**1. Problem Description :**

In this problem it is given that you will be provided with an infix expression and you are required to evaluate and print it's value. You don't need to worry about input; it is already managed for you.

But what is even an infix expression? :

An infix expression is a single letter, or an operator, proceeded by one infix string and followed by another infix string. x x + y (x + y) \* (a - b)

Basically it's just a fancy word for an algebraic expression which we have been studying since sixth or seventh grade. Only differences will be seen while solving these expressions because in this problem we will be considering the given constraints.

However there are also expressions called prefix and postfix expressions regarding which, we will be solving problems as we move further in this module. And those expressions are not in the form of a simple algebraic expression. We will deal with those problems only, not now.

Sample Input: 2 + 6 \* 4 / 8 - 3 Sample Output: 2

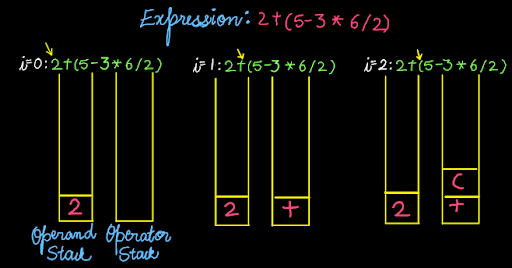
**2. Approach :**

Before trying to come up with an approach, let's try to analyse this problem and see where we can use a stack for infix evaluation. We know that we have to see the precedence of operators and brackets in mathematical expression before we try to evaluate it. So, to preserve this precedence we will use the stack. Now, let’s try to define more details for the stacks that we are going to use.

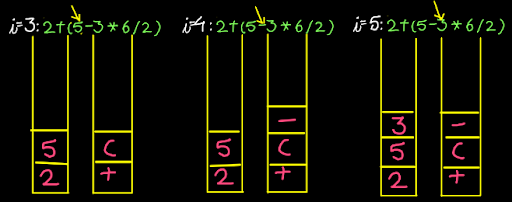
We will take two stacks one for operator and another for operand. Now while scanning the expression, as soon as we get an operand we push that in the operand stack. If we get an opening bracket while scanning the expression, we push that in the operator stack.

f we get a closing bracket while scanning the expression, we pop the items of the operator stack until we get an opening bracket. And as soon as we get an opening bracket we pop that out too. If an operator comes then all the operators in the operator stack with greater or equal precedence gets pop out until we get an opening bracket or the operator stack empties out. And then we push our current operator. And whenever an operator is popped out then at the same time two elements from the operand stack are also popped out and an operation is performed using two operands and one operator. While performing this operation, operand that is popped out second will be placed first and operand which is popped out first will be placed second i.e. after operator. Then the solved value needs to be pushed into the operand stack.

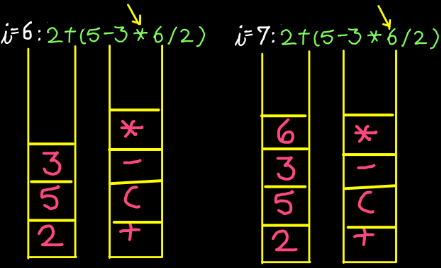
Now let's take an example, say we are given an expression: 2+ (5-3\*6/2)

****

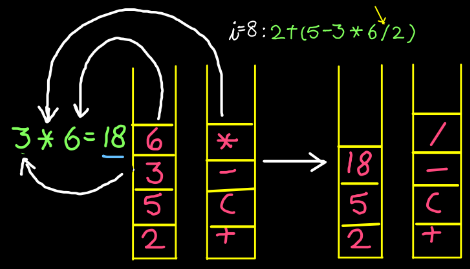
We push "2" into the operand stack and "+" into the operation stack because the stack is empty and none of the conditions is violated. "(", is pushed into the operation stack.

****

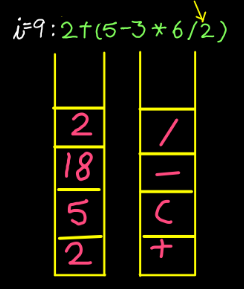
We push "5" into the operand stack and "-" into the operation stack because the topmost element of the operator stack is "(" and therefore none of the conditions is violated.

****

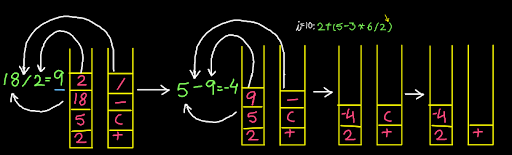
We push "3" into operand stack and "\*" into operation stack because the top most element of the operator stack is "-", which is of lower precedence. Then we push "6" into the operand stack.

****

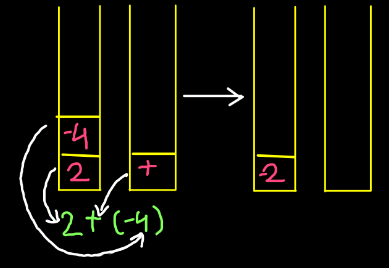
But before pushing "/" into operation stack, we check the top of the operation stack, there we find "\*" at the top which is equal in precedence to "/". So we pop "\*" and two topmost elements of the operand stack i.e. 6 and 3 in this case. Perform the operation using popped operation and 2 operands. Push the new value, i.e. 18 in this case into the operand stack.

****

Now we move to the next elements of expression, i.e. "2" which we push into the operand stack.

****

Next element is ")", on getting this we simply pop out the elements from both the stacks and perform the required operation as we performed before for multiplication. So in this case the topmost operator is "/" and 2 operands at the top are "2" and "18". We get "9" on performing this operation and therefore we push this into the operand stack. Now the topmost operator is "-" and 2 operands at the top are "9" and "5". We get "-4" on performing this operation and therefore we push this into the operand stack. After this, we find "(" at the top of the operand stack, we pop this out and. Now we empty out both the stacks by popping out elements out of these and performing operations.

****

Here we are left with only one operator which means only one operation is left to perform Now the only and topmost operator is "+" and 2 operands at the top are "-4" and "2". We get "-2" on performing this operation, which is our final answer.

**3. Code :**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new

InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(br.readLine());

}

int k = Integer.parseInt(br.readLine());

// nge begin

int[] nge = new int[arr.length];

Stack< Integer> st = new Stack<>();

st.push(arr.length - 1);

nge[arr.length - 1] = arr.length;

for (int i = arr.length - 2; i >= 0; i--) {

while (st.size() > 0 && arr[i] >= arr[st.peek()]) {

st.pop();

}

if (st.size() == 0) {

nge[i] = arr.length;

} else {

nge[i] = st.peek();

}

st.push(i);

}

// nge end

int i = 0;

for (int w = 0; w <= arr.length - k; w++) {

if (i < w) {

i = w;

}

while (nge[i] < w + k) {

i = nge[i];

}

System.out.println(arr[i]);

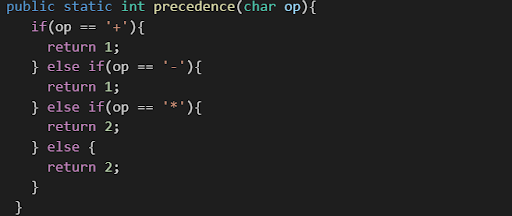
}

}

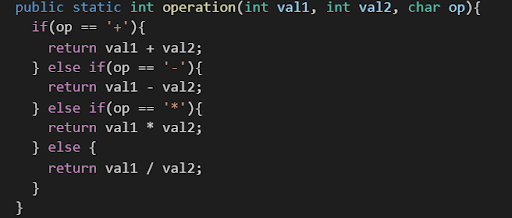
}

**4. Code Discussion :**

We need two functions which we will be using often. So first look at those. First is the precedence function which will return us the precedence of the particular operator. Since the precedence of "+" and "-" operator is the same, we return 1. Also the precedence of "\*" and "/" is the same so we return 2.

****

Second function is the operation function, which takes two values of operand and one operator. It performs the required operation and returns the answer.

****

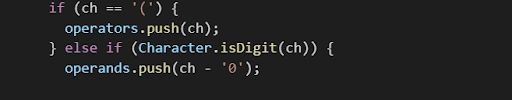
Now we start to write our code in main. Here we first of all define our two stacks, one for operand and another for operators.

****

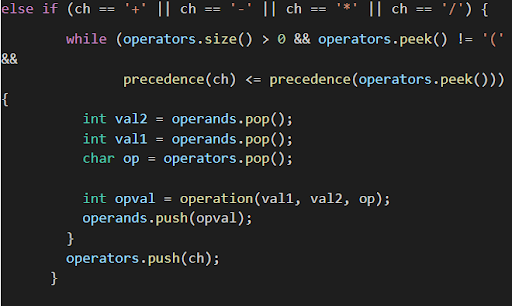
We run a for loop on our expression and process its each character one by one by capturing it in a character variable ch.

****

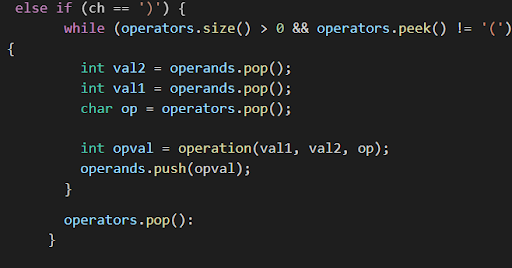
If the character is "(" or any digit (convert this char digit into integer using (ch- "0") than without thinking any further we push these elements into operator and operand stack respectively.

****

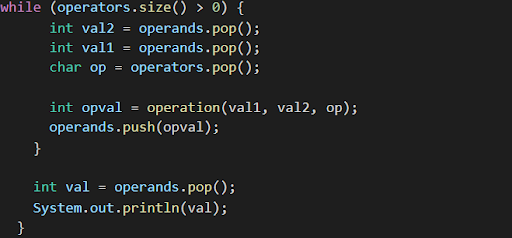
Character ch can also be an operator. In that case we first pop out the operators out of the operator stack with greater precedence, making sure that at any time the stack is not empty and the topmost element is not "(". After popping out the operator we store it in a character op and simultaneously pop out 2 elements of the operand stack and store these in val2 and val1 respectively. Then we call the function operation and pass these 2 stored operands and 1 operator to it and in return it gives us the solved value, which we are supposed to push in the operand stack. And finally we push our current operator ch into the operator stack.

****

Character ch can also be ")". In that case also we pop out the operators out of the operator stack, making sure that at any time the stack is not empty and the topmost element is not "(". After popping out the operator we store it in a character op and simultaneously pop out 2 elements of the operand stack and store these in val2 and val1 respectively. Then we call the function operation and pass these 2 stored operands and 1 operator to it and in return it gives us the solved value, which we are supposed to push in the operand stack. And finally we pop out "(" which will be the peak of the operator stack.

****

When we come out of the for loop, it's important to empty out the operator stack in case it's not. First of all we pop out the operator out of the operator stack, making sure that at any time the stack is not empty. After popping out the operator we store it in a character op and simultaneously pop out 2 elements of the operand stack and store these in val2 and val1 respectively. Then we call the function operation and pass these 2 stored operands and 1 operator to it and in return it gives us the solved value, which we are supposed to push in the operand stack. The loop runs until the operator stack is not empty. At last we print operand.pop which stores our final answer.

****

**5. Analysis :**

Time Complexity : O(n)

The complexity of this is O(n) because only a for loop is running n times, which is the size of the given expression.

Space Complexity : O(n)

2 stacks are used but at any moment the total space used will be less than or equal to O(n), making the space complexity O(n).

End Note :

It would be an extreme pleasure if this article was helpful to you. Our desire to make you learn will remain unsatisfactory if you still have doubts. We strongly recommend you to watch our video lecture on Infix Evaluation for clearing any type of doubts. Suggestions and feedback are always welcomed. You can contact us via our website. Don't give up on any condition. Attempt more and more questions daily. Choose the difficult ones even if you fail to do that, even if you fear that problem, All the best for a bright future!, Happy Coding!