CSE 214 - Data Structures
Fall 1997
Program #2

Due: Monday, October 6, 1997

Pocket calculators are great tools for doing arithmetic, but unfortunately they have certain limitations. In particular, most calculators can only work with integers of at most 8-12 digits, before having to convert to scientific notation and a loss of precision. In this program, we will build a calculator which can work with arbitrarily large integers, so we can compute on *very* big numbers.

Our calculator will use reverse Polish notation, just like HP calculators. Reverse Polish notation the numerical values are pushed on a stack, and each operation pops off the top two operations and pushes down the result. In our complete program, the following commands must be supported:

command	meaning
+	addition (eg. 2 4 + is 6)
-	subtraction (eg. 3 5 - is -2)
=	print total
Q	quit (end program)

I recommend representing each integer by a linked list of individual digits, with the least significant digit pointing to the next least significant digit, and the most significant digit pointing to nil. The sign bit should be kept as a separate field in the record. Addition and subtraction are both best done starting from the least significant bit first. It is also possible to represent each integer by a doubly linked list.

Your program should read the standard input and keep the current total on top of the stack. Once an '=' operator is received, the top element of the stack should popped and printed.

To allow easy processing of input, the unary negative operation will not be supported, so to enter a negative number it will be necessary to explicitly subtract from zero (eg. 0 4 - to enter -4). At least one space will separate each command. All integers are to be entered and printed without commas, and any other symbols should produce an error message. In printing a long number, make sure to explicitly insert line breaks after each 72 digits.

Sample input and output (output lines begin with ':'):

```
123  5 + =
:128
99999999999999999999999 99999999999999999999 1 + + =
:10009999999999999999999999
17  2  A =
:error in input
14 19 + -
:operator stack empty
 3  7  -  =
:-4
q
```

If an error occurs, print out an error message, skip through the input until an '=' or 'q' is found, reset your stack, and continue processing.

Challenge Problems

Each assignment will contain one or more extra credit problems for students who want a challenge, or to pursue the subject in greater depth. NOTE THAT THESE PROBLEMS ARE NOT REQUIRED.

(1) Can you extend your infinite precision calculator to more powerful operations? The addition and subtraction routines could be used repeatedly for multiplication and division, but doing repeated addition or subtraction will be too slow to permit working with large numbers. Thus you should use the standard algorithm where each digit in the second number is multiplied individually by the first number, with the results added together after being padded by the appropriate number of zeros. For division, simulate long division using the algorithm you used in school. For exponentiation, repeated multiplication will be too slow for large exponents, so use partial products for faster exponentiation. For example, to compute 2^{50} use 2×2 to get 2^2, $2^2 \times 2^2$ to get 2^4, $2^4 \times 2^4$ to get 2^8, and so on. You may treat exponentiation where the second operand is a negative number as an error.

(2) Can you implement your arithmetic algorithms so they calculate in a higher base, say *maxint*/2, instead of base 10? This should dramatically increase the speed of your calculator. You will thus need a routine to convert integers between your base and base 10.

Rules of the Game

(1) The main purpose of this assignment is to give you more experience with pointers and dynamic data structures. You must have no explicit constants to limit the size of any integers. Thus even the stacks must be implemented as linked lists.

(2) This program is not easy, and you only have a little over two weeks to complete it. Most of you do not stand a chance unless you start early. As discussed below, you should build it in stages, and set realistic deadlines for the completion of each stage. DO NOT START ON THE (I+1)ST STAGE UNTIL YOU ARE CERTAIN YOUR PROGRAM CORRECTLY SATISFIES THE ITH STAGE REQUIREMENTS!!

Stage 1 - implement a working version of the calculator front end, using stubs and doing computations on conventional integers to test it out. You should definitely aim to complete this part by September 26th to allow enough time to build the integer package.

Stage 2 - add arbitrary precision addition to the calculator. You should definitely aim to complete this part by October 1st allow time for completing the rest of the calculator. I expect all students to be able to complete this stage.

Stage 3 - add arbitrary precision addition to the calculator. This is required to get full credit on the project.

Stage 4 - add arbitrary precision multiplication and exponentiation to the calculator. This is a challenge problem, and hence optional, but it might be good for your soul to try it. This should not be *too* difficult once you have addition and subtraction working.

(3) To judge whether your program passes a stage, try it on the test files "~cse214/Fall1997/stage1in", "~cse214/Fall1997/stage2in", "~cse214/Fall1997/stage3in". To verify that the numbers you get are right, you can use the UNIX calculator dc, which is a very similar to the proposed calculator.

(4) You will be graded according to the highest number stage which is *correctly* working. A complete program which crashes on the ith stage is not worth more credit than a program which only attempts up to the $(i-1)$th stage, but does it correctly. Thus make sure to keep a separate copy of the program once it correctly handles each stage.

(5) For full credit, your program must use separate modules with interface files for (1) the stack type and (2) the long integer type.

(6) Don't hesitate to ask me or the TAs to explain how the arithmetical algorithms should work. Without a clear understanding of this, your program will never work correctly.

(7) Don't forget to write out and sign the pledge. ''On my honor as a student I have neither given nor recieved aid on this program''. Your program will not be graded without it.