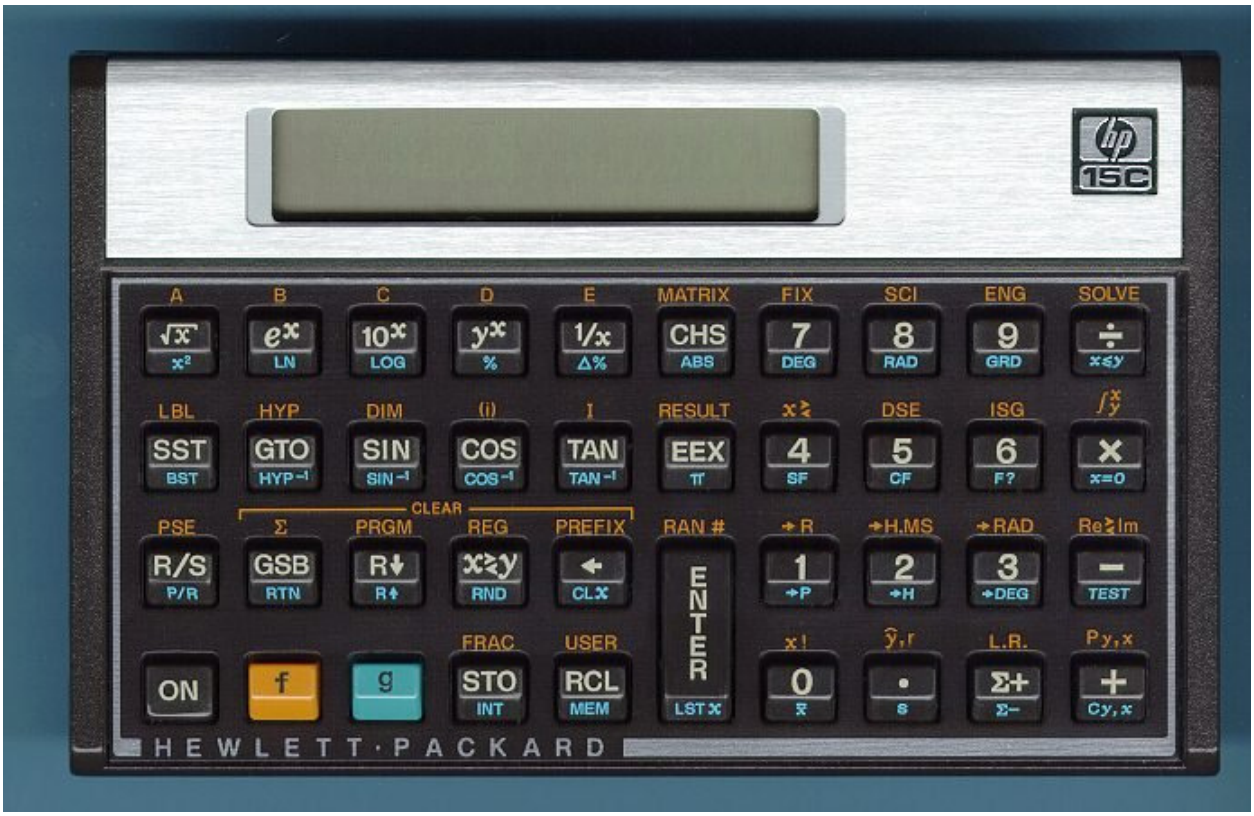
$Id: lab3c-rpnstack-array.mm,v 1.54 2016-01-11 21:47:45-08 - - $
PWD: /afs/cats.ucsc.edu/courses/cmps012b-wm/Labs-cmps012m/lab3c-rpnstack-array
URL: http://www2.ucsc.edu/courses/cmps012b-wm/:/Labs-cmps012m/lab3c-rpnstack-array/



In this lab, you will be introduced to ANSI C11 and make use of a stack data structure implemented as an array. Input will be read using **`scanf`**(1) and output printed with **`printf`**(1). The program will implement a Reverse Polish notation calculator.

1. Reverse Polish notation

Reverse Polish notation[1] (RPN) was invented by the Polish logician Jan Łukasiewicz[2] and places all operators after their operands, which makes it much easier to parse than infix notation, and does not need parentheses to override operator precedence. For example, the infix expression

```
3 * 4 + 5 * 6
```

is represented as

```
3 4 * 5 6 * +
```

in RPN.

1. **`http://en.wikipedia.org/wiki/Reverse_Polish_notation`**

2. **`http://en.wikipedia.org/wiki/Jan_%C5%81ukasiewicz`** — The notation **`%C5%81`**, is a two-byte UTF-8 representation of the Unicode character **`U+0141`**, Latin capital letter L with stroke (Ł).

2. HP-15C RPN Scientific Calculator

A Free HP-15C RPN Scientific Calculator Simulator is available in the directory `HP15C-Calculator/`.[3] The Java Swing and Javascript versions have been downloaded. Other versions are available at the original site.[4] To run the locally stored Java version :

```
cd Labs-cmps012m/lab3c-rpnstack-array/HP15C-Calculator/
HP15C.jar &
```

If you prefer to use the web version, starting with the URL for this lab, click on `http:HP15C-Calculator`, then click on the image of the calculator, which will then bring up a Javascript web emulator.

3. Program input

Input to the program will consist of double precision IEEE-754[5] floating point numbers and calculator operators. You have been provided with a reference implementation `jrpn.java` which you are to translate into `crpn.c`, for which a skeleton outline has been provided.

(i) Any word on input that looks like a number to the function `strtod`(3) is pushed onto a stack.

(ii) If an input word consists of one of the operators `'+'`, `'-'`, `'*'`, or `'/'`, two numbers will be popped off the stack, the right operand being popped first and the left operand next, then the result is pushed back onto the stack.

(iii) The operator `';'` cause all numbers on the stack to be printed, from bottom to top using the format `"%.15g\n"`.

(iv) The operator `'@'` clears the stack.

(v) Any input word beginning with the character `'#'` causes the rest of the line to be ignored as a comment.

(vi) Anything else on input is an error.

4. Outline of the functions

The following functions are to be implemented. See the Java version for detailed implementation requirements.

(i) `void bad_operator (char *oper)`
takes an invalid operator and prints an error message.

(ii) `void push (struct stack *stack, double number)`
pushes the number onto the stack if there is space but prints an error message if not.

---

3. `http:Labs-cmps012m/lab3c-rpnstack-array/HP15C-Calculator/`

4. `http://hp15c.com/`

5. `http://en.wikipedia.org/wiki/IEEE_floating_point`

(iii) `void do_binop (struct stack *stack, char operator)`
accepts a binary operator, pops the right operand then the left operand, computes the answer then pushes the result on the stack. An error message is printed if there are not at least two numbers on the stack.

(iv) `void do_print (struct stack *stack)`
prints the stack bottom to top, one number per line, or indicates that the stack is empty.

(v) `void do_clear (struct stack *stack)`
causes the stack to be emptied.

(vi) `void do_operator (struct stack *stack, char *operator)`
accepts any operator and dispatches the appropriate function to perform the operation.

(vii) `int main (int argc, char **argv)`
loops reading input and calls other functions to perform appropriate operations.

5. The `code/` and `misc/` subdirectories

The `code/` subdirectory contains two programs: `jrpn.java` contains the implementation of your lab in Java. `crpn.c` contains a partial implementation of your lab in C. The `main` function is complete, and you have to fill in the bodies of the rest of the functions to that they do exactly the same thing as the Java version. The exact format of the numbers printed will be slighly different because of the way the languages format numbers differently.

The `misc/` subdirectory contains various sample programs illustration some facets of ANSI C11.

6. Compiling your code

Do not type `gcc` in at the terminal directly, since it is properly used with multiple options, which would normally be put in a `Makefile`. This time, you will write a simple shell script called `mk` to perform the compilation. The script should contain the following lines:

```
#!/bin/sh
# $Id$
gcc -g -O0 -Wall -Wextra -std=gnu11 crpn.c -o crpn
```

Add your name after the RCS Id comment line in another comment. Use `chmod +x` to make it executable. When you have created the file, check it into an RCS subdirectory using `cid`. Also do this with your program `crpn.c`. Note: The hash-bang line (starting with `#!`) may not have any spaces.

7. What to submit

Submit the script `mk` and the program `crpn.c`, and if you are doing pair programming, also the `PARTNER` file.