|  |  |
| --- | --- |
| Stacks and Postfix Notation Using Forth |  |
| |  |  |  | | --- | --- | --- | | **Word or Symbol** | **Description** | **Examples** | | .s | Non - destructively displays the contents of the stack | .s | | . | Outputs the value currently on top of the stack, and removes it from the stack. | 25 11 . ( outputs 11, 25 left on stack) | | + | Adds together the two items currently on top of the stack, and leaves the result on top of the stack | 3 7 + . ( outputs 10 ) | | - | Subtracts the item that is on top of the stack from the item before it. | 5 3 - . ( outputs 2 ) | | \* | Multiplies the two items on top of the stack together, and leaves the result on the stack | 5 4 \* . ( outputs 20 ) | | / | Divides the item that is second from the top of the stack by the item that is on top of the stack, leaves the resulting integer on top of the stack. | 12 3 / . ( outputs 4 ) | | mod | Divides the number that is second from the top of the stack by the number that is top of the stack, leaves the remainder on top of the stack | 13 3 mod . ( outputs 1 ) | | depth | Returns a count of the number items on the stack. | 0 1 depth ( stack now contains 0 1 2 ) | | dup | Duplicates the value currently on top of the stack. | 25 dup ( stack now contains 25 25 ) | | swap | Swaps the two items currently on top of the stack | 9 101 swap ( 9 is top of the stack now) | | drop | Removes the item currently on top of the stack | 5 7 drop ( only 5 is left on the stack) | | = | Compares the two items currently on top of the stack. If equal, leaves true (-1) on top of the stack, if not equal leaves false (0) on top of the stack. Not equals is <>. | 12 11 = ( will leave 0 on the stack) | | hex | Any numbers being input and output will be interpreted as hexadecimal. | hex 10 ( puts 16 decimal on stack) | | decimal | Any numbers being input and output will be interpreted as decimal. | decimal 10 ( puts 10 decimal on stack) | | ." | Print a string literal | ." Hello world" | | cr | Print a new line character | cr | | : ; | Starts and ends the definition of a user defined subroutine (in Forth called a word) | : doNothing dup swap drop ; ( n -- n ) | | ( ) | Anything between the brackets is ignored, allowing you to add comments to your code. It is conventional when writing Forth programs to note what effects your code will have on the stack. In the example to the right, two numbers n1 and n2 are expected to be found on the stack when the code runs, and the code will leave a single item on the stack which is the sum of the two numbers. | ( Add two numbers together n1 n2 -- sum) | | |

|  |  |
| --- | --- |
| Code examples |  |
| |  |  | | --- | --- | | **Example** | **Description** | | : question **if** ." It's true!" **else** ." No, it's not true." then ;  false question  >No, it's not true. ok  true question  >It's true! ok | A new subroutine is defined called question. It expects to find a value on the stack that is either -1 or 0. In Forth 0 is the value for false and -1 is the value for true.  Testing the question subroutine involves putting a true or false value on the stack and calling the routine by its name. | | : bigHello 10 0 **do** cr ." Hello!" **loop** ;  bigHello  Hello!  Hello!  ...  Hello!  Hello! ok | A count controlled loop is used here to print a new line character followed by the string Hello! Note how ." has to be followed by a space in order for the Forth interpreter to recognise where the command ends and the string starts. This item on top of the stack must be smaller than the item below it. | | : multiples cr 11 1 **do** dup **i** \* . **loop** drop ;  7 multiples  > 7 14 21 28 35 42 49 56 63 70 ok | To make use of the index variable i, reference it inside the loop and the current value will appear on the stack.  In this example i goes from 1 to 10, and is multiplied each time around the loop.  As the initial value will be consumed in the multiply, it is duplicated before the multiply operation so it will be available for the next iteration. The initial value is dropped from the stack before the code exits. | | : keyValues **begin** key dup . 113 = **until** ; ok  keyValues  >51 110 65 81 113 ok | A condition controlled loop which tests at the bottom of the loop. It starts with the keyword begin and ends with the keyword until. The test just before the until will determine when the loop ends. In this case, each time the loop runs a key is expected to be typed at the keyboard, its ASCII value is printed and this repeats until a lower case q (ASCII 113) is typed. | | : stars **begin** dup 0 > **while** ." \*" 1 - **repeat** drop ;  5 stars \*\*\*\*\* ok  -1 stars ok | A condition controlled loop which tests at the top of the loop, which means that the code inside the loop may never run.  In this example the value on top of the stack is compared to 0. While it is greater than zero, the loop will execute printing an asterisk and subtracting one from the value on top of the stack. It will terminate when the value on top of the stack is zero, dropping the value off the stack before returning. | | : in5s 50 0 do i . 5 +loop ;  in5s  >0 5 10 15 20 25 30 35 40 45 ok | The +loop word allows you to increment a loop counter in steps other than 1. In this case the counter i increments from 0 to 45 in steps of 5. A negative step allows you to count backwards. | | : question **if** ." It's true!" **else** ." No, it's not true." then ;  variable rumour  false rumour !  rumour @ question  >No, it's not true. ok | As Forth tends to use the stack for parameters and return values, you can actually write many simple programs without using variables at all. Variables generally are only used where data needs to be stored for a longer period of time or if hardware is being accessed.  The example code shows the question subroutine defined in an earlier example. This time however the value being put onto the stack is retrieved from a variable.  lines that follow define a variable called rumour, and store the value false into that variable. The final line takes the contents of the rumour variable and places it on the stack, finally calling the question subroutine. | | sp0 @ ( gives the address of the bottom of the stack)  sp@ ( gives the address of the top of the stack)  r@ ( fetches the item on top of the return stack and puts onto data stack) | These will not work in repl.it but if you want to explore further using Gforth, these words will respectively give you a pointer to the address of the bottom of the stack (which can be used to see if the stack is empty), the top of the stack (which can be used to retrieve the current value on top of the stack) and the return stack pointer (which is used for return addresses and also index values in loops). Note the stack in most systems grows downwards, i.e. the bottom of the stack is in a high memory location, and the stack pointer decreases as items are added to the stack, hence as the stack grows the top of the stack will be at a lower memory location. | | |