

```

1  #include <iostream>
2  #include <string>
3  #include <cstring>
4
5  using namespace std;
6
7
8  class Node{
9
10     private:
11         int data;
12         Node* nextNodePtr;
13         Node* prevNodePtr;
14
15     public:
16         Node() {}
17
18         void setData(int d){
19             data = d;
20         }
21
22         int getData(){
23             return data;
24         }
25
26         void setNextNodePtr(Node* nodePtr){
27             nextNodePtr = nodePtr;
28         }
29
30         Node* getNextNodePtr(){
31             return nextNodePtr;
32         }
33
34         void setPrevNodePtr(Node* nodePtr){
35             prevNodePtr = nodePtr;
36         }
37
38         Node* getPrevNodePtr(){
39             return prevNodePtr;
40         }
41
42 };
43
44 class Stack{
45
46     private:
47         Node* headPtr;
48         Node* tailPtr;
49
50
51     public:
52         Stack(){
53             headPtr = new Node();
54             tailPtr = new Node();
55             headPtr->setNextNodePtr(0);
56             tailPtr->setPrevNodePtr(0);
57         }
58
59         Node* getHeadPtr(){
60             return headPtr;
61         }
62
63         Node* getTailPtr(){
64             return tailPtr;

```

```

65     }
66
67     bool isEmpty() {
68
69         if (headPtr->getNextNodePtr() == 0)
70             return true;
71
72         return false;
73     }
74
75
76     void push(int data) {
77
78         Node* newNodePtr = new Node();
79         newNodePtr->setData(data);
80         newNodePtr->setNextNodePtr(0);
81
82         Node* lastNodePtr = tailPtr->getPrevNodePtr();
83
84         if (lastNodePtr == 0) {
85
86             headPtr->setNextNodePtr(newNodePtr);
87             newNodePtr->setPrevNodePtr(0);
88
89         }
90         else {
91
92             lastNodePtr->setNextNodePtr(newNodePtr);
93             newNodePtr->setPrevNodePtr(lastNodePtr);
94
95         }
96
97         tailPtr->setPrevNodePtr(newNodePtr);
98
99     }
100
101
102     int pop() {
103
104         Node* lastNodePtr = tailPtr->getPrevNodePtr();
105         Node* prevNodePtr = 0;
106
107         int poppedData = -100000; //empty stack
108
109         if (lastNodePtr != 0) {
110             prevNodePtr = lastNodePtr->getPrevNodePtr();
111             poppedData = lastNodePtr->getData();
112         }
113         else
114             return poppedData;
115
116         if (prevNodePtr != 0) {
117             prevNodePtr->setNextNodePtr(0);
118             tailPtr->setPrevNodePtr(prevNodePtr);
119         }
120         else {
121             headPtr->setNextNodePtr(0);
122             tailPtr->setPrevNodePtr(0);
123         }
124
125         return poppedData;
126     }
127
128

```

```

129     int peek(){
130
131         Node* lastNodePtr = tailPtr->getPrevNodePtr();
132
133         if (lastNodePtr != 0)
134             return lastNodePtr->getData();
135         else
136             return -100000; // empty stack
137
138     }
139
140
141
142
143 };
144
145 int main(){
146
147     Stack stack;
148
149     string expression;
150
151     cout << "Enter the expression to evaluate: ";
152     getline(cin, expression);
153     char* expressionArray = new char[expression.length()+1];
154     strcpy(expressionArray, expression.c_str());
155
156     char* cptr = strtok(expressionArray, ", ");
157
158     while (cptr != 0){
159
160         string token(cptr);
161
162         bool isOperator = false;
163
164         if ( (token.compare("*") == 0) || (token.compare("/") == 0) || (token.compare(
165             "+") == 0) || (token.compare("-") == 0) )
166             isOperator = true;
167
168         if (!isOperator){
169             int val = stoi(token);
170             stack.push(val);
171         }
172
173         if (isOperator){
174
175             int rightOperand = stack.pop();
176             int leftOperand = stack.pop();
177
178             if (token.compare("*") == 0){
179                 int result = leftOperand * rightOperand;
180                 cout << "intermediate result: " << result << endl;
181                 stack.push(result);
182             }
183             else if (token.compare("/") == 0){
184                 int result = leftOperand / rightOperand;
185                 cout << "intermediate result: " << result << endl;
186                 stack.push(result);
187             }
188             else if (token.compare("+") == 0){
189                 int result = leftOperand + rightOperand;
190                 cout << "intermediate result: " << result << endl;
191                 stack.push(result);

```

```
192     }
193     else if (token.compare("-") == 0){
194         int result = leftOperand - rightOperand;
195         cout << "intermediate result: " << result << endl;
196         stack.push(result);
197     }
198 }
199 }
200
201     cptr = strtok(NULL, ", ");
202 }
203
204     cout << "final result: " << stack.pop() << endl;
205
206
207 return 0;
208 }
```

```
Enter the expression to evaluate: 2, 3, *, 1, 5, *, +, 4, -
intermediate result: 6
intermediate result: 5
intermediate result: 11
intermediate result: 7
final result: 7
```
