ATMA RAM SANATAN DHARMA

DSA ASSIGNMENT

-Shishirant Singh

-22/28081

ASSIGNMENT

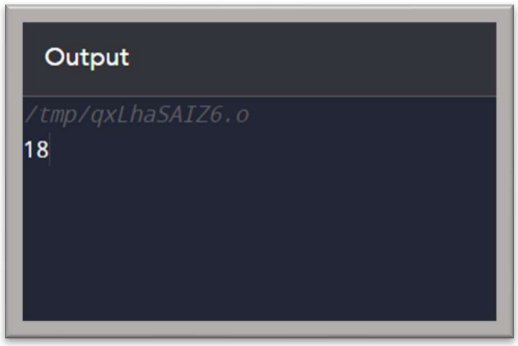
Q1.) Evaluation of postfix expression:
ANSWER-

CODE:

```
#include <iostream>
#include <string>
#include <stack>
using namespace std;

// Function to evaluate a given postfix expression
int evalPostfix(string exp)
{
    // create an empty stack
    stack<int> stack;

    // traverse the given expression
    for (char c: exp)
    {
        // if the current character is an operand, push it into the stack
        if (c >= '0' && c <= '9') {
            stack.push(c - '0');
        }
        // if the current character is an operator
        else {
            // remove the top two elements from the stack
            int x = stack.top();
            stack.pop();

            int y = stack.top();
            stack.pop();

            // evaluate the expression 'x op y', and push the
            // result back to the stack
            if (c == '+') {
                stack.push(y + x);
            }
            else if (c == '-') {
                stack.push(y - x);
            }
            else if (c == '*') {
                stack.push(y * x);
            }
            else if (c == '/') {
```

```
                stack.push(y / x);
            }
        }
    }

    // At this point, the stack is left with only one element, i.e.,
    // expression result
    return stack.top();
}

int main()
{
    string exp = "138*+825*+";

    cout << evalPostfix(exp);

    return 0;
}
```

OUTPUT:



Q2.) INFIX TO PREFIX:

ANSWER-

CODE:

```
// C++ program to convert infix to prefix
#include <bits/stdc++.h>
using namespace std;

// Function to check if the character is an operator
bool isOperator(char c)
{
	return (!isalpha(c) && !isdigit(c));
}

// Function to get the priority of operators
int getPriority(char C)
{
	if (C == '-' || C == '+')
		return 1;
	else if (C == '*' || C == '/')
		return 2;
	else if (C == '^')
		return 3;
	return 0;
}

// Function to convert the infix expression to postfix
string infixToPostfix(string infix)
{
	infix = '(' + infix + ')';
	int l = infix.size();
	stack<char> char_stack;
	string output;

	for (int i = 0; i < l; i++) {

		// If the scanned character is an
		// operand, add it to output.
		if (isalpha(infix[i]) || isdigit(infix[i]))
			output += infix[i];

		// If the scanned character is an
		// '(', push it to the stack.
```

```
        else if (infix[i] == '(')
                char_stack.push('(');

        // If the scanned character is an
        // ‘)’, pop and output from the stack
        // until an ‘(‘ is encountered.
        else if (infix[i] == ')') {
                while (char_stack.top() != '(') {
                        output += char_stack.top();
                        char_stack.pop();
                }

                // Remove '(' from the stack
                char_stack.pop();
        }

        // Operator found
        else {
                if (isOperator(char_stack.top())) {
                        if (infix[i] == '^') {
                                while (
                                        getPriority(infix[i])
                                        <= getPriority(char_stack.top())) {
                                        output += char_stack.top();
                                        char_stack.pop();
                                }
                        }
                        else {
                                while (
                                        getPriority(infix[i])
                                        < getPriority(char_stack.top())) {
                                        output += char_stack.top();
                                        char_stack.pop();
                                }
                        }

                        // Push current Operator on stack
                        char_stack.push(infix[i]);
                }
        }
}
while (!char_stack.empty()) {
        output += char_stack.top();
        char_stack.pop();
}
return output;
```

```
}

// Function to convert infix to prefix notation
string infixToPrefix(string infix)
{
	// Reverse String and replace ( with ) and vice versa
	// Get Postfix
	// Reverse Postfix
	int l = infix.size();

	// Reverse infix
	reverse(infix.begin(), infix.end());

	// Replace ( with ) and vice versa
	for (int i = 0; i < l; i++) {

		if (infix[i] == '(') {
			infix[i] = ')';
		}
		else if (infix[i] == ')') {
			infix[i] = '(';
		}
	}

	string prefix = infixToPostfix(infix);

	// Reverse postfix
	reverse(prefix.begin(), prefix.end());

	return prefix;
}

// Driver code
int main()
{
	string s = ("x+y*z/w+u");

	// Function call
	cout << infixToPrefix(s) << std::endl;
	return 0;
}
```

OUTPUT:

Output

```
/tmp/qxLhaSAIZ6.o
++X/*YZWU
```