 - Lexical analysis, [36](#)
- S_BREAK
 - Lexical analysis, [36](#)
- S_CLASS
 - Lexical analysis, [36](#)
- S_COMMA
 - Lexical analysis, [37](#)
- S_CONTINUE
 - Lexical analysis, [36](#)
- S_DIV
 - Lexical analysis, [37](#)
- S_DO
 - Lexical analysis, [36](#)
- S_DOUBLE
 - Lexical analysis, [36](#)
- S_ELSE
 - Lexical analysis, [36](#)
- S_EOF
 - Lexical analysis, [37](#)
- S_EQUAL
 - Lexical analysis, [37](#)
- S_FALSE
 - Lexical analysis, [36](#)
- S_FOR
 - Lexical analysis, [36](#)
- S_FULL_IDENT
 - Lexical analysis, [37](#)
- S_GREATER
 - Lexical analysis, [37](#)
- S_GREATER_EQUAL
 - Lexical analysis, [37](#)
- S_IF
 - Lexical analysis, [37](#)
- S_INT
 - Lexical analysis, [37](#)
- S_LEFT_BRACE
 - Lexical analysis, [37](#)
- S_LEFT_PARE
 - Lexical analysis, [37](#)
- S_LESS
 - Lexical analysis, [37](#)
- S_LESS_EQUAL
 - Lexical analysis, [37](#)
- S_MINUS
 - Lexical analysis, [37](#)
- S_MUL
 - Lexical analysis, [37](#)
- S_NOT
 - Lexical analysis, [37](#)
- S_NOT_EQUAL
 - Lexical analysis, [37](#)
- S_OR
 - Lexical analysis, [37](#)
- S_PLUS
 - Lexical analysis, [37](#)
- S_RETURN
 - Lexical analysis, [37](#)
- S_RIGHT_BRACE
 - Lexical analysis, [37](#)
- S_RIGHT_PARE
 - Lexical analysis, [37](#)
- S_SEMICOMMA
 - Lexical analysis, [37](#)
- S_SIMPLE_IDENT
 - Lexical analysis, [37](#)
- S_SIZE
 - Lexical analysis, [36](#)
- S_STATIC
 - Lexical analysis, [37](#)
- S_STRING
 - Lexical analysis, [37](#)
- S_TRUE
 - Lexical analysis, [37](#)
- S_VOID
 - Lexical analysis, [37](#)
- S_WHILE
 - Lexical analysis, [37](#)
- SPEC_CHAR_FSEEK
 - Lexical analysis, [36](#)
- STACK_HTAB_INIT_SIZE
 - Structures, [44](#)
- STACK_INSTR_INIT_SIZE
 - Structures, [44](#)
- STRDUP
 - Expression processing, [10](#)
- shellsort
 - Functions for string processing, [21](#)
- size
 - array_htab, [51](#)
 - array_string, [52](#)
 - stack_htab, [57](#)
 - stack_instr, [57](#)
 - t_stack_int, [58](#)
- skip_comment
 - Lexical analysis, [42](#)

- Stack of integers, 23
 - stack_int_clean, 23
 - stack_int_create, 23
 - stack_int_destroy, 24
 - stack_int_is_empty, 24
 - stack_int_is_full, 24
 - stack_int_pop, 24
 - stack_int_push, 25
 - stack_int_top, 25
- stack_expresion, 56
- stack_expression_destroy
 - Expression processing, 12
- stack_expression_empty
 - Expression processing, 12
- stack_expression_full
 - Expression processing, 12
- stack_expression_init
 - Expression processing, 13
- stack_expression_pop
 - Expression processing, 13
- stack_expression_push
 - Expression processing, 13
- stack_expression_top
 - Expression processing, 15
- stack_htab, 56
 - data, 57
 - size, 57
 - Structures, 44
 - top, 57
- stack_htab_destroy
 - Structures, 47
- stack_htab_find_htab_item
 - Interpret processing, 33
- stack_htab_get_first
 - Structures, 49
- stack_htab_get_item
 - Structures, 49
- stack_htab_init
 - Structures, 49
- stack_htab_pop
 - Structures, 49
- stack_htab_push
 - Structures, 50
- stack_instr, 57
 - data, 57
 - size, 57
 - Structures, 44
 - top, 57
- stack_int_clean
 - Stack of integers, 23
- stack_int_create
 - Stack of integers, 23
- stack_int_destroy
 - Stack of integers, 24
- stack_int_is_empty
 - Stack of integers, 24
- stack_int_is_full
 - Stack of integers, 24
- stack_int_pop
 - Stack of integers, 24
- stack_int_push
 - Stack of integers, 25
- stack_int_top
 - Stack of integers, 25
- str2num
 - Lexical analysis, 42
- Structures, 43
 - ARRAY_HTAB_INIT_SIZE, 44
 - ARRAY_STRING_INIT_SIZE, 44
 - array_htab, 44
 - array_htab_destroy, 44
 - array_htab_get_item, 46
 - array_htab_init, 46
 - array_htab_insert, 46
 - array_string, 44
 - array_string_destroy, 46
 - array_string_find, 47
 - array_string_init, 47
 - array_string_insert, 47
 - STACK_HTAB_INIT_SIZE, 44
 - STACK_INSTR_INIT_SIZE, 44
 - stack_htab, 44
 - stack_htab_destroy, 47
 - stack_htab_get_first, 49
 - stack_htab_get_item, 49
 - stack_htab_init, 49
 - stack_htab_pop, 49
 - stack_htab_push, 50
 - stack_instr, 44
- substring
 - Functions for string processing, 21
- t_stack_int, 58
 - data, 58
 - size, 58
 - top, 58
- TYPE_BOOLEAN
 - Lexical analysis, 37
- TYPE_DOUBLE
 - Lexical analysis, 37
- TYPE_DOUBLE_HEX
 - Lexical analysis, 37
- TYPE_INT
 - Lexical analysis, 37
- TYPE_INT_BIN
 - Lexical analysis, 37
- TYPE_INT_HEX
 - Lexical analysis, 37
- TYPE_INT_OCTAL
 - Lexical analysis, 37
- TYPE_STRING
 - Lexical analysis, 37
- token, 58
 - id, 59
 - Lexical analysis, 36
 - ptr, 59
- top

- stack_htab, [57](#)
 - stack_instr, [57](#)
 - t_stack_int, [58](#)
- type_name_conversion
 - Expression processing, [15](#)
- type_priority
 - Expression processing, [15](#)