```
%{
/*
Vahe Karamian - CS 440 - Project 2
Filename: project2.l, project2.h, project2.c
*/
#include <stdio.h>
%}

%token ID
%token SEMICOLON
%token OBRAK
%token CBRAK
%token NUM
%token INT
%token VOID
%token OPAR
%token CPAR
%token CBRAC
%token OBRAC
%token IF
%token ELSE
%token WHILE
%token RETURN
%token RELOP
%token ADDOP
%token MULOP
%token COMMA
%token ASSIGN

%%

program:  declaration_list { printf("program -> declaration_list \n"); }
          ;

declaration_list :  declaration_list declaration
                            {
                               printf("declaration_list -> declaration_list decleration \n");
                            }
                    |  declaration
                            {
                               printf("declaration_list -> declaration \n");
                            }
                    ;

declaration  :  var_declaration
                    {
                       printf("declaration -> var_declaration \n");
                    }
                |  fun_declaration
                    {
                       printf("declaration -> fun_declaration \n");
                    }
                ;

var_declaration :  type_specifier ID SEMICOLON
                        {
                           printf( "var_declaration -> type_specifier ID; \n" );
                        }
                    |  type_specifier ID OBRAK NUM CBRAK SEMICOLON
```

```
                        {
                          printf(" var_declaration -> type_specifier ID [ NUM ]; \n");
                        }
              | error SEMICOLON
                  {
                    printf("var decl error");
                  }
                    ;

type_specifier :  INT
                    {
                      printf(" type_specifier -> int \n");
                    }
                  | VOID
                    {
                      printf(" type_specifier -> void \n");
                    }
                  ;

fun_declaration :  type_specifier ID OPAR params CPAR compound_stmt
                      {
                        printf(" fun_declaration -> type_specifier ID ( params ) compound_stmt \n");
                      }
                    ;

params :  param_list
                {
                  printf(" params -> param_list \n");
                }
         | VOID
                {
                  printf(" params -> void \n");
                }
         ;

param_list :  param_list COMMA param
                    {
                      printf(" param_list -> param_list, param \n");
                    }
              | param
                    {
                      printf(" param_list -> param \n");
                    }
           | param_list COMMA ID
                {
                  printf("PL error");
                }
           | param_list COMMA type_specifier
                {
                  printf("PL error2");
                }
              ;

param   :  type_specifier ID
              {
                printf(" param -> type_specifier ID \n");
              }
         | type_specifier ID OBRAK CBRAK
              {
```

```
            printf(" param -> type_specifier ID[] \n");
          }
      ;

compound_stmt :  OBRAC local_declarations statement_list CBRAC
                 {
                   printf(" compound_stmt -> { local_declarations statement_list } \n");
                 }
              ;

local_declarations :  local_declarations var_declaration
                      {
                        printf(" local_declarations -> local_declarations var_declaration \n");
                      }
                   |  {printf(" e \n");}
                   ;

statement_list :  statement_list statement
                  {
                    printf(" statement_list -> statement_list statement \n");
                  }
               |  {printf(" e \n");}
               ;

statement :  expression_stmt
             {
               printf(" statement -> expression_stmt \n");
             }
          |  compound_stmt
             {
               printf(" statement -> compound_stmt \n");
             }
          |  selection_stmt
             {
               printf(" statement -> selection_stmt \n");
             }
          |  iteration_stmt
             {
               printf(" statement -> iteration_stmt \n");
             }
          |  return_stmt
             {
               printf(" statement -> return_stmt \n");
             }
          ;
          /*
          |  error  SEMICOLON   { yyerror(" error inside statement "); yyerrok; }
          ;
          */

expression_stmt   :  expression SEMICOLON
                     {
                       printf(" expression_stmt -> expression ; \n");
                     }
                  |  SEMICOLON
                     {
                       printf(" expression_stmt -> ; \n");
                     }
                  ;
```

```
selection_stmt :  IF OPAR expression CPAR statement
                        {
                          printf(" selection_stmt -> if (expression) statement \n");
                        }
                | IF OPAR expression CPAR statement ELSE statement
                        {
                          printf(" selection_stmt -> if (expression) statement else statement \n");
                        }
                ;

iteration_stmt :  WHILE OPAR expression CPAR statement
                        {
                          printf(" iteration_stmt -> while (expression) statement \n");
                        }
                ;

return_stmt :  RETURN SEMICOLON
                      {
                        printf(" return_stmt -> return; \n");
                      }
             | RETURN expression SEMICOLON
                      {
                        printf(" return_stmt -> return expression; \n");
                      }
             ;

expression :  var ASSIGN expression
                      {
                        printf(" expression -> var = expression \n");
                      }
            | simple_expression
                      {
                        printf(" expression -> simple_expression \n");
                      }
            ;

var  :  ID
          {
            printf(" var -> ID \n");
          }
     | ID OBRAK expression CBRAK
          {
            printf(" var -> ID [ expression ] \n");
          }
     ;

simple_expression   :  simple_expression RELOP additive_expression
                         {
                           printf(" simple_expression -> simple_expression relop additive_expression \n");
                         }
                    | additive_expression
                         {
                           printf(" simple_expression -> additive_expression \n");
                         }
                    ;

additive_expression  :  additive_expression ADDOP term
                           {
```

```
                        printf(" additive_expression addop term \n");
                    }
                | term
                    {
                        printf(" additive_expression -> term \n");
                    }
                ;

term   : term MULOP factor
            {
                printf(" term -> term mulop factor \n");
            }
        | factor
            {
                printf(" term -> factor \n");
            }
        ;

factor :  OPAR expression CPAR
            {
                printf(" factor -> ( expression ) \n");
            }
         | var
            {
                printf(" factor -> var \n");
            }
         | NUM
            {
                printf(" factor -> NUM \n");
            }
         | call
            {
                printf(" factor -> call \n");
            }
         ;

call :  ID OPAR args CPAR
            {
                printf(" call -> ID ( args ) \n");
            }
        ;

args   : arg_list
            {
                printf(" args -> arg_list \n");
            }
        |  {printf(" e \n");}
        ;

arg_list :  arg_list COMMA expression
                {
                    printf(" arg_list -> arg_list, expression \n");
                }
           | expression
                {
                    printf(" arg_list -> expression \n");
                }
           ;
```

```
%%

#include            "proj1.c"
```