# Workshop 09 - Programming in Jack

### Programming in Jack

##### [Practical Marker](https://cs.adelaide.edu.au/services/pracmarker/)

#### Docker

**Note:** the docker images csadelaide/cscli and csadelaide/csvnc have been updated to include the Simulators.jar, Unix.class and DP.class files attached below.

#### Unix Environment

This workshop assumes that you are able to run the Nand2Tetris tools from the command line, the Jack Compiler only runs from the command line. For this to work you need to ensure that your shell program knows where to find the tools.

On Unix/Linux/OS X systems the shell programs look for an environment variable named PATH that provides a colon (:) separated list of directory names. Every time you enter a command, the shell will look for an executable script or program with that name in each of these directories. It will run the first one that it finds. So to run the Nand2Tetris tools, you first need to add the name of the directory containing your tools directory to the environment variable PATH.

Assume that you have placed your Nand2Tetris software in the directory: */users/bob/courses/nand2tetris* which means that the tools directory will be called: */users/bob/courses/nand2tetris/tools*.

If you are using the **bash** shell, you should edit the file **~/.bash\_profile** and add the following line:

export PATH=$PATH:/users/bob/courses/nand2tetris/tools

If you are using the **tcsh** shell, you should edit the file **~/.cshrc** and add the following line:

setenv PATH ${PATH}:/users/bob/courses/nand2tetris/tools

Once you have made the appropriate edit, the Nand2Tetris tools will be available next time you login. To use the tools immediately, just cut and paste the appropriate **export** or **setenv** command into your terminal.

The last thing you need to do is make the tools commands executable. Change to the tools directory and type the following two commands:

% chmod 644 \*.bat  
% chmod 755 \*.sh

These commands make the Linux versions of the commands executable but not the Windows versions. The commands shown below should now work as shown.

#### Windows Environment

On a Windows system you also need to set the PATH variable but how this is achieved varies a bit between systems. The directory names are also separated by semi-colon (;) rather than colon (:).  Instructions on how to set the PATH variable in Windows can be found here:

[http://www.computerhope.com/issues/ch000549.htm (Links to an external site.)Links to an external site.](http://www.computerhope.com/issues/ch000549.htm)

#### Compiling Jack Programs

To write a Jack program first create an empty directory where your program will be written. This directory will contain all the **.jack** files you create, one for each class in your **Jack** program.

For each class in your **Jack** program create a file in your directory that starts with the class name and is suffixed by **.jack**. Once your have written a class, use the command **JackCompiler.sh** to compile the **.jack** file and create a matching **.vm** file. At least of the classes must be called **Main** and contain a function called **main()**.

You can compile a single source file, **Ball.jack**, with the following command:

% JackCompiler.sh Ball.jack

You can compile every **.jack** file in the current directory with the following command:

% JackCompiler.sh

#### Running Jack Programs in the VMEmulator

Once you have a compiled **.vm** file you can run your program by loading it into the Virtual Machine emulator. Run **VMEmulator.sh** with no command line arguments and use the "Load Program" entry in the file menu to choose the **Main.vm** file. If the directory does not contain all the Jack OS classes used by your program, the VMEmulator will ask if it can substitute any missing classes with its own builtin classes. There is a builtin class for each of the Jack OS classes. Once loaded you can run the program from within the VMEmulator GUI.

If your program consists of more than one class, you need to choose the directory containing the program rather than the **Main.vm** file. When using the "Load Program" entry in the file menu, go to the parent of the directory containing your program, selected the program directory, then click the "Load Program" button. This will load every **.vm** file in your directory on the assumption that it is part of your program. Do not have classes from more than one Jack program in the same directory.

#### Running Jack Programs on the Command Line

An alternative to using the GUI version of the VMEmulator is to run it with a test script. Two test scripts are attached below. The first only loads the **Main.vm** file whilst the second will load every **.vm** file in your directory. In both cases the program will run until it chooses to exit.

To run a program that consists of just the Main.vm file, change to that directory and type:

% VMEmulator.sh runMain.txt

To run a program that consists of several **.vm** files, change to that directory and type:

% VMEmulator.sh runAll.txt

These scripts will run the program as above but you will not be able to see any output or provide input because the GUI is not being used. To overcome this, you can access standard input and output using our **Unix** class which is described below.

**Note:** in order to use these scripts you must install a new version of the **Simulators.jar** file which is attached below. Copy this file into the *tools/bin/lib* directory of your Nand2Tetris software.

**Question 1 - Jack Programs in the GUI**

Try to write the following Jack programs and running them in the VMEmulator's GUI.

Write, compile and run a Jack program to print Hello World.

Write, compile and run a Jack program to prompt for and read in two numbers and print out their sum.

Write, compile and run a Jack program to read in a series of numbers and print out their average and standard deviation. You will need to use the **DP** class for some of the arithmetic.

#### Question 2 - Jack Programs on the Command Line

Now try to run the same programs by running them from the command line. Note that you will need to modify the program to use the **Unix** class for reading and writing.

Run a Jack program to print Hello World.

Run a Jack program to prompt for and read in two numbers and print out their sum.

Run a Jack program to read in a series of numbers and print out their average and standard deviation.

### Extended Jack OS

In order to enable I/O when running Jack programs on the command line and to support some useful arithmetic operations, we have provided two extra classes for the Jack OS, Unix and DP.

#### Unix

This class provides basic I/O functions that allow reading/writing of standard input/output and error. To use this class, change to the directory containing your Nand2Tetris software and copy the Unix.class file attached below into the tools/builtInVmCode directory.

##### Standard Input

* + public static int readChar(): reads a character from standard input, it returns -1 on I/O errors.
  + public static String readLine(short message): writes the message to standard output, then reads a line from standard input
  + public static int readInt(short message): writes the message to standard output, then reads a line from standard input and converts it to an int.

##### Standard Output

* + function void printChar(char c): writes the character c to standard output.
  + function void printString(String s): writes the String s to standard output.
  + function void backSpace(): writes a backspace character to standard output.
  + function void printInt(int i): writes a newline character to standard output.
  + function void println(): writes a newline character to standard output.

##### Standard Error

* + function void printCharErr(char c): writes the character c to standard error.
  + function void printStringErr(String s): writes the String s to standard error.
  + function void backSpaceErr(): writes a backspace character to standard error.
  + function void printIntErr(int i): writes a newline character to standard error.
  + function void printlnErr(): writes a newline character to standard error.

##### Useful Extras

* function String strcpy(String s): this returns a copy the string s.
* function int strcmp(String s1, String s2): this returns -1 if s1 < s2, or 0 if s1 == s2, or 1 of s1 > s2.
* function void exit(int exit\_status): the process exits with the specified exit status.

#### DP

This class provides double precision, DP, floating point arithmetic functions A DP object is an array of 4 integers in order to achieve a 64-bit representation of the DP floating point number. To use this class, change to the directory containing your Nand2Tetris software and copy the DP.class file attached below into the tools/builtInVmCode directory.

##### Creating / Deleting DP Objects

* + constructor DP new(int i): constructs a new DP object and initialises it with the value i.
  + method void dispose(): frees the memory allocated to this DP object.

##### Copy, Parse, E and PI

* + function void assign(DP L, DP R): copies the value of R into object L.
  + function void parse(DP d, String s): parses the number in String s and copies its value into object d.
  + function String toString(DP d): returns a string representation of object d's number.
  + function void E(DP d): copies the mathematical value E into object d.
  + function void PI(DP d): copies the mathematical value PI into object d.

##### Comparisons

* + function boolean lt(DP L, DP R): returns true if this object L's value is less than R's value.
  + function boolean lteq(DP L, DP R): returns true if this object L's value is less than or equal to R's value.
  + function boolean eq(DP L, DP R): returns true if this object L's value is equal to R's value.

##### Basic Arithmetic

* + function void add(DP a,DP b, DP c): copies the result of b + c into object a.
  + function void subtract(DP a, DP b, DP c): copies the result of b - c into object a.
  + function void multiply(DP a, DP b, DP c): copies the result of b \* c into object a.
  + function void divide(DP a, DP b, DP c): copies the result of b / c into object a.
  + function void neg(DP a, DP b): copies the result of -b into object a.

##### Java Math.\* Functions

* function void abs(DP a, DP b): copies the absolute value of b into object a.
* function void sqrt(DP a, DP b): copies the square root of b into object a.
* function void log10(DP a, DP b): copies the base 10 logarithm of b into object a.
* function void log(DP a, DP b): copies the natural logarithm of b into object a.
* function void exp(DP a, DP b): copies the result of E raised to the power b into object a.
* function void power(DP a, DP b, DP c): copies the result of b raised to the power c into object a.