Name: M Fahad

Sap ID: 37125

Section: BSCS-3B

Course: DSA

Assignment#3

Instructor: Noor Ullah Khan

Question 1:

Write a C++ program to implement a stack using an array. The program should support the following operations:

• push: add an element to the top of the stack

• pop: remove the element at the top of the stack

• is_empty: check if the stack is empty

```cpp
#include <iostream>
using namespace std;
const int stackSize = 5;

class Stack {
private:
	int stack[];
	int top;

public:
	Stack() {
		top = -1;
	}

	void Push() {
		int value;
		cout << "Enter Push Value: " << endl;
		cin >> value;
		if (top >= stackSize - 1) {
			cout << "Stack is Full" << endl;
		}
		else {

			top++;
			stack[top] = value;
		}
	}

	void Pop() {
		if (top < 0) {
			cout << "Error: stack is empty" << endl;
		}
		else {
			top--;
		}
	}

	bool is_empty()
	{
		return top < 0;
	}
};

int main() {
        Stack s;
        int c;
        while (true)
        {
                cout << "1. Push" << endl;
                cout << "2. Pop" << endl;
                cout << "3. IsEmptry" << endl;
                cin >> c;
                switch (c)
                {
                case 1:
                        s.Push();
                        break;
                case 2:
                        s.Pop();
                        break;
                case 3:
                        s.is_empty();
                        break;

                }

        }
}
```

Question 2:

Write a C++ program to implement a queue using an array. The program should support the following operations:

• enqueue: add an element to the end of the queue

• dequeue: remove the element at the front of the queue

• is_empty: check if the queue is empty

```
#include <iostream>
using namespace std;
const int size = 5;

class Queue {
private:
    int arr[size];
    int front;
    int rear;

public:
   Queue()
   {
        front = 0;
        rear = -1;
    }

    void enqueue(int value) {
        if (rear == size - 1) {
            cout << "Queue is full" << endl;
            return;
        }
        else
        {
            arr[++rear] = value;
        }

    }

    void dequeue() {
        if (front > rear)
        {
            cout << "Queue is empty" << endl;
            return;
        }
        else
        {
            ++front;
        }

    }

    int get_front()
    {
        if (front > rear)
        {
            cout << "Queue is empty" << endl;
            return -1;
        }
        return arr[front];
    }

    bool is_empty()
    {
        return front > rear;
    }
};

int main() {
    Queue q;
    q.enqueue(1);
    q.enqueue(2);
    q.enqueue(3);
    q.enqueue(4);
    q.enqueue(5);
    while (!q.is_empty())
    {
        cout << q.get_front() << endl;
        q.dequeue();

    }

    return 0;
}
```

Question 3:

Write a C++ program to convert an infix expression to a postfix expression using a stack. The program should

support the following operations:

• push: add an element to the top of the stack

• pop: remove the element at the top of the stack

• peek: get the element at the top of the stack without removing it

• is_empty: check if the stack is empty

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

class Stack
{

private:
    char* stack;
    int top;
    int stackSize;

public:

    Stack()
    {
        stackSize = 100;
        stack = new char[stackSize];
        top = -1;
    }

    void push(char value) {
        if (top >= stackSize - 1) {
            cout << "Stack is Full" << endl;
        }
        else {
            top++;
            stack[top] = value;
        }
    }

    void pop() {
        if (top < 0) {
            cout << "Stack is empty" << endl;
        }
        else {
            top--;
        }
    }

    bool is_empty() {
        return top < 0;
    }

    char peek()
    {
        if (top < 0) {
            cout << "Stack is empty" << endl;
            return '\0';
        }
        else {
            return stack[top];
        }
    }

    int getPriority(char ch)
    {
        if (ch == '+' || ch == '-')
        {
            return 1;
        }
        else if (ch == '*' || ch == '/')
        {
            return 2;
        }
        else if (ch == '^')
        {
            return 3;
        }
        return 0;
    }

    string infixToPostfix(string infix)
    {
        string postfix = "";
        Stack stack;

        for (int i = 0; i < infix.length(); i++)
        {
            char ch = infix[i];
            if (isdigit(ch))
            {
                postfix += ch;
            }
            else if (ch == '(')
            {
                stack.push(ch);
            }
            else if (ch == ')')
            {
                while (!stack.is_empty() && stack.peek() != '(')
                {
                    postfix += stack.peek();
                    stack.pop();

                }
                stack.pop();
            }
            else
            {
                while (!stack.is_empty() && getPriority(ch) <=
getPriority(stack.peek()))
                {
                    postfix += stack.peek();
                    stack.pop();
                    cout << postfix << endl;
                }
                stack.push(ch);
            }
        }

        while (!stack.is_empty())
        {
            postfix += stack.peek();
            stack.pop();
        }

        return postfix;
    }
};


int main()
{
    Stack s;
    string infix = "(5+2)*7+6/3";
    cout << "Infix expression: " << infix << endl;
    cout << "Postfix expression: " << s.infixToPostfix(infix) << endl;

    return 0;
}
```