

Practical 3: HP-35

Task

Write a program to emulate calculations performed on the HP-35 calculator.

SVN check out

Before you begin, you will need to check out the practical directory from the topic repository. This can be achieved by selecting Get from Version Control on the CLion welcome screen. In the window that appears, select Subversion from the dropdown list. If you have not previously connected CLion to the repository, click the add button (+) and paste the following URL into the text field (replacing **FAN** with your FAN):

<https://topicsvn.flinders.edu.au/svn-repos/COMP2711/FAN>

If you are prompted to login, use your University FAN and password. Expand the repository listing and select the **hp-35** directory. Click Check Out then select a location to store your project (preferably in a practicals directory) and click Open. From the destination list, choose the second option to create a project directory named **hp-35** and then click OK to complete the check out. A dialogue may appear confirming the subversion working format; if so, 1.8 is fine. Finally, when prompted if you would like to open the project, click Yes.

Automated marking

This practical is available for automatic marking via the quiz **Practical 3**, which is located in the Assessment module on FLO. To assess your solution, first commit your changes to the topic repository. This can be completed within CLion via the Commit option under the VCS menu (or alternatively with $\text{⌘}+K$ on macOS or $\text{Ctrl}+K$ on Windows).

You should adhere to good development habits and enter a commit message that describes what you have changed since your last commit. When ready, click Commit to upload your changes and receive a revision number. Enter this revision number into the answer box for the relevant question (level).

There are no penalties for incorrect solutions, so if you do not pass all test cases, check the report output, modify your solution, commit, and try again. Remember to finish and submit the quiz when you are ready to hand in. You may complete the quiz as many times as you like—your final mark for the practical will be the highest quiz mark achieved. If you do start a new quiz attempt, ensure you reassess any levels you have previously completed.

Background

The HP-35 was the first scientific handheld calculator, released in 1972 and an immediate success with engineers and scientists. Amongst other features, it used an unusual way of entering numbers, called RPN. RPN (reverse Polish notation, named for Polish mathematician Jan Lukasiewicz) allows complex calculations to be entered without the use of parentheses. The key difference is that operators are entered after the operands rather than between them.

The HP-35 implemented RPN using a stack of 4 registers, referred to as X, Y, Z, and T. The value in the X register is the one shown on the calculator's LED display; the other registers are used for temporary values during calculations. New values are entered into the X register, with existing values *pushed* up the register stack: the old value of X goes into Y, the old value of Y goes into Z, the old value of Z goes into T, and the old value of T is lost.

Single-operand functions, such as square root, use and replace the value in the X register; other registers are unchanged. Functions that operate on 2 values, such as the arithmetic operators, use the values in Y and X, with the result going back into X. Values in the Z and T registers are *dropped* down the stack with Z overwriting the old Y and T overwriting the old Z. The value of T is preserved, providing a simple way to do calculations with repeated values.

This scheme allows calculations to be entered without using parentheses. For example, you could compute the compound expression $(5 + 4) / (3 - 2)$ like this:

5 ENTER 4 + 3 ENTER 2 - ÷



Hints

You will need to implement the class **HPStack** to represent the calculator's operand stack, with operations such as **push** (push all values up 1 level and store a value into X), **pop** (drop all values down 1 level and return the old X value—note that **T** is not cleared), and **peek** (return the current X value). The simplest approach is to use an array to store the register values with the methods manipulating the array elements.

The calculator should read and process tokens representing numbers and operator keys, with the current value of the X register displayed after processing each line. It should exit when all lines of input have been processed.

Level 1: Basic

Implement the arithmetic operations so that the calculator can compute basic expressions:

```
5 4 + 3 2 - /           // should be 9
3 4 5 6 + + +           // should be 18
1.23 4.5 67 / *         // should be 0.0826119
```

Level 2: Scientific

Implement π (**pi**) and the functions **chs** (change the sign of X), **1/x** (**recip**), **log** (decimal logarithm), **ln** (natural logarithm), **e^x** (**exp**), \sqrt{x} (**sqrt**), **sin**, **cos**, **tan** (and their inverses **arcsin**, **arccos**, and **arctan**), and **x^y** (**pow**). The function names should be accepted in either upper or lower case. Use the functions in the **cmath** library to do the actual work. The predefined constant **M_PI** contains the value of pi.

```
PI 2 / SIN               // should be 1 (or very close)
100 100 * sqrt           // should be 100
1E-5 LOG CHS             // should be 5
5 EXP LN                 // should be 5
3 recip 8 pow            // should be 2
2 1 arctan cos pow       // should be 0.5
```

Level 3: Memory

Implement **ST0** (store X into memory) and **RCL** (recall from memory into X, pushing other values up), **CLR** (set all registers to 0), and **CLX** (drop the X value).

```
100 sto 2 * rcl /        // should be 2
1 2 clx 3 +              // should be 4
1 2 3 4 clr + + +        // should be 0
```

Level 4: Stack control

Implement **x \leftrightarrow y** (**swap** X and Y), **R \downarrow** (**roll** the stack down with each register moved into the register below it and X moved into T), and **ENTER \uparrow** (push all values up the stack, preserving X and discarding T). Note that this interpretation of **ENTER** is not identical to that on the original calculator where it is also used to separate consecutive numbers.

```
2 8 swap /              // should be 4
1 2 3 4                 // should be 4
roll roll               // now 2
roll roll               // back to 4
2 enter enter enter     // 2
+ + + +                 // 10. If you do not get 10, check your pop function logic.
```