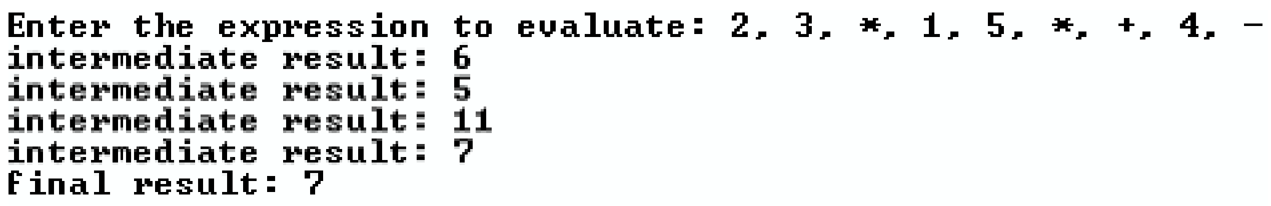
```cpp
#include <iostream>
#include <string>
#include <cstring>

using namespace std;


class Node{

    private:
        int data;
        Node* nextNodePtr;
        Node* prevNodePtr;

    public:
        Node(){}

        void setData(int d){
            data = d;
        }

        int getData(){
            return data;
        }

        void setNextNodePtr(Node* nodePtr){
                nextNodePtr = nodePtr;
        }

        Node* getNextNodePtr(){
            return nextNodePtr;
        }

        void setPrevNodePtr(Node* nodePtr){
                prevNodePtr = nodePtr;
        }

        Node* getPrevNodePtr(){
            return prevNodePtr;
        }

};

class Stack{

    private:
        Node* headPtr;
        Node* tailPtr;


    public:
        Stack(){
            headPtr = new Node();
            tailPtr = new Node();
            headPtr->setNextNodePtr(0);
            tailPtr->setPrevNodePtr(0);
        }

        Node* getHeadPtr(){
            return headPtr;
        }

        Node* getTailPtr(){
            return tailPtr;
        }

        bool isEmpty(){

            if (headPtr->getNextNodePtr() == 0)
                return true;

            return false;
        }


        void push(int data){

            Node* newNodePtr = new Node();
            newNodePtr->setData(data);
            newNodePtr->setNextNodePtr(0);

            Node* lastNodePtr = tailPtr->getPrevNodePtr();

            if (lastNodePtr == 0){

                headPtr->setNextNodePtr(newNodePtr);
                newNodePtr->setPrevNodePtr(0);

            }
            else{

                lastNodePtr->setNextNodePtr(newNodePtr);
                newNodePtr->setPrevNodePtr(lastNodePtr);

            }

            tailPtr->setPrevNodePtr(newNodePtr);

        }


        int pop(){

            Node* lastNodePtr = tailPtr->getPrevNodePtr();
            Node* prevNodePtr = 0;

            int poppedData = -100000; //empty stack

            if (lastNodePtr != 0){
                prevNodePtr = lastNodePtr->getPrevNodePtr();
                poppedData = lastNodePtr->getData();
            }
            else
                return poppedData;

            if (prevNodePtr != 0){
                prevNodePtr->setNextNodePtr(0);
                tailPtr->setPrevNodePtr(prevNodePtr);
            }
            else{
                headPtr->setNextNodePtr(0);
                tailPtr->setPrevNodePtr(0);
            }

            return poppedData;
        }


        int peek(){

            Node* lastNodePtr = tailPtr->getPrevNodePtr();

            if (lastNodePtr != 0)
                return lastNodePtr->getData();
            else
                return -100000; //  empty stack


        }



};

int main(){

    Stack stack;

    string expression;

    cout << "Enter the expression to evaluate: ";
    getline(cin, expression);
    char* expressionArray = new char[expression.length()+1];
    strcpy(expressionArray, expression.c_str());

    char* cptr = strtok(expressionArray, ", ");

    while (cptr != 0){

        string token(cptr);

        bool isOperator = false;

        if ( (token.compare("*") == 0) || (token.compare("/") == 0) || (token.compare(
        "+") == 0) || (token.compare("-") == 0) )
            isOperator = true;

        if (!isOperator){
            int val = stoi(token);
            stack.push(val);
        }


        if (isOperator){

            int rightOperand = stack.pop();
            int leftOperand = stack.pop();

            if (token.compare("*") == 0){
                int result = leftOperand * rightOperand;
                cout << "intermediate result: " << result << endl;
                stack.push(result);
            }
            else if (token.compare("/") == 0){
                int result = leftOperand / rightOperand;
                cout << "intermediate result: " << result << endl;
                stack.push(result);
            }
            else if (token.compare("+") == 0){
                int result = leftOperand + rightOperand;
                cout << "intermediate result: " << result << endl;
                stack.push(result);
            }
            else if (token.compare("-") == 0){
                int result = leftOperand - rightOperand;
                cout << "intermediate result: " << result << endl;
                stack.push(result);
            }

        }


        cptr = strtok(NULL, ", ");
    }

    cout << "final result: " << stack.pop() << endl;

return 0;
}
```



```
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
```