Lex

```
%{
#include "y.tab.h"
#include <string.h>
%}

%%

"if"                    { return IF; }
"else"                  { return ELSE; }
"while"                 { return WHILE; }
"read"                  { return READ; }
"write"                 { return WRITE; }
"int"                   { return INT; }
"float"                 { return FLOAT; }
"string"                { return STRING; }
"char"                  { return CHAR; }
"array"                 { return ARRAY; }
"of"                    { return OF; }

"("                     { return LPAREN; }
")"                     { return RPAREN; }
"{"                     { return LBRACE; }
"}"                     { return RBRACE; }
"["                     { return LBRACKET; }
"]"                     { return RBRACKET; }
";"                     { return SEMICOLON; }
","                     { return COMMA; }
"="                     { return ASSIGN; }

"+"                     { return PLUS; }
"-"                     { return MINUS; }
"*"                     { return MULTIPLY; }
"/"                     { return DIVIDE; }
"%"                     { return MODULO; }
"=="                    { return EQUALS; }
"!="                    { return NOTEQUAL; }
"<"                     { return LT; }
">"                     { return GT; }
```

```
"<="                        { return LTE; }
">="                        { return GTE; }
"&&"                        { return AND; }
"||"                        { return OR; }

[a-zA-Z_][a-zA-Z0-9_]*  { yylval.str_val = strdup(yytext); return IDENTIFIER; }
[0-9]+                  { yylval.int_val = atoi(yytext); return INTEGER_CONSTANT; }
[0-9]+\.[0-9]+          { yylval.float_val = atof(yytext); return FLOAT_CONSTANT; }


\"([^"\\]|\\.)*\" {
    yytext[strlen(yytext) - 1] = '\0';
    yylval.str_val = strdup(yytext + 1);
    return STRING_CONSTANT;
}

[ \t\n]+                    ;
"//".*                      ;

.                           { printf("Unrecognized token: %s\n", yytext); }

%%

int yywrap() {
    return 1;
}
```

Yacc

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int yylex();
extern FILE *yyin;

void yyerror(const char *s);

struct row_entry {
    char *name;
    struct row_entry *first_child;
    struct row_entry *next_sibling;
};

static struct row_entry* cons(const char* name, struct row_entry* first_child) {
    struct row_entry* node = malloc(sizeof(struct row_entry));
    if (!node) {
        fprintf(stderr, "Memory allocation failed!\n");
        exit(1);
    }
    node->name = strdup(name);
    node->first_child = first_child;
    node->next_sibling = NULL;
    return node;
}

static void display(struct row_entry* node, int depth) {
    if (!node) return;
    for (int i = 0; i < depth; i++) {
        printf("  ");
    }
    printf("%s\n", node->name);
    display(node->first_child, depth + 1);
    display(node->next_sibling, depth);
}

static void free_row_entry(struct row_entry* node) {
    if (!node) return;
```

```
    free_row_entry(node->first_child);
    free_row_entry(node->next_sibling);
    free(node->name);
    free(node);
}
%}


%union {
    struct row_entry* row;
    int int_val;
    float float_val;
    char* str_val;
}

%type <row> program statement_list statement assignment_statement if_statement
while_statement io_statement declaration_statement type array_type default_type
expression_list expression term factor condition relational_operator
%token <int_val> INTEGER_CONSTANT
%token <float_val> FLOAT_CONSTANT
%token <str_val> STRING_CONSTANT IDENTIFIER
%token IF ELSE WHILE READ WRITE INT FLOAT STRING CHAR ARRAY OF
%token LPAREN RPAREN LBRACE RBRACE LBRACKET RBRACKET SEMICOLON COMMA ASSIGN
%token PLUS MINUS MULTIPLY DIVIDE MODULO EQUALS NOTEQUAL LT GT LTE GTE AND OR

%%

program:
    statement_list { display($1, 0); free_row_entry($1); }
;

statement_list:
    statement
    { $$ = cons("statement_list", $1); }
    | statement statement_list
     { $$ = cons("statement_list", $1); $1->next_sibling = $2; }
;

statement:
    assignment_statement
    { $$ = cons("assignment_statement", $1); }
    | if_statement
      { $$ = cons("if_statement", $1); }
```

```
    | while_statement
      { $$ = cons("while_statement", $1); }
    | io_statement
      { $$ = cons("io_statement", $1); }
    | declaration_statement
      { $$ = cons("declaration_statement", $1); }
;

assignment_statement:
    IDENTIFIER ASSIGN expression SEMICOLON
    { $$ = cons("assignment_statement", cons($1, $3)); }
;

if_statement:
    IF LPAREN condition RPAREN LBRACE statement_list RBRACE
    { $$ = cons("if", $3); $3->next_sibling = $6; }
    | IF LPAREN condition RPAREN LBRACE statement_list RBRACE
      ELSE LBRACE statement_list RBRACE
     { $$ = cons("if-else", $3); $3->next_sibling = $6; $6->next_sibling = $10; }
;

while_statement:
    WHILE LPAREN condition RPAREN LBRACE statement_list RBRACE
    { $$ = cons("while", $3); $3->next_sibling = $6; }
;

io_statement:
    WRITE LPAREN expression_list RPAREN SEMICOLON { $$ = cons("write", $3); }
    | READ LPAREN IDENTIFIER RPAREN SEMICOLON { $$ = cons("read", cons($3, NULL)); }
;

declaration_statement:
    type IDENTIFIER SEMICOLON
    { $$ = cons("declaration_statement", cons($2, NULL)); }
    | type IDENTIFIER ASSIGN expression SEMICOLON
      { $$ = cons("declaration_statement_with_assignment", cons($2, $4)); }
;

type:
    INT { $$ = cons("INT", NULL); }
    | FLOAT { $$ = cons("FLOAT", NULL); }
    | STRING { $$ = cons("STRING", NULL); }
    | CHAR { $$ = cons("CHAR", NULL); }
```

```
    | array_type { $$ = cons("array_type", $1); }
;

array_type:
    ARRAY LBRACKET INTEGER_CONSTANT RBRACKET OF default_type
    { $$ = cons("array_type",
            cons("ARRAY",
            cons("LBRACKET",
            cons("INTEGER_CONSTANT",
             cons("RBRACKET",
             cons("OF", $6))))));  }
;

default_type:
    INT { $$ = cons("INT", NULL); }
    | FLOAT { $$ = cons("FLOAT", NULL); }
    | STRING { $$ = cons("STRING", NULL); }
    | CHAR { $$ = cons("CHAR", NULL); }
;

expression_list:
    expression { $$ = cons("expression_list", $1); }
    | expression COMMA expression_list
      { $$ = cons("expression_list", $1); $1->next_sibling = $3; }
;

expression:
    term { $$ = cons("expression", $1); }
    | expression PLUS term { $$ = cons("+", $1); $1->next_sibling = $3; }
    | expression MINUS term { $$ = cons("-", $1); $1->next_sibling = $3; }
;

term:
    factor { $$ = cons("term", $1); }
    | term MULTIPLY factor { $$ = cons("*", $1); $1->next_sibling = $3; }
    | term DIVIDE factor { $$ = cons("/", $1); $1->next_sibling = $3; }
    | term MODULO factor { $$ = cons("%", $1); $1->next_sibling = $3; }
;

factor:
    IDENTIFIER { $$ = cons($1, NULL); }
    | INTEGER_CONSTANT { $$ = cons("int_const", NULL); }
    | FLOAT_CONSTANT { $$ = cons("float_const", NULL); }
```

```
    | STRING_CONSTANT { $$ = cons("string_const", NULL); }
    | LPAREN expression RPAREN { $$ = $2; }
;

condition:
    expression relational_operator expression
    { $$ = cons("condition", $1); $1->next_sibling = $3; }
;

relational_operator:
    EQUALS { $$ = cons("==", NULL); }
    | NOTEQUAL { $$ = cons("!=", NULL); }
    | LT { $$ = cons("<", NULL); }
    | GT { $$ = cons(">", NULL); }
    | LTE { $$ = cons("<=", NULL); }
    | GTE { $$ = cons(">=", NULL); }
;

%%

void yyerror(const char *s) {
    fprintf(stderr, "Error: %s\n", s);
}

int main(int argc, char **argv) {
    if (argc > 1) {
        FILE *file = fopen(argv[1], "r");
        if (!file) {
            perror("Error opening file");
            return 1;
        }
        yyin = file;
    }

    yyparse();

    if (argc > 1) {
        fclose(yyin);
    }

    return 0;
}
```

Test

p1.mm

```
int a;
read(a);
int b;
read(b);
int max;
if (a > b) {
    max = a;
}
else {
    max=b;
}
write(max);

/* OUTPUT:
statement_list
  declaration_statement
    declaration_statement
      a
  statement_list
    io_statement
      read
        a
    statement_list
      declaration_statement
        declaration_statement
          b
      statement_list
        io_statement
          read
            b
        statement_list
          declaration_statement
            declaration_statement
              max
          statement_list
            if_statement
              if-else
                condition
```

```
                expression
                  term
                    a
                expression
                  term
                    b
              statement_list
                assignment_statement
                  assignment_statement
                    max
                      expression
                        term
                          a
              statement_list
                assignment_statement
                  assignment_statement
                    max
                      expression
                        term
                          b
        statement_list
          io_statement
            write
              expression_list
                expression
                  term
                    max
*/
```

p2.mm

```
int a;
read(a);

int i;
i = 1;

int sum;
sum = 0;

while (i <= a) {
    if (a % i == 0) {
        sum = sum + i;
    }
    i = i + 1;
}

write(sum);
/* OUTPUT:
statement_list
  declaration_statement
    declaration_statement
      a
  statement_list
    io_statement
      read
        a
    statement_list
      declaration_statement
        declaration_statement
          i
      statement_list
        assignment_statement
          assignment_statement
            i
              expression
                term
                  int_const
        statement_list
          declaration_statement
            declaration_statement
              sum
```

```
statement_list
  assignment_statement
    assignment_statement
      sum
        expression
          term
            int_const
  statement_list
    while_statement
      while
        condition
          expression
            term
              i
          expression
            term
              a
        statement_list
          if_statement
            if
              condition
                expression
                  %
                    term
                      a
                    i
                expression
                  term
                    int_const
              statement_list
                assignment_statement
                  assignment_statement
                    sum
                      +
                        expression
                          term
                            sum
                        term
                          i
            statement_list
              assignment_statement
                assignment_statement
                  i
```

```
                        +
                          expression
                            term
                              i
                          term
                            int_const
                statement_list
                  io_statement
                    write
                      expression_list
                        expression
                          term
                            sum
*/
```

p3.mm

```
int n;
n = 7;
n = 7.5;
n = 7.00000;
n = 0.00007;
n = 0.700000;
n = 0.0000700;
read(n);

int sum;
sum = 0;

int i;
i = 0;

int val;

while (i < n) {
    read(val);
    sum = sum + val;
    i = i + 1;
}

write("sum", sum);
write("su\"m", sum);
write("su\'m", sum);
write('a', sum);
/* OUTPUT:
statement_list
  declaration_statement
    declaration_statement
      n
  statement_list
    io_statement
      read
        n
    statement_list
      declaration_statement
        declaration_statement
          sum
      statement_list
```

```
assignment_statement
  assignment_statement
    sum
      expression
        term
          int_const
statement_list
  declaration_statement
    declaration_statement
      i
  statement_list
    assignment_statement
      assignment_statement
        i
          expression
            term
              int_const
    statement_list
      declaration_statement
        declaration_statement
          val
      statement_list
        while_statement
          while
            condition
              expression
                term
                  i
              expression
                term
                  n
            statement_list
              io_statement
                read
                  val
              statement_list
                assignment_statement
                  assignment_statement
                    sum
                      +
                        expression
                          term
                            sum
```

```
                        term
                          val
                statement_list
                  assignment_statement
                    assignment_statement
                      i
                        +
                          expression
                            term
                              i
                          term
                            int_const
          statement_list
            io_statement
              write
                expression_list
                  expression
                    term
                      string_const
                  expression_list
                    expression
                      term
                        sum
*/
```