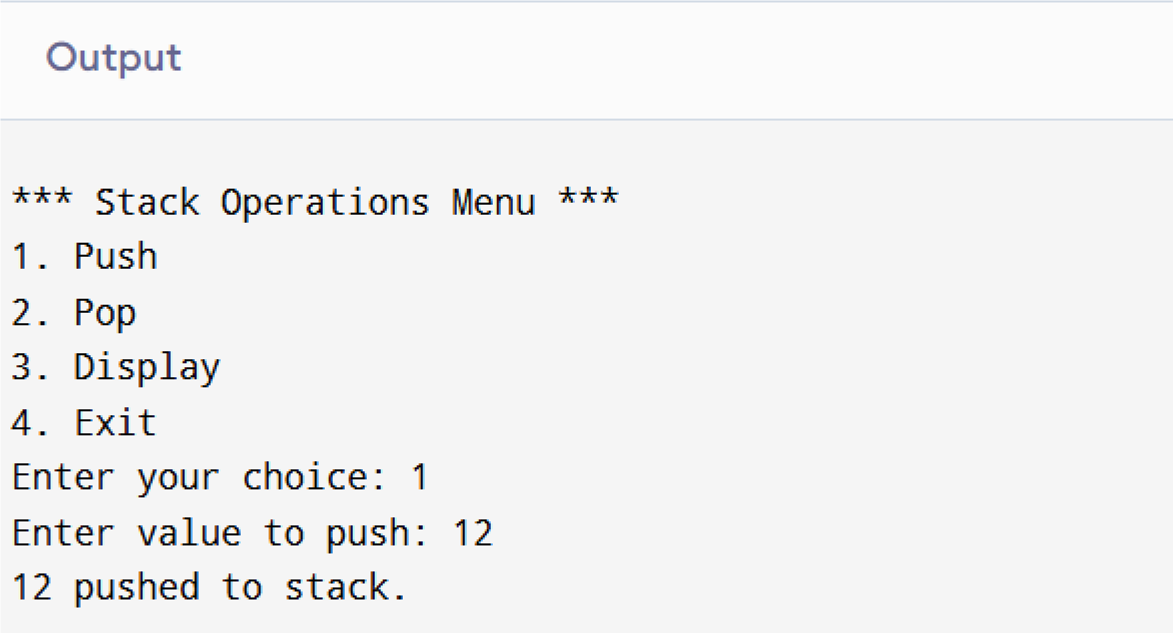
Assignment-4

Q1) Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using arrays.

```
#include <iostream>

const int MAX_SIZE = 5; // Define the maximum size of the stack

class Stack {
private:
  int arr[MAX_SIZE];
  int top;

public:
    Stack() {
    top = -1; // -1 indicates the stack is empty
  }

    void push(int value) {
    if (top >= MAX_SIZE - 1) {
      std::cout << "Stack Overflow! Cannot push " << value << ". The stack is full." <<
std::endl;
    } else {
      top++;
      arr[top] = value;
      std::cout << value << " pushed to stack." << std::endl;
    }
  }

   void pop() {
    if (top < 0) {
      std::cout << "Stack Underflow! Cannot pop. The stack is empty." << std::endl;
    } else {
      std::cout << arr[top] << " popped from stack." << std::endl;
      top--;
    }
  }
```

```
    void display() {
        if (top < 0) {
            std::cout << "Stack is empty." << std::endl;
        } else {
            std::cout << "Stack elements: ";
            for (int i = top; i >= 0; i--) {
                std::cout << arr[i] << " ";
            }
            std::cout << std::endl;
        }
    }
};

int main() {
    Stack s;
    int choice, value;

    do {
        std::cout << "\n*** Stack Operations Menu ***" << std::endl;
        std::cout << "1. Push" << std::endl;
        std::cout << "2. Pop" << std::endl;
        std::cout << "3. Display" << std::endl;
        std::cout << "4. Exit" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter value to push: ";
                std::cin >> value;
                s.push(value);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                s.display();
                break;
            case 4:
                std::cout << "Exiting program. Goodbye!" << std::endl;
                break;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
        }
    } while (choice != 4);

    return 0;
}
```



Output

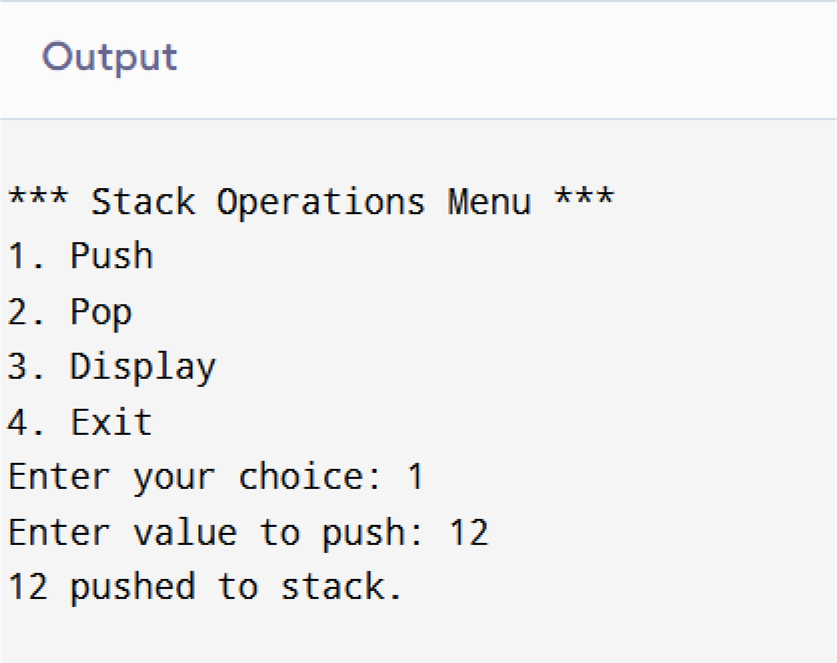
```
*** Stack Operations Menu ***
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 12
12 pushed to stack.
```

Q2) Write a menu driven program with 4 options (Push, Pop, Display, and Exit) to demonstrate the working of stacks using linked-list.

```
#include <iostream>

struct Node {
    int data;
    Node* next;
};

class Stack {
private:
    Node* top;

public:
       Stack() {
        top = nullptr;
    }

     void push(int value) {
        Node* newNode = new Node();
        if (!newNode) {
            std::cout << "Stack Overflow! Memory allocation failed." << std::endl;
            return;
        }
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        std::cout << value << " pushed to stack." << std::endl;
    }

       void pop() {
        if (isEmpty()) {
            std::cout << "Stack Underflow! The stack is empty." << std::endl;
            return;
        }
        Node* temp = top;
        top = top->next;
        std::cout << temp->data << " popped from stack." << std::endl;
        delete temp;
    }

    void display() {
        if (isEmpty()) {
            std::cout << "Stack is empty." << std::endl;
            return;
        }
        std::cout << "Stack elements: ";
        Node* current = top;
        while (current != nullptr) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
```

```
     bool isEmpty() {
        return top == nullptr;
    }
};

int main() {
    Stack s;
    int choice, value;

    do {
        std::cout << "\n*** Stack Operations Menu ***" << std::endl;
        std::cout << "1. Push" << std::endl;
        std::cout << "2. Pop" << std::endl;
        std::cout << "3. Display" << std::endl;
        std::cout << "4. Exit" << std::endl;
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter value to push: ";
                std::cin >> value;
                s.push(value);
                break;
            case 2:
                s.pop();
                break;
            case 3:
                s.display();
                break;
            case 4:
                std::cout << "Exiting program. Goodbye!" << std::endl;
                break;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
        }
    } while (choice != 4);

    return 0;
```

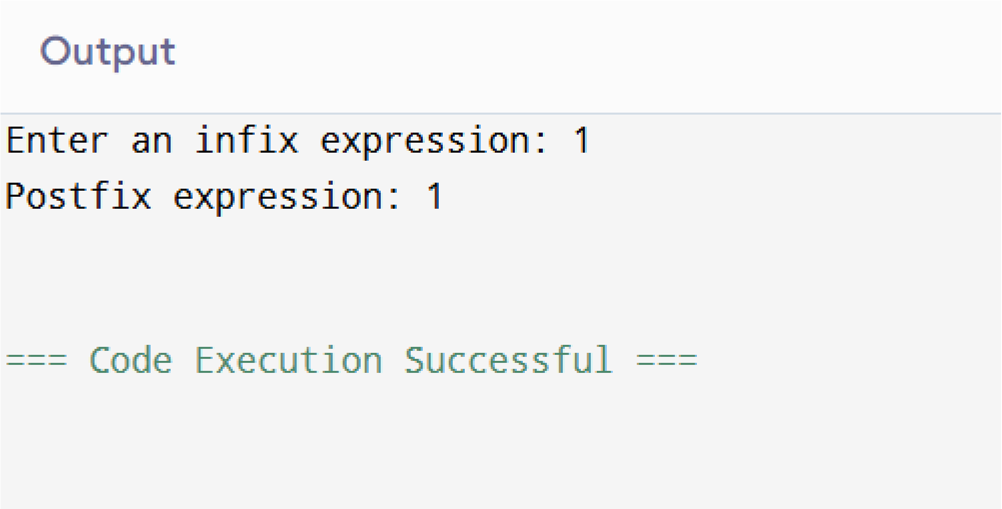
```
}
```



```
*** Stack Operations Menu ***
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 12
12 pushed to stack.
```

Q3) Write a program to convert infix expression into postfix expression using stack.

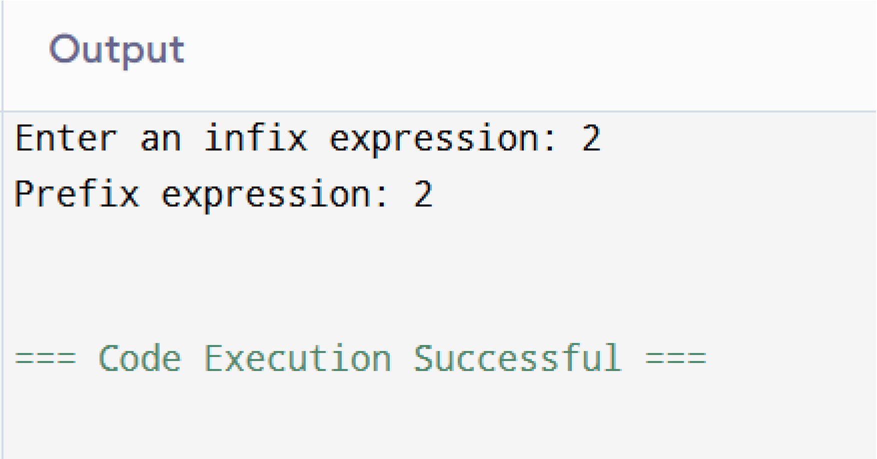
```
#include <iostream>
#include <stack>
#include <string>
bool isOperator(char c) {
   return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}
int precedence(char op) {
   if (op == '^') return 3;
   if (op == '*' || op == '/') return 2;
   if (op == '+' || op == '-') return 1;
   return 0; // For other characters like parentheses
}

std::string infixToPostfix(const std::string& infix) {
   std::string postfix = "";
   std::stack<char> s;

   for (char c : infix) {
      if (isalnum(c)) {
            postfix += c;
        } else if (c == '(') {
            s.push(c);
        } else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            if (!s.empty()) {
                s.pop();
            }
        } else if (isOperator(c)) {
            while (!s.empty() && s.top() != '(' && precedence(s.top()) >= precedence(c)) {
                postfix += s.top();
                s.pop();
            }
            s.push(c);
        }
    }
    while (!s.empty()) {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}

int main() {
    std::string infix_expression;
    std::cout << "Enter an infix expression: ";
    std::getline(std::cin, infix_expression);

    std::string postfix_expression = infixToPostfix(infix_expression);
    std::cout << "Postfix expression: " << postfix_expression << std::endl;

    return 0;
}
```



```
Enter an infix expression: 1
Postfix expression: 1


=== Code Execution Successful ===
```

Q4) Write a program to convert infix expression into prefix expression using stack.

```
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0; // For parentheses
}
std::string infixToPostfix(const std::string& infix) {
    std::string postfix = "";
    std::stack<char> s;

    for (char c : infix) {
        if (isalnum(c)) {
            postfix += c;
        } else if (c == '(') {
            s.push(c);
        } else if (c == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            if (!s.empty()) {
                s.pop(); // Pop the opening parenthesis
            }
        } else if (isOperator(c)) {
            while (!s.empty() && s.top() != '(' && precedence(s.top()) >= precedence(c)) {
                postfix += s.top();
                s.pop();
            }
            s.push(c);
        }
    }

    while (!s.empty()) {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}
std::string infixToPrefix(std::string infix) {
       std::reverse(infix.begin(), infix.end());
    for (char& c : infix) {
        if (c == '(') {
            c = ')';
        } else if (c == ')') {
            c = '(';
        }
    }

        std::string postfix = infixToPostfix(infix);

       std::reverse(postfix.begin(), postfix.end());

    return postfix;
}

int main() {
    std::string infix_expression;
    std::cout << "Enter an infix expression: ";
    std::getline(std::cin, infix_expression);

    std::string prefix_expression = infixToPrefix(infix_expression);
    std::cout << "Prefix expression: " << prefix_expression << std::endl;

    return 0;
}
```

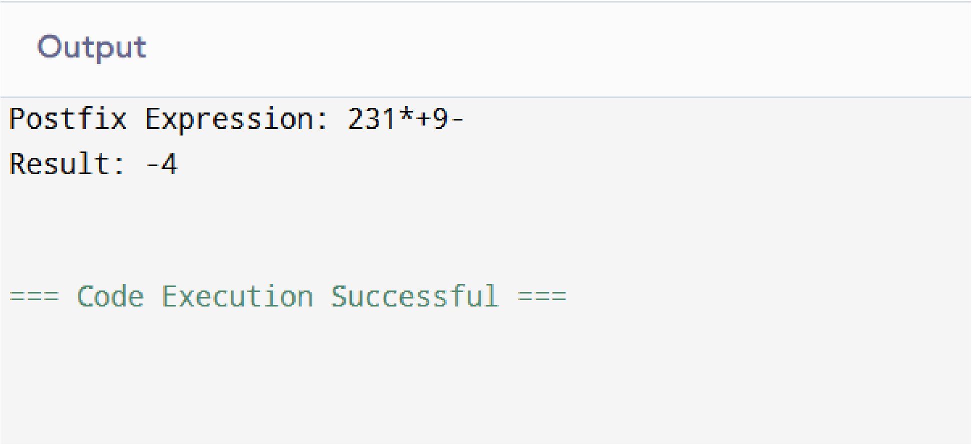


```
Output

Enter an infix expression: 2
Prefix expression: 2


=== Code Execution Successful ===
```

Q5) Write a program to evaluate a postfix expression using stack

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype> // for isdigit()

int evaluatePostfix(const std::string& expression) {
    std::stack<int> s;

    for (char c : expression) {
        if (isdigit(c)) {
                       s.push(c - '0');
        } else if (c == '+' || c == '-' || c == '*' || c == '/') {
                       int operand2 = s.top();
            s.pop();
            int operand1 = s.top();
            s.pop();

                      switch (c) {
```

```
                case '+':
                    s.push(operand1 + operand2);
                    break;
                case '-':
                    s.push(operand1 - operand2);
                    break;
                case '*':
                    s.push(operand1 * operand2);
                    break;
                case '/':
                    s.push(operand1 / operand2);
                    break;
            }
        }
    }

    return s.top();
}

int main() {
    std::string postfix_expression = "231*+9-"; // Example: (2 + 3 * 1) - 9

    std::cout << "Postfix Expression: " << postfix_expression << std::endl;
    int result = evaluatePostfix(postfix_expression);
    std::cout << "Result: " << result << std::endl;

    return 0;
}
```



```
Postfix Expression: 231*+9-
Result: -4


=== Code Execution Successful ===
```