CSC201, SECTION 002, Fall 2000: Homework Assignment #4 Program

(Note: the homework problem descriptions are in a separate part of this assignment.)

DUE DATE

November 17 for the program, in class.

INSTRUCTIONS FOR PROGRAMMING

- Print off your program and hand in, folded lengthwise, stapled, with your name, course number, assignment number, and date on the outside.
- Submit your program electronically also.
- In your program, include the statement ".include ../pmacros.inc" (note the ".." to find the file in the parent directory). Do NOT include the sasmacros.inc file; you will not use these macros.
- Keep all your program files strictly in your EOS locker space, with no access given to other users. Do not put your files on the local machine, such as in temp space!
- Be sure your programs include a header comment indicating the assignment, date, and your name. Include a reasonable number of comments in your code as well.

Thanks for your help.

PROGRAMMING

1. Write a program called "hw4.asm" (note: don't use "hw4a.asm" this time, since there is only one program). This program reads a sequence of lines from the standard input. Each line is processed, and an output produced, before reading the next line. The program terminates when end of input is reached, or a line containing nothing (just a carriage return) is entered.

   Each input line consists of an arithmetic expression. A valid expression contains some combinations of numbers and operators (with no spaces). A number is a 1-digit unsigned decimal value (in ASCII) (ex.: 0, 8, 4, ...). An operator is '*', '/' (integer divide), '-', or '+'. As usual, '*' and '/' have higher precedence than '+' and '-'.

As each line is read, it is evaluated by your program and the value of the expression is printed as the output, in decimal. The only error condition you need to check for is a divide by zero, which produces an error message. For outputting an integer value, you are allowed to use the "put_i" macro of SASM (but no other macros from SASM), or you may write your own code.

As an example, for the input

```
1*2+3-5
3+1*2/2+8
1-2-0*2
8/0-2+1
9*9*9*9
3
```

the output should be

```
Value = 0
Value = 12
Value = -1
Error: divide by zero
Value = 6561
Value = 3
```

To evaluate these expressions, you will need to do stack-based evaluation. The main routine is as follows:

```
  define a number_stack, and an operator_stack;

  char = getchar();
  while (char != -1) {
    initialize number_stack to empty;
    initialize operator_stack to empty;
    while (char != 0ah) {
      if (char is a digit) {
        convert char to a number and push on the number_stack;
      }
      else {
        while ((operator_stack not empty ) && (precedence(char) <= precede
          evaluate top of operator_stack;
        }
        push(operator_stack, char);
      }
      char = getchar();
    }
    while (operator_stack not empty) {
      evaluate top of operator_stack;
    }
    print the top of the number_stack;
    char = getchar();
  }
```

The "evaluate top of operator_stack" routine is as follows:

```
operator = pop(operator_stack);
number2 = pop(number_stack);
number1 = pop(number_stack);
value = result of evaluating the expression "number1 operator number2";
push(number_stack, value);
```

For instance, suppose the user enters the expression "2+3*4/5-1*6". The execution flow is roughly as follows:

1. char = getchar() = '2'.
2. initialize the stacks to empty.
3. push value of '2' on the number_stack.
4. char = getchar() = '+'.
5. operator_stack is empty, so push '+' on the number_stack.
6. char = getchar() = '3'; push value on the number_stack (contents now = 2, 3).
7. char = getchar() = '*'.
8. operator_stack not empty, and precedence of '*' is greater than '+', so push '*' on operator stack (contents now = '+', '*').
9. char = getchar() = '4'; push value on the number_stack (contents = 2, 3, 4).
10. char = getchar() = '/'. operator_stack is not empty, and precedence of '/' is = to precedence of top of operator_stack, so evaluate the top expression:
    1. operator = pop operator_stack = '*' (contents now = '+').
    2. number2 = pop number_stack = 4 (contents now = 2, 3).
    3. number1 = pop number_stack = 3 (contents now = 2).
    4. value = 3 * 4 = 12.
    5. push value on number_stack (contents now = 2, 12).
11. operator_stack is not empty, and precedence of '/' is greater than '+', so push '/' on the operator_stack (contents now = '+', '/').
12. etc.

I recommend you use subroutines for your program, and reserve the system stack for this purpose (calls and returns, passing parameters, etc.) In homework #5, I will expand the scope of this problem (expressions will use parentheses, other operators, etc.), which will justify the use of stacks. Therefore, your "operator_stack" and "number_stack" data structures will have to be defined in normal memory, and you will have to program code equivalent to "push" and "pop" for these data structures (hint: use a macro for this purpose).

GRADING

This homework is graded on a scale of 100 points; the points for each problem are shown above. The homework score will be weighted to contribute 5% of your course grade.

Problems will be graded according to the following interpretation:

- All right = full credit
- halfway to mostly right = half credit
- Not much or none right = no credit