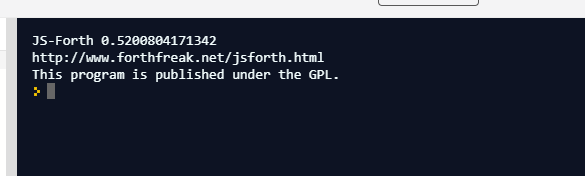
Visit <https://repl.it/languages/forth>, or simply <https://repl.it/> and choose the language Forth from the list of languages that appear at the bottom of the screen.



You can type the commands directly into the console on the right hand side of the screen to see their effect.

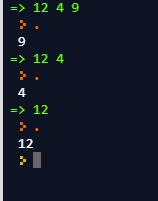
Putting (or pushing) a number on the stack is as simple as just typing the number and pressing enter.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  | | --- | | **Top of stack** | | 9 | | 4 | | 12 | |
|  |  |

Each time you add a number to the stack a green arrow appears showing you the contents of the stack. In the screenshot we see **12** was added to the stack when it was empty so we see **12** alone in green. Then when **4** is typed in, the **12** remains on the stack, but **4** is added on top.

Finally, **9** is entered, and the stack now contains **12, 4** and **9,** with **9** at the top of the stack. The stack is displayed on its side in this view and the arrow is pointing in the direction the stack will grow. It will only show the first 5 items on the stack, if there are more than 5 **...** signifies the items not being displayed.

Typing **.s** also displays the stack, and will display every item not just the first 5. Try entering the numbers as in the screenshot above.

If you make a mistake when typing and the Forth environment doesn't understand what you want to do, it will give an error then empty the stack.

Entering a single **.** and pressing enter will take the item currently on top of the stack off the stack (pop), and print its value. Do this repeatedly and you will then see the stack getting smaller each time until there is nothing left.

What happens if you try to take an item off an empty stack? Try it.

So far we have done nothing useful with the stack. Let's now use it as it was intended: to store values to be acted on by other operators or procedures in a program.

The slightly strange thing about Forth is it uses **postfix notation** (also known **as Reverse Polish Notation**). This means that the numbers precede the operator that is going to act on them.

We are used to using **infix** notation like this: 3 + 4 which means that the operator (+ in this case) is in between the two numbers being used in the calculation. With postfix, the operator follows after the numbers so the equivalent calculation would be 3 4 +

Weird, huh? But it turns out that this is a much simpler way for compilers to evaluate mathematical expressions and has another handy advantage we will discover.

Typing 3 4 + into the Forth interpreter will leave the result 7 on the stack as you would expect.

## Task 1

Now, use the stack to do the following calculations in postfix notation. If correct, write the expression next to the question below.

1. 12 + 2 12 2 +
2. 7 + 15 + 31 7 15 + 31 +
3. 67 - 23 67 23 -
4. 8 \* 7 8 7 \*
5. 24 / 6 24 6 /
6. (3+9)\*4 3 9 + 4 \*
7. (3+9) \* (4+6) 3 9 + 4 6 + \*
8. ((12-4) / 4 ) + 3 12 4 - 4 / 3 +
9. ( ((3\*9) - 7 )/ 4)+2 3 9 \* 7 - 4 / 2 +
10. (12 + 3 +7 ) / (17 - (3\*2)) 12 3 + 7 + 17 3 2 \* - /

Can you see an advantage to post fix notation?

## Task 2

Tree traversal is the process of visiting the nodes in a special type of graph called a tree. Compilers make use of trees to evaluate mathematical expressions in order to check the syntax and then generate the correct machine code instructions. An example of a tree can be seen below.

When visiting the nodes of the tree **in order**, we draw a dot **under** each node in the tree, and then trace out the dots moving from left to right starting at the root, but only if we do not need to cross a line.

The order we have visited the nodes using in order is therefore:

3 + 8 \* 5

Post order traversal involves drawing a dot **to the right** of each node, and following as above.

2 a) What would be the order for post order traversal for the above tree?

3 8 5 \* +

2 b) Type the post order traversal into Forth and see what result you get. 43. Is it the same as the infix? Yes.

2 c) Now do an in order traversal of the tree below. Is it clear what the result of the expression would be?

It evaluates to -1, but is tricky to follow, as order of precedence needs to be worked out.

2 d) Do a post order traversal of the tree below, and type into Forth to see what result you get.

8 4 2 - / 2 1 + - gives a result of 1

## Task 3

Typical stack operations in a high level language would include:

push to put an item on the stack

pop to remove an item from the stack

peek to return the item from the top of the stack without removing it

isEmpty to return a boolean telling us whether the stack is empty or not

isFull to return a boolean telling us whether the stack is full or not

The above operations would be trivial to implement in Forth given that the stack is fundamental to the language.

To create a new function in Forth (called a word) you simply put a colon followed by the name of the function. You can then add your function code and finally terminate it with a semi-colon.

Because all parameters are expected to be on the stack when calling the function, defining push involves doing nothing!

: push ;

9 push

4 push

12 push

Type the code above into the editor in repl.it and click run. It will put 9 4 and 12 on the stack, with 12 being at the top.

3 i) Define a word for pop, to remove and print out the value on top of the stack

3 ii) Define a word for peek, to print out the value on top of the stack without removing it. (Hint: you can create a copy of the value on top of the stack using the keyword dup).

3 iii) Define a word for isEmpty which leaves a 0 on the stack if the stack is empty, or -1 if it is not empty. (Hint: see the cheat sheet for details of the word depth and the comparison operator = )

Answers,

: push ( n -- n ) ;

: pop . ( -- ) ;

: peek ( -- n ) dup . ;

: isEmpty ( -- n) depth 0 = ;

## Task 4

A Forth word is the equivalent of a procedure or function in other languages. In most high level languages when we call a function or procedure, we pass it arguments inside brackets. Values are returned from functions and stored into variables or used as arguments to other functions/procedures, e.g.

number=randInt(1,6) // store into number a random integer between 1&6

print( random(1,6) ) // print a random integer between 1&6

In Forth, functions and procedures expect to find arguments on the stack, and functions will leave results on the stack.

For example, to create a word called double, which doubles any number passed to it, we do this:

: double 2 \* ;

3 double ( will leave 6 on the stack )

Create and test words to do the following:

1. Take two numbers and multiply them together

: mult \* ;

1. Take a number of centimeters and convert to millimeters

: cm>mm 10 \* ;

1. Take a number and test whether it is even ( leave -1 on the stack if it is, 0 if it isn't)

: isEven 2 mod 0 = ;

1. Take a number and multiply it by itself.

: square dup \* ;