Evaluating postfix expressions using stacks

The Postfix notation is used to represent algebraic expressions. The expressions written in postfix form are evaluated faster compared to infix notation as parenthesis are not required in postfix. We have already discussed the conversion of infix to postfix expressions. In this post, the next step after that, that is evaluating a postfix expression is discussed.

Following is the algorithm for evaluation of postfix expressions:

1. Create a stack to store operands (or values).
2. Scan the given expression and do following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
3. When the expression is ended, the number in the stack is the final answer.

Example: Let the given expression be "***2 3 1 * + 9 -***". We will first scan all elements one by one.

1. Scan '2', it's a number, so push it to stack. Stack contains '2'
2. Scan '3', again a number, push it to stack, stack now contains '2 3' (from bottom to top)
3. Scan '1', again a number, push it to stack, stack now contains '2 3 1'
4. Scan '*', it's an operator, pop two operands from the stack, apply the * operator on operands, we get 3*1 which results in 3. We push the result '3' to stack. Stack now becomes '2 3'.
5. Scan '+', it's an operator, pop two operands from the stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result '5' to stack. Stack now becomes '5'.
6. Scan '9', it's a number, we push it to the stack. Stack now becomes '5 9'.
7. Scan '-', it's an operator, pop two operands from the stack, apply the - operator on operands, we get 5 - 9 which results in -4. We push the result '-4' to stack. Stack now becomes '-4'.
8. There are no more elements to scan, we return the top element from the stack (which is the only element left in the stack).

Below is the implementation of the above algorithm:

C++ Java

```cpp
// C++ program to evaluate value of a postfix expression
#include <iostream>
#include <string.h>
#include<stack>
using namespace std;
// Function that returns the value of a given postfix expression
int evaluatePostfix(char* exp)
{
    // Create a stack
    stack<int> st;
    int i;     // Scan all characters one by one
    for (i = 0; exp[i]; ++i)
    {
        // If the scanned character is an operand (number here),push it to the stack.
        if (isdigit(exp[i]))
            st.push(exp[i] - '0');
        // If the scanned character is an operator, pop two
        // elements from stack apply the operator
        else
        {
            int val1 = st.top();
            st.pop();
            int val2 = st.top();
            st.pop();
            switch (exp[i])
            {
```

```
                case '+': st.push(val2 + val1); break;
                case '-': st.push(val2 - val1); break;
                case '*': st.push(val2 * val1); break;
                case '/': st.push(val2/val1); break;
            }
        }
    }
    return st.top();
}

// Driver Code
int main()
{
    char exp[] = "231*+9-";
    cout<<"postfix evaluation: "<< evaluatePostfix(exp);
    return 0;
}

```

Output:

```
postfix evaluation: -4
```

*The **time complexity** of evaluation algorithm is O(n) where n is number of characters in input expression.*

There are following limitations of the above implementation:

1. It supports only 4 binary operators '+', '*', '-' and '/'. It can be extended for more operators by adding more switch cases.
2. The allowed operands are only single-digit operands. The program can be extended for multiple digits by adding a separator like space between all elements (operators and operands) of the given expression.

Below given is the extended program which allows operands having multiple digits.

C++ Java

```
// CPP program to evaluate value of a postfix expression having multiple digit operands
#include <bits/stdc++.h>
using namespace std;
// Function that returns value of a given postfix expression
int evaluatePostfix(char* exp)
{
    // Create a stack
    stack<int> st;
    int i;
    // Scan all characters one by one
    for (i = 0; exp[i]; ++i)
    {
        // if the character is blank space then continue
        if(exp[i] == ' ')continue;
        // If the scanned character is an operand (number here), extract the full number
        // Push it to the stack.
        else if (isdigit(exp[i]))
        {
            int num=0; // extract full number
            while(isdigit(exp[i]))
            {
                num = num * 10 + (int)(exp[i] - '0');
                    i++;
            }
            i--;
            // push the element in the stack
            st.push(num);
        }

```

```
        // If the scanned character is an operator, pop two
        // elements from stack apply the operator
        else
        {
            int val1 = st.top();
            st.pop();
            int val2 = st.top();
            st.pop();
            switch (exp[i])
            {
                case '+': st.push(val2 + val1); break;
                case '-': st.push(val2 - val1); break;
                case '*': st.push(val2 * val1); break;
                case '/': st.push(val2/val1); break;
            }
        }
    }
    return st.top();
}
// Driver code
int main()
{
    char exp[] = "100 200 + 2 / 5 * 7 +";
    cout << evaluatePostfix(exp);
    return 0;
}
```

Output :

```
757
```