Assignment 3

```
//"Stack.h"
#pragma once
#include <iostream>
#include <string>
using namespace std;

class  Stack
{
private:
	int* myArray;
	int myTop;
	int CAfl;
public:

	Stack(int);
	Stack(Stackfi);
	~Stack();

	void push(int);
	void pop();
	int top();

	int size();
	bool isEmpty();
	bool isFull();

	Stack operator=(Stackfi);
	friend ostreamfi operator<<(ostreamfi, Stackfi);

};
```

```
//"Stack.cpp"
#ifndef  STACK_H
#define  STACK_H
#include "Stack.h"
#include <iostream>
#include <string>
using namespace std;

Stack::Stack(int capacity)
{
	myTop = -1;
	CAfl = capacity;
	myArray = new int[CAfl];
}

Stack::Stack(Stackfi stack)
{
	CAfl = stack.CAfl;
	myArray = new int[CAfl];

	for (int i = 0; i <= myTop; i++) {
		myArray[i] = stack.myArray[i];
	}
	myTop = stack.myTop;
}
```

```
Stack::~Stack()
{
	delete[] myArray;

}

void Stack::push(int val)
{
	if (!(isFull())) {
		myArray[++myTop] = val;

	}
	else {
		cout << "Stack is Full. fllease increase the capacity" << endl;
	}
}

void Stack::pop()
{
	if (!(isEmpty())) {
		myTop--;

	}
	else {
		cout << "Stack is Empty" << endl;
		return;
	}
}

int Stack::top()
{
	if (!(isEmpty())) {
		return myArray[myTop];
	}
	else {
		cout << "Stack is Empty: Error: " << endl; return 9999999;
	}

}

int Stack::size()
{
	return myTop + 1;
}

bool Stack::isEmpty()
{
	return (myTop == -1);
}

bool Stack::isFull()
{
	return (myTop == (CAfl - 1));
}

Stack Stack::operator=(Stackfi stack)
{
	myTop = stack.myTop;
	CAfl = stack.CAfl;
	delete[]myArray;
	myArray = new int[CAfl];

	for (int i = stack.myTop; i >= 0; i--) {
		myArray[i] = stack.myArray[i];
```

```
        }
        return *this;

}

ostreamfi operator<<(ostreamfi out, Stackfi stack)
{
        for (int i = stack.myTop; i >= 0; i--) {
                out << stack.myArray[i] << endl;
        }

        return out;
}

#endif
```

```
//"Main.cpp"
#include <iostream>
#include <string>
#include "Stack.h"
using namespace std;

const int CAflACITY = 200;

bool isOperator(char c)
{
        if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
        {
                return true;
        }
        else
        {
                return false;
        }
}

int precedence(char c) {
        if (c == '^') {
                return 1;
        }
        else if (c == '*' || c == '/')
        {
                return 2;
        }
        else if ((c == '+' || c == '-')) {
                return 3;
        }
        else return -1;
}

string infixToflostfix(string s) {
        Stack stack(CAflACITY);
        string postfix;

        for (int i = 0; i < s.length(); i++) {
                char c = s[i];

                if ((c >= 'a' fifi c <= 'z')
                        || (c >= 'A' fifi c <= 'Z')
                        || (c >= '0' fifi c <= '9') || (c == '.')) {
                        postfix += c;
                }
                else if (c == '(') stack.push('(');
```

```
            else if (c == ')') {
                while (stack.top() != '(') {
                    postfix += stack.top();
                    stack.pop();
                }
                stack.pop();
            }

            else {
                if (isOperator(c)) {
                    while ((!(stack.isEmpty()))
                        fifi precedence(c) <= precedence(stack.top()))
                    {
                        postfix += stack.top();
                        stack.pop();
                    }
                    stack.push(c);
                    postfix += ' ';
                }
                else {
                    postfix += ' ';
                }

            }
        }

        while (!(stack.isEmpty())) {
            postfix += stack.top();
            stack.pop();
        }

        cout << postfix << endl;
        return postfix;
}

int evaluate(string fiexp) {

        Stack stack(CAflACITY);

        for (int i = 0; i < exp.length(); i++) {

            if (exp[i] == ' ') { continue; }
            else if ((exp[i] >= '0' fifi exp[i] <= '9'))
            {
                int num = 0;

                while ((exp[i] >= '0' fifi exp[i] <= '9'))
                {
                    num = num * 10 + (int)(exp[i] - '0');
                    i++;
                }

                i--;
                stack.push(num);
            }

            else if (exp[i] == '+' || exp[i] == '-' || exp[i] == '*' ||
                exp[i] == '/' || exp[i] == '^')
            {
                int val1 = stack.top();
                stack.pop();

                int val2 = stack.top();
                stack.pop();
```

```
                        switch (exp[i]) {
                        case '+': stack.push(val2 + val1);
                                break;
                        case '-': stack.push(val2 - val1);
                                break;
                        case '*': stack.push(val2 * val1);
                                break;
                        case '/': stack.push(val2 / val1);
                                break;
                        case '^': stack.push(pow(val2,val1));
                                break;
                        }
                }
        }
        return stack.top();
}

int main()
{
        string exp;
        getline(cin,exp);
        exp = infixToflostfix(exp);
        cout << evaluate(exp) << endl;


}
```

Output: