```
/*
 * hw8.c
 *
 *  Created on: Nov 15, 2014
 *      Author: Scot Matson
 *      Assn: 7
 *      Cour: CS49C
 *      Sect: 1
 */
#include <stdio.h>
#include <stdlib.h>     /* for atof() */
#include <ctype.h>
#include <string.h>
#include <math.h>
#include "calc.h"
#define MAXOP   100     /* max size of operand or o
perator */
#define NUMBER '0'
#define COMMENT '1'
#define MATHFUNC '2'
#define MEMORY '3'
#define STORE '4'
#define RECALL '5'
#define ERR '6'

/* reverse polish calculator */
int RPNCalc (FILE *fpi, FILE *fpo) {

    int type;
    double op1, op2, S0, S1, S2, S3, S4, S5, S6, S7
, S8, S9;
    char s[MAXOP];

    while ((type = getop(s, fpi)) != EOF) {
        switch(type) {
        case NUMBER:
            push(atof(s));
            break;
                case COMMENT:
                        fprintf(fpo, "%s\n", s);
                        break;
                // Quick cover your eyes, very larg
```

```
e blocks of code coming through!
                case MATHFUNC:
                        if      (strcmp(s, "sqrt")
== 0) { push(sqrt(pop())); }
                        else if (strcmp(s, "sin") =
= 0) { push(sin(pop())); }
                        else if (strcmp(s, "cos") =
= 0) { push(cos(pop())); }
                        else if (strcmp(s, "tan") =
= 0) { push(tan(pop())); }
                        else if (strcmp(s, "asin")
== 0) { push(asin(pop())); }
                        else if (strcmp(s, "acos")
== 0) { push(acos(pop())); }
                        else if (strcmp(s, "atan")
== 0) { push(atan(pop())); }
                        else if (strcmp(s, "exp") =
= 0) { push(exp(pop())); }
                        else if (strcmp(s, "log") =
= 0) { push(log(pop())); }
                        else if (strcmp(s, "pow") =
= 0) {
                                op2 = pop();
                                op1 = pop();

                                push(pow(op1, op2))
;
                        }
                        break;
                case STORE:
                        if      (strcmp(s, "S0") ==
 0) { S0 = pop(); }
                        else if (strcmp(s, "S1") ==
 0) { S1 = pop(); }
                        else if (strcmp(s, "S2") ==
 0) { S2 = pop(); }
                        else if (strcmp(s, "S3") ==
 0) { S3 = pop(); }
                        else if (strcmp(s, "S4") ==
 0) { S4 = pop(); }
                        else if (strcmp(s, "S5") ==
 0) { S5 = pop(); }
```

```
                        else if (strcmp(s, "S6") ==
 0) { S6 = pop(); }
                        else if (strcmp(s, "S7") ==
 0) { S7 = pop(); }
                        else if (strcmp(s, "S8") ==
 0) { S8 = pop(); }
                        else if (strcmp(s, "S9") ==
 0) { S9 = pop(); }
                        break;

                case RECALL:
                        if (strcmp(s, "R0") == 0) {
 push(S0); }
                        else if (strcmp(s, "R1") ==
 0) { push(S1); }
                        else if (strcmp(s, "R2") ==
 0) { push(S2); }
                        else if (strcmp(s, "R3") ==
 0) { push(S3); }
                        else if (strcmp(s, "R4") ==
 0) { push(S4); }
                        else if (strcmp(s, "R5") ==
 0) { push(S5); }
                        else if (strcmp(s, "R6") ==
 0) { push(S6); }
                        else if (strcmp(s, "R7") ==
 0) { push(S7); }
                        else if (strcmp(s, "R8") ==
 0) { push(S8); }
                        else if (strcmp(s, "R9") ==
 0) { push(S9); }
                        break;
                case ERR:
                        op1 = pop();
                        fprintf(fpo, "error: unknow
n memory command\n");
            fprintf(fpo, "\t%.16g\n", op1);
                        break;
        case '+':
            push(pop() + pop());
            break;
        case '*':
```

```
            push(pop() * pop());
            break;
        case '-':
            op2 = pop();
            push(pop() - op2);
            break;
        case '/':
            op2 = pop();
            if (op2 != 0.0) {
                push(pop() / op2);
            }
            else {
                fprintf(fpo, "error: zero divisor\n
");
            }
            break;
        case '\n':
            op2 = pop();
            fprintf(fpo, "\t%.16g\n", op2);
            push(op2);
            break;
        case '=':
            op2 = pop();
            fprintf(fpo, "\t%.16g\n", op2);
            push(op2);
            break;
        case 'X':
            op1 = pop();
            op2 = pop();
            push(op1);
            push(op2);
            break;
                case 'D':
                        op1 = pop();
                        push(op1);
                        push(op1);
                        break;
        default:
            fprintf(fpo, "error: unknown command %s
\n", s);
            break;
        }
```

```
    }

    return 0;
}

int getop(char s[], FILE *fpi){
    int i, c;

    // This is getting an element from the array
    while ((s[0] = c = getch(fpi)) == ' ' || c == '
\t');

    s[1] = '\0';

    i = 0;
    if (!isdigit(c) && c != '.' && c != '-' && c !=
 '#' && !islower(c) && !isupper(c))
        return c;

    // For comments
    if (c == '#') {
        while(c != '\n') {
          s[++i] = c = getch(fpi);
        }
        s[i] = '\0';
        return COMMENT;
    }

    // For memory locations
    if (isupper(c)) {
      s[++i] = c = getch(fpi);
      if (isdigit(c)) {
        s[++i] = c = getch(fpi);
        s[i] = '\0';
        if (s[0] == 'S') {
          return STORE;
        }
        else if (s[0] == 'R') {
          return RECALL;
        }
        else
          return ERR;
```

```
      }
      else {
        if (c != EOF)
          ungetch(c);
        return s[0];
      }
    }

    // For math functions
    if (islower(c)) { // The only lowercase functio
ns, so it must be a math function.
      while (islower(s[++i] = c = getch(fpi))); //
Grabbing all the lowercase letters

      if (c != EOF) { // Putting back the last lett
er which broke the loop if not the EOF
        ungetch(c);
      }
       s[i] = '\0';
       return MATHFUNC;
    }

    // For negative numbers
    if (c == '-') {
        if (isdigit(c = getch(fpi)) || c == '.') {
            s[++i] = c;
        }
        else {
            if (c != EOF)
                ungetch(c);
            return '-';
        }
    }

    // For positive numbers
    if (isdigit(c)) {
        while (isdigit(s[++i] = c = getch(fpi)));
    }

    if (c == '.') {
        while (isdigit(s[++i] = c = getch(fpi)));
    }
```

```
    // If we reached the end of the file
    s[i] = '\0';
    if (c != EOF) {
        ungetch(c);
    }
    return NUMBER;
}
```