

```

/*****
Scanner
*****/
%option noyywrap

%{
#include <string.h>
#include "axe_struct.h"
#include "collections.h"
#include "Acse.tab.h"
#include "axe_constants.h"

extern int line_num;
extern int num_error;

extern int yyerror(const char* errmsg);

%}
/*****
TOKEN DEFINITIONS
*****/
DIGIT [0-9]
ID [a-zA-Z_][a-zA-Z0-9_]*

/*****
TOKENS
*****/
%option noyywrap
%x comment
%%

"\r\n" { ++line_num; }
"\n" { ++line_num; }

[ \t\f\v]+ { /* Ignore whitespace. */ }

/*"[^\\n]*
/*" BEGIN(comment);

<comment>[^\\n]*
<comment>[^\\n]*\n { ++line_num; }
<comment>"+[^\n]*
<comment>"+[^\n]*\n { ++line_num; }
<comment>"+[^\n]*
BEGIN(INITIAL);

" " { return LBRACE; }
" [" { return RBRACE; }
" [" { return LSQUARE; }
" [" { return RSQUARE; }
" (" { return LPAR; }
" (" { return RPAR; }
";" { return SEMI; }
";" { return COLON; }
";" { return PLUS; }
";" { return MINUS; }
";" { return MUL_OP; }
"/" { return DIV_OP; }
"% " { return MOD_OP; }
"& " { return AND_OP; }
"| " { return OR_OP; }
"! " { return NOT_OP; }
"=" { return ASSIGN; }
"< " { return LT; }
"> " { return GT; }
"<< " { return SHL_OP; }
">> " { return SHR_OP; }
"=" { return EQ; }
"!=" { return NOTEQ; }
"<=" { return LTEQ; }
">=" { return GTEQ; }
"&& " { return ANDAND; }
"|" { return OROR; }
"|" { return COMMA; }

"do" { return DO; }
"else" { return ELSE; }
"for" { return FOR; }
"if" { return IF; }
"int" { yylval.intval = INTEGER_TYPE; return TYPE; }
"while" { return WHILE; }
"return" { return RETURN; }
"read" { return READ; }
"write" { return WRITE; }

{ID} { yylval.svalue=strdup(yytext); return IDENTIFIER; }
{DIGIT}+ { yylval.intval = atoi(yytext); return NUMBER; }

. { yyerror("Error: unexpected token");
num_error++;
return (-1); /* invalid token */
}

```

```

/*****
PARSER
*****/
%{
#include "axe_struct.h"
#include "symbol_table.h"
#include "axe_errors.h"
#include "collections.h"
#include "axe_expressions.h"
#include "axe_gencode.h"
#include "axe_utils.h"
#include "axe_array.h"
#include "axe_io_manager.h"

int line_num;
int num_error; /* number of errors */
int num_warning; /* number of warnings */
t_program_infos *program; /* ALL PROGRAM INFORMATION */

%}
/*****
SEMANTIC RECORDS
*****/
%union {
int intval;
char *svalue;
t_axe_expression expr;
t_axe_declaration *decl;
t_list *list;
t_axe_label *label;
t_while_statement while_stmt;
}
/*****
TOKENS
*****/
%start program

%token LBRACE RBRACE LPAR RPAR LSQUARE RSQUARE
%token SEMI COLON PLUS MINUS MUL_OP DIV_OP MOD_OP
%token AND_OP OR_OP NOT_OP
%token ASSIGN LT GT SHL_OP SHR_OP EQ NOTEQ LTEQ GTEQ
%token ANDAND OROR
%token COMMA
%token FOR
%token RETURN
%token READ
%token WRITE

%token <label> DO
%token <while_stmt> WHILE
%token <label> IF
%token <label> ELSE
%token <intval> TYPE
%token <svalue> IDENTIFIER
%token <intval> NUMBER

%type <expr> exp
%type <decl> declaration
%type <list> declaration_list
%type <label> if_stmt

/*****
OPERATOR PRECEDENCES
*****/
%left COMMA
%left ASSIGN
%left OROR
%left ANDAND
%left OR_OP
%left AND_OP
%left EQ NOTEQ
%left LT GT LTEQ GTEQ
%left SHL_OP SHR_OP
%left MINUS PLUS
%left MUL_OP DIV_OP
%right NOT

/*****
BISON GRAMMAR
*****/
%}

program : var_declarations statements
{
/* Notify the end of program and generates HALT */
set_end_Program(program);
YYACCEPT;
}
;

```

```

var_declarations : var_declarations var_declaration
| /* empty */
;

var_declaration : TYPE declaration_list SEMI
{
set_new_variables(program, $1, $2);
}
;

declaration_list : declaration_list COMMA declaration
{
$$ = addElement($1, $3, -1);
declaration
}
;
{
$$ = addElement(NULL, $1, -1);
}
;

declaration : IDENTIFIER ASSIGN NUMBER
{
$$ = alloc_declaration($1, 0, 0, $3);
if ($$ == NULL)
notifyError(AXE_OUT_OF_MEMORY);
}
;
{
IDENTIFIER LSQUARE NUMBER RSQUARE
}
;
{
$$ = alloc_declaration($1, 1, $3, 0);
if ($$ == NULL)
notifyError(AXE_OUT_OF_MEMORY);
}
;
{
IDENTIFIER
}
;
{
$$ = alloc_declaration($1, 0, 0, 0);
if ($$ == NULL)
notifyError(AXE_OUT_OF_MEMORY);
}
;

code_block : statement
| LBRACE statements RBRACE
;

statements : statements statement
| statement
;

statement : assign_statement SEMI
| control_statement
| read_write_statement SEMI
| SEMI
{ gen_nop_instruction(program); }
;

control_statement : if_statement
| while_statement
| do_while_statement SEMI
| return_statement SEMI
;

read_write_statement : read_statement
| write_statement
;

assign_statement : IDENTIFIER LSQUARE exp RSQUARE ASSIGN exp
{
storeArrayElement(program, $1, $3, $6);
free($1);
}
;
{
IDENTIFIER ASSIGN exp
}
;
{
int location;
t_axe_instruction *instr;
location = get_symbol_location(program, $1, 0);
}
;
{
if ($3.expression_type == IMMEDIATE)
instr = gen_add_instruction
(program, location, REG_0, $3.value);
else
instr = gen_add_instruction
(program, location, REG_0, $3.value, CG_DIRECT_ALL);
}
;
{
free($1);
}
;

if_statement : if_stmt
{
assignLabel(program, $1);
}
;
{
if_stmt ELSE
}
;
{
$2 = newLabel(program);
gen_bt_instruction(program, $2, 0);
assignLabel(program, $1);
}
;
{
code_block
}
;
{
assignLabel(program, $2);
}
;

```

```

if_stmt : IF
{
    $1 = newLabel(program);
    LPAR exp RPAR
    {
        if ($4.expression_type == IMMEDIATE)
            gen_load_immediate(program, $4.value);
        else
            gen_andb_instruction(program, $4.value,
                $4.value, $4.value, CG_DIRECT_ALL);
            gen_beg_instruction (program, $1, 0);
    }
    code_block { $$ = $1; }
;

while_statement : WHILE
{
    $1 = create_while_statement();
    $1.label_condition
        = assignNewLabel(program);
    LPAR exp RPAR
    {
        if ($4.expression_type == IMMEDIATE)
            gen_load_immediate(program, $4.value);
        else
            gen_andb_instruction(program, $4.value,
                $4.value, $4.value, CG_DIRECT_ALL);
            $1.label_end = newLabel(program);
            gen_beg_instruction (program, $1.label_end, 0);
    }
    code_block
    {
        /* jump to the beginning of the loop */
        gen_bt_instruction
            (program, $1.label_condition, 0);

        /* fix the label 'label_end' */
        assignLabel(program, $1.label_end);
    }
;

do_while_statement : DO
{
    $1 = newLabel(program);
    assignLabel(program, $1);
    code_block WHILE LPAR exp RPAR
    {
        if ($6.expression_type == IMMEDIATE)
            gen_load_immediate(program, $6.value);
        else
            gen_andb_instruction(program, $6.value,
                $6.value, $6.value, CG_DIRECT_ALL);
            gen_bne_instruction (program, $1, 0);
    }
;

return_statement : RETURN
{
;

read_statement : READ LPAR IDENTIFIER RPAR
{
    int location;
    location = get_symbol_location(program, $3, 0);
    gen_read_instruction (program, location);
    free($3);
;

write_statement : WRITE LPAR exp RPAR
{
    int location;
    if ($3.expression_type == IMMEDIATE)
    {
        location = gen_load_immediate(program, $3.value);
    }
    else
        location = $3.value;
    gen_write_instruction (program, location);
;

exp : NUMBER { $$ = create_expression ($1, IMMEDIATE); }
| IDENTIFIER {
    int location;
    location = get_symbol_location(program, $1, 0);
    $$ = create_expression (location, REGISTER);
    free($1);
}
| IDENTIFIER LSQUARE exp RSQUARE {
    int reg;
    reg = loadArrayElement(program, $1, $3);
    $$ = create_expression (reg, REGISTER);
    free($1);
;

```

```

} NOT_OP NUMBER { if ($2 == 0)
    $$ = create_expression (1, IMMEDIATE);
    else
    $$ = create_expression (0, IMMEDIATE);
;

} NOT_OP IDENTIFIER {
    int identifier_location;
    int output_register;
    identifier_location =
        get_symbol_location(program, $2, 0);
    output_register =
        getNewRegister(program);
    gen_notl_instruction (program, output_register
        , identifier_location);
    $$ = create_expression (output_register, REGISTER);
    free($2);
;

} exp AND_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, ANDB); }
| exp OR_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, ORB); }
| exp PLUS exp {
    $$ = handle_bin_numeric_op(program, $1, $3, ADD); }
| exp MINUS exp {
    $$ = handle_bin_numeric_op(program, $1, $3, SUB); }
| exp MUL_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, MUL); }
| exp DIV_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, DIV); }
| exp LT exp {
    $$ = handle_binary_comparison (program, $1, $3, _LT_); }
| exp GT exp {
    $$ = handle_binary_comparison (program, $1, $3, _GT_); }
| exp EQ exp {
    $$ = handle_binary_comparison (program, $1, $3, _EQ_); }
| exp NOTEQ exp {
    $$ = handle_binary_comparison (program, $1, $3, _NOTEQ_); }
| exp LTEQ exp {
    $$ = handle_binary_comparison (program, $1, $3, _LTEQ_); }
| exp GTEQ exp {
    $$ = handle_binary_comparison (program, $1, $3, _GTEQ_); }
| exp SHL_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, SHL); }
| exp SHR_OP exp {
    $$ = handle_bin_numeric_op(program, $1, $3, SHR); }
| exp ANDAND exp {
    $$ = handle_bin_numeric_op(program, $1, $3, ANDL); }
| exp OROR exp {
    $$ = handle_bin_numeric_op(program, $1, $3, ORL); }
| LPAR exp RPAR { $$ = $2; }
| MINUS exp { if ($2.expression_type == IMMEDIATE)
    {
        $$ = $2;
        $$ .value = - ($$.value);
    }
    else
    {
        t_axe_expression exp_r0;
        exp_r0.value = REG_0;
        exp_r0.expression_type = REGISTER;
        $$ = handle_bin_numeric_op
            (program, exp_r0, $2, SUB);
    }
;

%%
/*=====
MAIN
=====*/
int main (int argc, char **argv)
{
    /* initialize all the compiler data structures and global variables */
    init_compiler(argc, argv);

    yyparse();

    return 0;
;

/*=====
YYERROR
=====*/
int yyerror(const char* errmsg)
{
    errorcode = AXE_SYNTAX_ERROR;

    return 0;
;

```

```

*****
HOW TO GENERATE JUMPS
*****
This is an example:

gen_beg_instruction( ... label ... )
    Generates a jump-if-equal instruction (i.e., jump if flag zero is set)
    to 'label'.
    That means a jump to 'label' if the preceding expression is FALSE.

```

```

/* *****
 * axe_array.h
 * CODE GENERATION FOR ARRAY MANAGEMENT (LOAD/STORE)
 * *****/

/* Generates the instructions for loading the content of an
 * array element into a register.
 * ID: array name
 * index: index of the array cell
 * Returns the register number
 * */
extern int loadArrayElement(t_program_infos *program
, char *ID, t_axe_expression index);

/* Generates the instructions for loading an array cell
 * address into a register.
 * */
extern int loadArrayAddress(t_program_infos *program
, char *ID, t_axe_expression index);

/* Generates the instructions for storing data into the array
 * cell indexed by ID and index*/
extern void storeArrayElement(t_program_infos *program, char *ID
, t_axe_expression index, t_axe_expression data);

```

```

/* *****
 * axe_engine.h
 * Contains t_program_infos and some functions for LABEL MANAGEMENT
 * (reserve, fix, assign)
 * *****/
typedef struct t_program_infos
{
    t_list *variables;
    t_list *instructions;
    t_list *data;
    t_axe_label_manager *lmanager;
    t_symbol_table *sy_table;
    int current_register;
} t_program_infos;

/* initialize the informations associated with the program. */
extern t_program_infos * allocProgramInfos();

/* add a new instruction to the current program. This function is directly
 * called by all the functions defined in 'axe_gencode.h' */
extern void addInstruction(t_program_infos *program, t_axe_instruction *instr);

/* reserve a new label identifier and return the identifier to the caller */
extern t_axe_label * newLabel(t_program_infos *program);

/* assign the given label identifier to the next instruction. Returns
 * the label assigned; otherwise (an error occurred) LABEL_UNSPECIFIED */
extern t_axe_label * assignLabel(t_program_infos *program, t_axe_label *label);

/* reserve and fix a new label. It returns either the label assigned or the
 * value LABEL_UNSPECIFIED if an error occurred */
extern t_axe_label * assignNewLabel(t_program_infos *program);

/* add a variable to the program */
extern void createVariable(t_program_infos *program
, char *ID, int type, int isArray, int arraySize, int init_val);

/* get a previously allocated variable */
extern t_axe_variable * getVariable
(t_program_infos *program, char *ID);

/* get the label that marks the starting address of the variable
 * with name "ID" */
extern t_axe_label * getLabelFromVariableID
(t_program_infos *program, char *ID);

/* get a register still not used. This function returns
 * the ID of the register found*/
extern int getNewRegister(t_program_infos *program);

```

```

/* *****
 * axe_expressions.h
 * *****/

/* This function generates instructions for binary numeric
 * operations. It takes as input two expressions and a binary
 * operation identifier, and it returns a new expression that
 * represents the result of the specified binary operation
 * applied to 'expl' and 'exp2'.
 *
 * Valid values for 'binop' are:
 * ADD
 * ANDB
 * ORB
 * SUB
 * MUL
 * DIV */
extern t_axe_expression handle_bin_numeric_op (t_program_infos *program
, t_axe_expression expl, t_axe_expression exp2, int binop);

/* This function generates instructions that perform a
 * comparison between two values. It takes as input two
 * expressions and a binary comparison identifier, and it
 * returns a new expression that represents the result of the
 * specified binary comparison between 'expl' and 'exp2'.
 *
 * Valid values for 'condition' are:
 * _LT_ (used when is needed to test if the value of 'expl' is less than
 * the value of 'exp2')
 * _GT_ (used when is needed to test if the value of 'expl' is greater than
 * the value of 'exp2')
 * _EQ_ (used when is needed to test if the value of 'expl' is equal to
 * the value of 'exp2')
 * _NOTEQ_ (used when is needed to test if the value of 'expl' is not equal to
 * the value of 'exp2')
 * _LTEQ_ (used when is needed to test if the value of 'expl' is less than
 * or equal to the value of 'exp2')
 * _GTEQ_ (used when is needed to test if the value of 'expl' is greater tha
 * n
 * the value of 'exp2') */
extern t_axe_expression handle_binary_comparison (t_program_infos *program
, t_axe_expression expl, t_axe_expression exp2, int condition);

```

```

/* *****
 * axe_gencode.h
 * CODE GENERATION
 * *****
*/
/*-----
 * ----- NOP & HALT
 * -----*/
extern t_axe_instruction * gen_nop_instruction
(t_program_infos *program);

extern t_axe_instruction * gen_halt_instruction
(t_program_infos *program);

/*-----
 * ----- UNARY OPERATIONS
 * -----*/

/* A LOAD instruction requires the following parameters:
 * 1. A destination register (where will be loaded the requested value)
 * 2. A label information (can be a NULL pointer. If so, the address
 * value will be taken into consideration)
 * 3. A direct address (if label is different from NULL) */
extern t_axe_instruction * gen_load_instruction
(t_program_infos *program, int r_dest, t_axe_label *label, int address);

extern t_axe_instruction * gen_read_instruction
(t_program_infos *program, int r_dest);

extern t_axe_instruction * gen_write_instruction
(t_program_infos *program, int r_dest);

/* A STORE instruction copies a value from a register to a
 * specific memory location. The memory location can be
 * either a label identifier or a address reference.
 * In order to create a STORE instruction the caller must
 * provide a valid register location ('r_dest') and an
 * instance of 't_axe_label' or a numeric address */
extern t_axe_instruction * gen_store_instruction
(t_program_infos *program, int r_dest, t_axe_label *label, int address);

/* A MOVA instruction copies an address value into a register.
 * An address can be either an instance of 't_axe_label'
 * or a number (numeric address) */
extern t_axe_instruction * gen_mova_instruction
(t_program_infos *program, int r_dest, t_axe_label *label, int address);

/*-----
 * ----- STATUS REGISTER TEST INSTRUCTIONS
 * -----*/

/* rdest = 1 if the last numeric operation is gte 0
 * rdest = 0 otherwise */
extern t_axe_instruction * gen_sge_instruction
(t_program_infos *program, int r_dest);

/* EQ */
extern t_axe_instruction * gen_seq_instruction
(t_program_infos *program, int r_dest);

/* GT */
extern t_axe_instruction * gen_sgt_instruction
(t_program_infos *program, int r_dest);

/* LE */
extern t_axe_instruction * gen_sle_instruction
(t_program_infos *program, int r_dest);

/* LT */
extern t_axe_instruction * gen_slt_instruction
(t_program_infos *program, int r_dest);

/* NE */
extern t_axe_instruction * gen_sne_instruction
(t_program_infos *program, int r_dest);

/*-----
 * ----- BINARY OPERATIONS
 * -----*/

/*-----
 * ----- REGISTER = REGISTER OP IMMEDIATE
 * -----
 * Suffix "li" means logical
 * Suffix "bi" means bitwise
 * Prefix "e" means exclusive
 * -----*/
extern t_axe_instruction * gen_addi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_subi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_andli_instruction

```

```

(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_orli_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_eorli_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_andbi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_muli_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_orbi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_eorbi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_divi_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

// SHL = shift left
extern t_axe_instruction * gen_shli_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_shri_instruction
(t_program_infos *program, int r_dest, int r_source1, int immediate);

extern t_axe_instruction * gen_notl_instruction
(t_program_infos *program, int r_dest, int r_source1);

extern t_axe_instruction * gen_notb_instruction
(t_program_infos *program, int r_dest, int r_source1);

/*-----
 * ----- TERNARY OPERATIONS
 * -----
 * REGISTER = REGISTER OP REGISTER
 * Suffix "l" means logical
 * Suffix "b" means bitwise
 * -----*/

extern t_axe_instruction * gen_add_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_sub_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_andl_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_orl_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_eorl_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_andb_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_orb_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_eorb_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_mull_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_div_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_shl_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_shr_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

extern t_axe_instruction * gen_neg_instruction
(t_program_infos *program, int r_dest, int r_source1);

extern t_axe_instruction * gen_spl_instruction
(t_program_infos *program, int r_dest, int r_source1, int r_source2, int flags);

/*-----
 * ----- JUMP INSTRUCTIONS
 * -----*/

extern t_axe_instruction * gen_bt_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bf_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bhi_instruction

```

```

(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bls_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bcc_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bcs_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bne_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_beq_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bvc_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bvs_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bpl_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bmi_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bge_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_blt_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_bgt_instruction
(t_program_infos *program, t_axe_label *label, int addr);

extern t_axe_instruction * gen_ble_instruction
(t_program_infos *program, t_axe_label *label, int addr);

/*
 * See also:
 * axe_utils.h for gen_load_immediate()
 */

```

```

/* *****
 * axe_labels.h
 * AUXILIARY FUNCTIONS FOR LABEL MANAGEMENT
 * *****

/* get the number of labels inside the list of labels */
extern int get_number_of_labels(t_axe_label_manager *lmanager);

/* return TRUE if the two labels hold the same identifier */
extern int compareLabels(t_axe_label *labelA, t_axe_label *labelB);

/* test if a label will be assigned to the next instruction */
extern int isAssignedLabel(t_axe_label_manager *lmanager);

```

```

/*
 See also:
 axe_engine.h for the main label-related functions
 */

```

```

/* *****
 * axe_struct.h
 * FUNDAMENTAL DATA STRUCTURES
 * *****

typedef struct t_axe_label
{
    int labelID; /* label identifier */
} t_axe_label;

typedef struct t_axe_register
{
    int ID; /* an identifier of the register */
    int indirect; /* a boolean value: 1 if the register value is a pointer */
} t_axe_register;

typedef struct t_axe_address
{
    int addr; /* a Program Counter */
    t_axe_label *labelID; /* a label identifier */
    int type; /* one of ADDRESS_TYPE or LABEL_TYPE */
} t_axe_address;

/* A structure that defines the internal data of a 'Acse variable' */
typedef struct t_axe_variable
{
    int type; /* a valid data type @see 'axe_constants.h' */
    int isArray; /* must be TRUE if the current variable is an array */
    int arraySize; /* the size of the array. This information is useful only
    * if the field 'isArray' is TRUE */
    int init_val; /* initial value of the current variable. Actually it is
    * implemented as a integer value. 'int' is
    * the only supported type at the moment,
    * future developments could consist of a modification of
    * the supported type system. Thus, maybe init_val will be
    * modified in future. */
    char *ID; /* variable identifier (should never be a NULL
    * pointer or an empty string "") */
    t_axe_label *labelID; /* a label that refers to the location
    * of the variable inside the data segment */
} t_axe_variable;

/* a symbolic assembly instruction */
typedef struct t_axe_instruction
{
    int opcode; /* instruction opcode (for example: AXE_ADD) */
    /
    t_axe_register *reg_1; /* destination register */
    t_axe_register *reg_2; /* first source register */
    t_axe_register *reg_3; /* second source register */
    int immediate; /* immediate value */
    t_axe_address *address; /* an address operand */
    char *user_comment; /* if defined it is set to the source code
    * instruction that generated the current
    * assembly. This string will be written
    * into the output code as a comment */
    t_axe_label *labelID; /* a label associated with the current
    * instruction */
} t_axe_instruction;

/* this structure is used in order to define assembler directives.
 * Directives are used in many cases such the definition of variables
 * inside the data segment. Every instance 't_axe_data' contains
 * all the informations about a single directive.
 * An example is the directive .word that is required when the assembler
 * must reserve a word of data inside the data segment. */
typedef struct t_axe_data
{
    int directiveType; /* the type of the current directive
    * (for example: DIR_WORD) */
    int value; /* the value associated with the directive */
    t_axe_label *labelID; /* label associated with the current data */
} t_axe_data;

typedef struct t_axe_expression
{
    int value; /* an immediate value or a register identifier */
    int expression_type; /* actually only integer values are supported */
} t_axe_expression;

typedef struct t_axe_declaration
{
    int isArray; /* must be TRUE if the current variable is an array */
    int arraySize; /* the size of the array. This information is useful o
nly
    * if the field 'isArray' is TRUE */
    int init_val; /* initial value of the current variable. */
    char *ID; /* variable identifier (should never be a NULL pointer
    * or an empty string "") */
} t_axe_declaration;

typedef struct t_while_statement
{
    t_axe_label *label_condition; /* this label points to the expression
    * that is used as loop condition */

```

```

    t_axe_label *label_end; /* this label points to the instruction
    * that follows the while construct */
} t_while_statement;

/* create a label */
extern t_axe_label * alloc_label(int value);

/* create an expression */
extern t_axe_expression create_expression (int value, int type);

/* create an instance that will maintain infos about a while statement */
extern t_while_statement create_while_statement();

/* create an instance of 't_axe_register' */
extern t_axe_register * alloc_register(int ID, int indirect);

/* create an instance of 't_axe_instruction' */
extern t_axe_instruction * alloc_instruction(int opcode);

/* create an instance of 't_axe_address' */
extern t_axe_address * alloc_address(int type, int address, t_axe_label *label);

/* create an instance of 't_axe_data' */
extern t_axe_data * alloc_data(int directiveType, int value, t_axe_label *label);

/* create an instance of 't_axe_variable' */
extern t_axe_variable * alloc_variable
(char *ID, int type, int isArray, int arraySize, int init_val);

/* finalize an instance of 't_axe_variable' */
extern void free_variable (t_axe_variable *variable);

/* create an instance of 't_axe_variable' */
extern t_axe_declaration * alloc_declaration
(char *ID, int isArray, int arraySize, int init_val);

/* finalize an instruction info. */
extern void free_instruction(t_axe_instruction *inst);

/* finalize a data info. */
extern void free_data(t_axe_data *data);

```

```

/* *****
 * axe_utils.h
 * Some important functions to access the list of symbols
 * *****/

/* create a variable for each 't_axe_declaration' inside
 * the list 'variables'. Each new variable will be of type
 * 'varType'. */
extern void set_new_variables(t_program_infos *program
, int varType, t_list *variables);

/* Given a variable/symbol identifier (ID) this function
 * returns a register location where the value is stored
 * (the value of the variable identified by 'ID').
 * If the variable/symbol has never been loaded from memory
 * to a register, first this function searches
 * for a free register, then it assign the variable with the given
 * ID to the register just found.
 * Once computed, the location (a register identifier) is returned
 * as output to the caller.
 * This function generates a LOAD instruction
 * only if the flag 'genLoad' is set to 1; otherwise it simply reserve
 * a register location for a new variable in the symbol table.
 * If an error occurs, get_symbol_location returns a REG_INVALID errorcode */
extern int get_symbol_location(t_program_infos *program
, char *ID, int genLoad);

/* Generate the instruction to load an 'immediate' value into a new register.
 * It returns the new register identifier or REG_INVALID if an error occurs */
extern int gen_load_immediate(t_program_infos *program, int immediate);

/* Notify the end of the program. This function is directly called
 * from the parser when the parsing process is ended */
extern void set_end_Program(t_program_infos *program);

```

```

/* *****
 * collections.h
 * A double linked list
 * *****/

/* a list element */
typedef struct t_list
{
    void *data;
    struct t_list *next;
    struct t_list *prev;
}t_list;

/* add an element 'data' to the list 'list' at position 'pos'./
extern t_list * addElement(t_list *list, void * data, int pos);

/* add sorted */
extern t_list * addSorted(t_list *list, void * data
, int (*compareFunc)(void *a, void *b));

/* add an element to the end of the list */
extern t_list * addLast(t_list *list, void * data);

/* add an element at the beginning of the list */
extern t_list * addFirst(t_list *list, void * data);

/* remove an element at the beginning of the list */
extern t_list * removeFirst(t_list *list);

/* remove an element from the list */
extern t_list * removeElement(t_list *list, void * data);

/* remove a link from the list 'list' */
extern t_list * removeElementLink(t_list *list, t_list *element);

/* find an element inside the list 'list'. The current implementation calls the
 * CustomfindElement' passing a NULL reference as 'func' */
extern t_list * findElement(t_list *list, void *data);

/* find an element inside the list 'list'. */
extern t_list * CustomfindElement(t_list *list, void *data
, int (*compareFunc)(void *a, void *b));

/* find the position of an 'element' inside the 'list'. -1 if not found */
extern int getPosition(t_list *list, t_list *element);

/* find the length of 'list' */
extern int getLength(t_list *list);

/* remove all the elements of a list */
extern void freeList(t_list *list);

/* get the last element of the list. Returns NULL if the list is empty
 * or list is a NULL pointer */
extern t_list * getLastElement(t_list *list);

/* retrieve the list element at position 'position' inside the 'list'.
 * Returns NULL if: the list is empty, the list is a NULL pointer or
 * the list holds less than 'position' elements. */
extern t_list * getElementAt(t_list *list, unsigned int position);

/* create a new list with the same elements */
extern t_list * cloneList(t_list *list);

/* add a list of elements to another list */
extern t_list * addList(t_list *list, t_list *elements);

/* add a list of elements to a set */
extern t_list * addListToSet(t_list *list, t_list *elements
, int (*compareFunc)(void *a, void *b), int *modified);

```

```

/* *****
 * symbol_table.h
 * Some important functions to manage the symbol table
 * *****/

/* a symbol inside the sy_table. An element of the symbol table is composed by
 * three fields: <ID>, <type> and <Location>.
 * 'ID' is a not-NULL string that is used as key identifier for a symbol
 * inside the table.
 * 'type' is an integer value that is used to determine the correct type
 * of a symbol. Valid values for 'type' are defined into "sy_table_constants.h".
 * 'reg_location' refers to a register location (i.e. which register contains
 * the value of 'ID'). */
typedef struct
{
    char *ID; /* symbol identifier */
    int type; /* type associated with the symbol */
    int reg_location; /* a register location */
}t_symbol;

/* put a symbol into the symbol table */
extern int putSym(t_symbol_table *table, char *ID, int type);

/* set the location of the symbol with ID as identifier */
extern int setLocation(t_symbol_table *table, char *ID, int reg);

/* get the location of the symbol with the given ID */
extern int getLocation(t_symbol_table *table, char *ID, int *errorcode);

/* get the type associated with the symbol with ID as identifier */
extern int getTypeFromID(t_symbol_table *table, char *ID, int type);

/* given a register identifier (location), it returns the ID of the variable
 * stored inside the register 'location'. This function returns NULL
 * if the location is an invalid location. */
extern char * getIDfromLocation(t_symbol_table *table
, int location, int *errorcode);

/*
 * See also:
 * axe_utils.h for wrapper functions related to variables
 * axe_array.h for wrapper functions related to array variables
 */

```