

## CSC310\_HW\_3\_1

```

5
6
7 private LinkedList<Integer> l = new LinkedList<Integer>();
8 private int n = -1;
9
10 //add new element to stack
11 void push(int num){
12     l.add(num);
13     n++;
14 }
15
16 //removes top element of stack
17 void pop(){
18     if(n>=0){
19         l.remove(n);
20         n--;
21     }
22 }
23
24 //returns min in stack
25 int getMin(){
26     int min=l.get(0);
27     for (int i = 0; i < n+1; i++) {
28         if(min > l.get(i)) {
29             min = l.get(i);
30         }
31     }
32     return min;
33 }

```

In this program I made a class called minstack and its meant to manage a int value with the option to find the min value in the stack. I decided to make it an int because it wasn't specific and integers are easy to work with and small data type. I decided to make the min value return a int because that is what it ask for. I also included the basic stack functions push and pop to add and remove from the stack.

```

35 int top(){
36     return l.get(n);
37 }
38
39 //empty stack
40 void emptyStack(){
41     l.clear();
42 }
43 }
44
45 public class csc310_hw_3_1 {
46
47     /**
48      * @param args the command line arguments
49      */
50     public static void main(String[] args) {
51
52         // simple menu for easy use
53         Scanner input = new Scanner(System.in);
54         String choice = "";
55         MinStack minStack = new MinStack();
56
57         do{
58             System.out.println("1. Run Demo");
59             System.out.println("2. Push");
60             System.out.println("3. Pop");
61             System.out.println("4. Top");
62             System.out.println("5. Get Min");
63             System.out.println("6. Empty Stack");
64             System.out.println("e. Exit");
65             System.out.print("Pick an option from the menu above:");
66             choice = input.next();
67
68             switch(choice.charAt(0)){
69                 case '1':
70                     switch(choice.charAt(0)){
71                         case '1':
72                             minStack.emptyStack();
73
74                             minStack.push(-2);
75                             minStack.push(0);
76                             minStack.push(-3);
77
78                             System.out.println(minStack.getMin());
79
80                             minStack.pop();
81
82                             System.out.println(minStack.top());
83                             System.out.println(minStack.getMin());
84
85                             minStack.emptyStack();
86                             break;
87                         case '2':
88                             System.out.print("Enter integer: ");
89                             minStack.push(input.nextInt());
90                             break;
91                         case '3':
92                             try{
93                                 minStack.pop();
94                             }catch(Exception e){
95                                 System.out.println("Stack is empty");
96                             }
97                             break;
98                         case '4':
99

```

I have another basic stack function top and added a emptyStack function to work better with the menu that you can see it's a basic and simple menu. I chose to ad a menu to easily use the function and to quickly to run the Demo with example in it.

This an continuation of the menu with the first 3 cases

```

Output - CSC310_HW_3_1 (run) #2 X
run:
1. Run Demo
2. Push
3. Pop
4. Top
5. Get Min
6. Empty Stack
e. Exit
Pick an option from the menu above:1
-3
0
-2

1. Run Demo
2. Push
3. Pop
4. Top
5. Get Min
6. Empty Stack
e. Exit
Pick an option from the menu above:

```

This is an example of the inputs and out puts for the program and how the menu works.

## CSC310\_HW\_3\_2

```

5 class Stack{
6
7     private LinkedList<String> l;
8     private int n;
9
10    Stack(){
11        l=new LinkedList<>();
12        n=0;
13    }
14
15    //add new element to stack
16    public void push(String c){
17        l.add(c);
18        n=n+1;
19    }
20
21    //returns the size of stack
22    public int size(){
23        return n;
24    }
25
26    //returns and takes out what is on top of stack
27    public String pop(){
28        if(n>=0){
29            String s = top();
30            l.remove(n-1);
31            n--;
32            return s;
33        }else
34            return "Stack is empty";
35    }

```

I made String Stack for easier manipulation of the data and so that I can use it for both the value and operations stack. It has similar functions to the first program with the basics with an additional size function.

```

37
38    //returns what is on top of stack
39    public String top(){
40        return l.get(n-1);
41    }
42
43    //clears stack
44    public void emptyStack(){
45        l.clear();
46    }

```

```

56 public class CSC310_HW_3_2 {
57     static Stack valStk= new Stack();
58     static Stack opStk= new Stack();
59
60     //Makes the expressions with two signs into the one with one
61     static String toOne(String s){
62         switch(s){
63             case "<=": //less then or equal to
64                 return "\u2264";
65             case ">=": //greater then or equal to
66                 return "\u2265";
67             case "!=": //not equal to
68                 return "\u2260";
69             default:
70                 return "ERROR";
71         }
72     }

```

I decided to make a function to handle if the user inputs characters that doesn't work well in the stack such as <= or != and change it to a more stack friendly format.

```

74 static int prec(String s){
75     switch(s){
76         case "+":
77             return 2;
78         case "-":
79             return 2;
80         case "*":
81             return 3;
82         case "/":
83             return 3;
84         case "=":
85             return 1;
86         case ">":
87             return 1;
88         case "<":
89             return 1;
90         case "\u2264": //less then or equal to
91             return 1;
92         case "\u2265": //greater then or equal to
93             return 1;
94         case "\u2260": //not equal to
95             return 1;
96         case "$":
97             return 0;
98     }
99 }
103 static void doOp(){
104     double y = Double.parseDouble(valStk.pop());
105     double x = Double.parseDouble(valStk.pop());
106
107     switch(opStk.pop()){
108         case "+":
109             valStk.push(""+(x+y));
110             break;
111         case "-":
112             valStk.push(""+(x-y));
113             break;
114         case "*":
115             valStk.push(""+(x*y));
116             break;
117         case "/":
118             valStk.push(""+(x/y));
119             break;
120         case ">":
121             valStk.push(""+(x>y));
122             break;
123         case "<":
124             valStk.push(""+(x<y));
125             break;
126         case "=":
127             valStk.push(""+(x==y));
128             break;
129         case "\u2264": //less then to equak to
130             valStk.push(""+(x<=y));
131             break;
132         case "\u2265": //greater then or equal to
133             valStk.push(""+(x>=y));
134             break;
135         case "\u2260": //not equal
136             valStk.push(""+(x!=y));
137             break;
138         default:
139             System.out.println("Not an op");
140     }
141 }

```

I made this function to keep track of the precedence of each operation and to help compare them better in if and loops.

I made this function to do the actual operations associated with the character. I felt like a switch statement was the best and most efficient way of handling this.

```

143 // decides when to use doOp
144 static void repeatOps(String op){
145     if(( opStk.size() > 1 && prec(op) <= prec(opStk.top()))){
146         doOp();
147     }
148     if(op == "+" && valStk.size()==2){
149         doOp();
150     }
151     if(op != "+")
152         opStk.push(op);
153 }
154
155 // checks if number
156 static boolean isNumeric(String c){
157     try{
158         Integer.parseInt(c);
159         return true;
160     }catch(Exception e){
161         return false;
162     }
163 }
164
165 static String EvalExp(String s){
166     while(valStk.size()<=0 || opStk.size()<=0){
167         if(isNumeric(s.substring(0,1))){
168             if(isNumeric(s.substring(1,2))){ //to handle numbers with multiple digits
169                 valStk.push(s.substring(0,2));
170                 s=s.substring(2);
171             }
172             else{
173                 valStk.push(s.substring(0,1));
174                 s=s.substring(1);
175             }
176         }
177         else{
178             if(!isNumeric(s.substring(1,2))){
179                 String s2= toOne(s.substring(0,2));
180                 opStk.push(s2);
181                 s=s.substring(2);
182             }
183             else{
184                 opStk.push(s.substring(0,1));
185                 s=s.substring(1);
186             }
187         }
188     }
189
190     while(s.length()>0){
191         while(s.length()>0){
192             if(isNumeric(s.substring(0,1))){
193                 if(s.length()>1 && isNumeric(s.substring(1,2))){
194                     valStk.push(s.substring(0,2));
195                     s=s.substring(2);
196                 }
197                 else{
198                     valStk.push(s.substring(0,1));
199                     s=s.substring(1);
200                 }
201             }
202             else{
203                 if(s.length()>1 && !isNumeric(s.substring(1,2))){
204                     String s2= (s.substring(0,2));
205                     repeatOps(s2);
206                     s=s.substring(2);
207                 }
208                 else{
209                     repeatOps(s.substring(0,1));
210                     s=s.substring(1);
211                 }
212             }
213         }
214     }
215     while(valStk.size()>1)
216         repeatOps("+");
217     return valStk.top();
218 }
219
220
221 public static void main(String[] args) {
222     Scanner input = new Scanner(System.in);
223
224     System.out.print("Enter Expression: ");
225     System.out.println(EvalExp(input.next()));
226 }
227

```

This function was made to handle when to do the operations

This function I created to check and to if the string is a number or not and returns a Boolean to easily sort between number and operation

This function handles the full expression and calls other functions that does the operations it also I decided to make sure that it can tell when it's a multidigit number or a two part operation such as!=.

The screenshot below is an example input and output of the program.

```

Output - CSC310_HW_3_2 (run) X
run:
Enter Expression: 14<=4-3*2+7
false
BUILD SUCCESSFUL (total time: 33 seconds)

```

```

55 public class CSC310_HW_3_3 {
56     static Stack valStk= new Stack();
57
58     // checks if number
59     static boolean isNumeric(String c){
60         try{
61             Integer.parseInt(c);
62             return true;
63         }catch(Exception e){
64             return false;
65         }
66     }
67
68     static void ops(String s){
69         double y = Double.parseDouble(valStk.pop());
70         double x = Double.parseDouble(valStk.pop());
71
72         switch(s){
73             case "+":
74                 valStk.push(""+(x+y));
75                 break;
76             case "-":
77                 valStk.push(""+(x-y));
78                 break;
79             case "*":
80                 valStk.push(""+(x*y));
81                 break;
82             case "/":
83                 valStk.push(""+(x/y));
84                 break;
85         }
86     }
87
88     static String EvalExp(String s){
89         while(s.length()>0) {
90             if(isNumeric(s.substring(0,1))){
91                 valStk.push(s.substring(0,1));
92             }else
93                 ops(s.substring(0,1));
94             s=s.substring(1);
95         }
96
97         return valStk.top();
98     }
99 }
100

```

For this program I was able to reuse a lot of the code from the last program it work very well such as the string stack class and the is numeric.

I was able to bacially sue all the code from the from the doOp function and renamed ops for operations which this function handles all the operations section

This function serves the same purpose as from the last program but I was able to streamline it with one stack and with a middle function I was able to send it straight to the ops function.

```

102 public static void main(String[] args) {
103     Scanner input = new Scanner(System.in);
104
105     System.out.print("Please enter postfix expression: ");
106     System.out.println(EvalExp(input.next()));
107 }
108
109 }

```

This is the cod that prompts the user for input.

```

Output - CSC310_HW_3_3 (run) X
run:
Please enter postfix expression: 52+83-*4/
8.75
BUILD SUCCESSFUL (total time: 21 seconds)

```

This is example of the postfix expression input and the out from it.