Thais Minet
CS 4223 Checkpoint #5
10/19/17

MAIN.C

```
#include "y.tab.h"
#include "hashtable.h"
#include <stdio.h>

int main() {
    if (yyparse())
        printf("Syntax error\n");
    else {
        printf("SUCCESS!\n");
    }
    disp_table();

    return 0;
}
```

PARSER.YY - BISON FILE

```
%{
    #include <stdio.h>
    #include <stdlib.h>
    #include "hashtable.h"
    int datatype;
    int addr;
%}

%union {
    char *sval;
    int ival;
}

%token        MAIN
%token        END_STMT
%token        END
%token        DATA
%token        COLON
%token        REAL
%token        INTEGER
%token <sval> VARIABLE
%token        LBRACKET
%token        RBRACKET
%token <ival> INT_CONST
%token        COMMA
%token        ALGORITHM
%token        IF
%token        ELSE
%token        WHILE
%token        COUNTING
%token        UPWARD
%token        DOWNWARD
%token        TO
%token        EXIT
%token        PRINT
%token        READ
%token        REAL_CONST
```

```
%token        CHAR_STRING_CONST
%token        ASSIGNMENT
%token        AND
%token        OR
%token        NOT
%token        LESS
%token        LESS_EQ
%token        GREATER
%token        GREATER_EQ
%token        EQUAL
%token        NOT_EQ
%token        ADD
%token        SUB
%token        MUL
%token        DIV
%token        MOD
%token        LPAREN
%token        RPAREN
%token        BANG
%token        TRASH

%%
prgm            : routine
                ;

routine         : MAIN END_STMT data algorithm END MAIN END_STMT
                ;

data            : DATA COLON declarationList
                | DATA COLON
                ;

declarationList : declaration END_STMT declarationList
                | declaration END_STMT
                ;

declaration     : dataType COLON variableList
                ;

dataType        : INTEGER { datatype = 0; }
                | REAL { datatype = 1; }
                ;

variableList    : VARIABLE COMMA variableList
                                    {
                                        if (insert($1, datatype, 0, addr, 1) == -1)
                                            printf("Error: duplicate variable '%s' not inserted\n", $1);
                                        addr += 1;
                                    }
                | VARIABLE LBRACKET INT_CONST RBRACKET COMMA variableList
                                    {
                                        if (insert($1, datatype, 1, addr, $3) == -1)
                                            printf("Error: duplicate variable '%s' not inserted\n", $1);

                                        addr += $3;
                                    }
                | VARIABLE
                                    {
                                        if (insert($1, datatype, 0, addr, 1) == -1)
```

```
                                        printf("Error: duplicate variable '%s' not inserted\n", $1);
                                    addr += 1;
                                }
                | VARIABLE LBRACKET INT_CONST RBRACKET
                                {
                                    if (insert($1, datatype, 1, addr, $3) == -1)
                                        printf("Error: duplicate variable '%s' not inserted\n", $1);
                                    addr += $3;
                                }
                ;

algorithm       : ALGORITHM COLON
                ;

%%

int yyerror() {
    printf("Called yyerror()\n");
    return 0;
}
```

HASHTABLE.H - symbol table header file

```
#define BASE (64) // hash base should be larger than number symbols allowed in a variable
#define M (1031) // hash value

// symbol struct in the symbol table
typedef struct {
    char *name;
    int datatype; // 0 = integer, 1 = real
    int type; // 0 = scalar, 1 = array
    int addr;
    int size; // scalar = 1, array >= 1
} symbol;

symbol *symboltable[M];

// function declarations
int insert(char *name, int datatype, int type, int addr, int size);
unsigned long hash(const char *s, unsigned long m);
symbol make_symbol(char *name, int datatype, int type, int addr, int size);
int insert_symbol(symbol *symboltable[], symbol *s);
void disp_symbol(symbol s);
void disp_table();
```

HASHTABLE.C - symbol table implemented as hash table with linear probing

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "hashtable.h"

int insert(char *name, int datatype, int type, int addr, int size) {
    symbol *s;
    s = (symbol *) malloc(sizeof(symbol));
    *s = make_symbol(name, datatype, type, addr, size);

    return insert_symbol(symboltable, s);
}

int insert_symbol(symbol *symboltable[], symbol *s) {
    int i = 0;
    unsigned long h = hash(s->name, M);

    while (symboltable[(h+i)%M] != NULL) {
        if (!strcmp(symboltable[(h+i)%M]->name, s->name)) return -1; // duplicate variable
        i++;
    }
    symboltable[(h+i)%M] = s;

    return 0;
}

symbol make_symbol(char *name, int datatype, int type, int addr, int size) {
    symbol s;

    s.name = strdup(name);
    s.datatype = datatype;
    s.type = type;
    s.addr = addr;
    s.size = size;

    return s;
}

void disp_symbol(symbol s) {
    printf("name = %s\n", s.name);

    if (s.datatype == 0)
        printf("datatype = %s\n", "INTEGER");
    else
        printf("datatype = %s\n", "REAL");

    if (s.type == 0)
        printf("type = %s\n", "SCALAR");
    else
        printf("type = %s\n", "ARRAY");

    printf("address = %d\n", s.addr);
    printf("size = %d\n", s.size);
}

void disp_table() {
    for (int i = 0; i < M; i++) {
```

```
        if (symboltable[i] != NULL) {
            printf("--------------------\n");
            printf("POS: %d\n", i);
            disp_symbol(*symboltable[i]);
        }
    }
}

unsigned long hash(const char *s, unsigned long m) {
    unsigned long h;
    unsigned const char *us;

    // ensure char values >= 0
    us = (unsigned const char *) s;

    h = 0;
    while (*us != '\0') {
        h = (h * BASE + *us) % m;
        us++;
    }

    return h;
}
```