**Stacks**

Table of Contents

[Function Stack 4](#_Toc66036286)

[Post-Fix Notation 5](#_Toc66036287)

[Post-Fix to Infix Conversion 6](#_Toc66036288)

[In-Fix to Post-Fix Conversion 8](#_Toc66036289)

A stack is a data storing method that follows the principle of ‘Last in First Out’. The data that is inserted last (most recently) is also the one that is removed first.

In Stack terminology, insertion is called push and removal is called pop. Thus, data can only be pushed to the end of the stack and popped from the end of the stack. In order to do this, we need to keep track of the end position. This is called the top of the stack. If the top tries to cross the array size or go below the 0th index, an error will display.

void push (int arr[], int arrLen; int item; int &top)  
{  
 if (top == arrLen) cout<<"Error. Stack is full.";  
 else  
 {  
 arr[top] = item;  
 top++;  
 }  
}  
  
void pop (int arr[], int &top)  
{  
 if (top == 0) cout<<"Error. Stack is empty.";  
 else  
 {  
 top--;  
 cout<<arr[top];  
 }  
}

C++

Other functions we can implement in relation to the stack are:

* bool isEmpty();
* bool isFull();
* void clear();
* int top();
* int size();

An example of a real-life use of stacks in the way compilers work. When we use braces in our code, every opening brace has to have a closing brace, and the order of the braces has to be correct ({[}] is incorrect usage). The compiler does this by adding every opening brace to a stack. When the corresponding closing brace is found, the opening brace is popped. If an incorrect closing brace is found, or more or fewer closing braces are entered, an error is thrown.

## Function Stack

When functions are used in programs, if we use a function inside a function, the outer function cannot end before the inner one does. To maintained this, every called function, at the time of the call, is added to a stack. Only when the function ends is it popped. This is also true for recursive functions.

## Post-Fix Notation

Normally, arithmetic equations are given like this:

The principle is that every operator has one operand on the left and one operand on the right. Although easy for humans, this notation, known as In-Fix Notation, is difficult to compute for computers. Instead, the computer converts this to Post-Fix Notation, which looks like this:

Here, the principle is that two numbers are taken, and the operation given by the operand following them is performed. For example, means , means and means . Since there are no braces, this is easier to handle for the computer. The way 2 digit or more numbers are handled may seem confusing on paper, but to the computer, each number occupies a separate memory address. So, it is not two numbers the computer works on, it is the values in two memory addresses. This makes the length of each number irrelevant. There is also something called Pre-Fix Notation which is not being discussed here.

## Post-Fix to Infix Conversion

To convert an equation from Post-Fix to In-Fix notation, we will be using the same method that a computer does while computing a Post-Fix equation. The computer handles Post-Fix Notation with the help of a stack. Whenever it encounters a number, it adds the number of a stack. When it encounters an operator, it pops two numbers from the stack, and places the first number of the right of the operator, and the second number on the left. It adds the result back to the stack.

For example, consider the following equation in Post-Fix Notation (the stacks are given horizontally instead of the normal vertical, so the ‘top’ is the right side):

The computer first encounters the three numbers, , and . It adds these to the stack, so the stack looks like this:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Next, it finds the operator, so it pops two elements, then . It places on the right of the operator and on the left. The result of is , so is added to the top of the stack.

|  |  |
| --- | --- |
|  |  |

Then it encounters the operator, so it again pops two elements, then . The result of is . This is added back to the stack.

|  |
| --- |
|  |

It encounters the number and adds it to the stack.

|  |  |
| --- | --- |
|  |  |

Finally, when the operator is found, and then are popped and the result, is put back in the stack.

|  |
| --- |
|  |

The end of the equation has been reached. The result is popped from the stack and printed. If at the end of the operation, there is more than one element in the stack, an error will be shown. If an operator is found while there are less than two elements in the stack, an error will be shown.

## In-Fix to Post-Fix Conversion

To convert from In-Fix notation to Post-Fix Notation, we will again be using a stack. This procedure is for human beings, not for computers. In the stack, instead of operands, we will be storing operators this time.

In mathematical equations, the priority of the operators is very important. In the equation , is not the same as . Humans know the priority order off the top of their head, but computers need to be told. Thus, the priority is defined for a program that performs this type of conversion. For all examples here, the priority will be considered to be .

Consider the equation below:

Every time we find a number (marked as alphabets here), we will take it directly to the Post-Fix equation, and every time we find an operator, it will be added to a stack. When adding an operator, if we find that the priority of the operator is lower than the operator currently on top, the entire stack will be popped and put into the equation, and then the current operator will be added.

We first encounter , and then the operator .

Equation:

Stack:

|  |
| --- |
|  |

­­­

Next, we find and then the operator . Since has a higher priority than , we add it to the stack.

Equation:

Stack:

|  |  |
| --- | --- |
|  |  |

Next, we find and add it to the equation. But the operator after it, , has a lower priority than , so we must pop each item from the stack one by one and add it to the equation before adding to the stack.

Equation:

Stack:

|  |
| --- |
|  |

Then we find , which we add to the equation, then , which is added to the stack since it has a higher priority than , and finally is added to the equation.

Equation:

Stack:

|  |  |
| --- | --- |
|  |  |

Since we have reached the end of the In-Fix equation, we will now pop each of the remaining operators in the stack one by one and add them to the end of the Post-Fix equation.

Equation:

Now consider an example that uses brackets. Remember that the priority of the bracket is lowest amongst all, and that the bracket is not actually going to be added to the equation (since it is not needed). It will only be added to the stack to maintain priority.

We first encounter two brackets, which we add to the stack, then , which goes to the equation, then which we add to the stack since it has a higher priority than the bracket, then , which we add to the equation.

Equation:

Stack:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Next, we find a closing bracket. When a closing bracket is found, we must pop all operands in the stack until an opening bracket is found. The opening bracket found is also popped, but not put in the equation. Thus,

Equation:

Stack:

|  |
| --- |
|  |

Continuing with the original equation, we find , which goes to the stack, and then , which goes to the equation. Thus,

Equation:

Stack:

|  |  |
| --- | --- |
|  |  |

Next, we encounter a operator. The operator has a lower priority than the operator, so we must pop all operators in the stack until an opening bracket is found. In this case, this means just the operator. We then add the operator after the opening bracket in the stack.

Equation:

Stack:

|  |  |
| --- | --- |
|  |  |

Next, we find , which we add to the equation, and then a closing bracket, which means we must again pop all operators until an opening bracket is found, and then pop the opening bracket as well.

Equation:

Stack:

|  |
| --- |
|  |

Next, we find a operator, which is add to the stack, and then , which we add to the equation. Since the end of the original equation has been reached, we must now pop all operators left in the stack, which consists of only the operator in this case.

Equation:

Keep in mind that if we reach the end of the original equation and we still have opening braces left in the stack, that means the original equation missed the corresponding closing braces and is thus incorrect.