Assignment 3 – Queue

Name: Kartik Banshi Katkar

Rollno : 29

Prn - 12111367

SY-IT-A

**Q. Implement**

**a. Circular Queue**

**b. Reverse Stack using Queue**

**Code:**

#include <iostream>

#include <cmath>

#include <stack>

using namespace std;

int Balancedp()

{

    string expr;

    cout<<"\n Enter the Expression you want to check \n";

    cin>>expr;

    // Declare a stack to hold the previous brackets.

    stack<char> temp;

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

        if (temp.empty()) {

            // if the stack is empty just push the current

            // bracket

            temp.push(expr[i]);

        }

        else if ((temp.top() == '(' && expr[i] == ')')

                 || (temp.top() == '{' && expr[i] == '}')

                 || (temp.top() == '[' && expr[i] == ']')) {

            // if we found any complete pair of bracket then pop the element

            temp.pop();

        }

        else {

            temp.push(expr[i]);

        }

    }

    if (temp.empty()) {

        // if stack is empty return

        cout << "Balanced Paranthesis";

        return 0;

    }

    cout << "Not Balanced Parenthesis";

    return 0;

}

int prec(char c)

{

    if (c == '^')

        return 3;

    else if (c == '/' || c == '\*')

        return 2;

    else if (c == '+' || c == '-')

        return 1;

    else

        return -1;

}

int infixtopostfix()

{

    string s;

    cout<<"\n Enter the Expression you want to solve \n";

    cin>>s;

    stack<char> st; // For stack operations, we are using built in c++ stack

    string result;

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

        char c = s[i];

        // If the scanned character is

        // an operand, add it to output string.

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')

            || (c >= '0' && c <= '9'))

            result += c;

        // If the scanned character is an

        // ‘(‘, push it to the stack.

        else if (c == '(')

            st.push('(');

        // If the scanned character is an ‘)’,

        // pop and to output string from the stack

        // until an ‘(‘ is encountered.

        else if (c == ')') {

            while (st.top() != '(') {

                result += st.top();

                st.pop();

            }

            st.pop();

        }

           else {

            while (!st.empty()

                   && prec(s[i]) <= prec(st.top())) {

                if (c == '^' && st.top() != '^')

                    break;

                else {

                    result += st.top();

                    st.pop();

                }

            }

            st.push(c);

        }

    }

    // Pop all the remaining elements from the stack

    while (!st.empty()) {

        result += st.top();

        st.pop();

    }

    cout << result << endl;

    return 0;

}

float scanNum(char ch) {

   int value;

   value = ch;

   return float(value-'0');   //return float from character

}

int isOperator(char ch) {

   if(ch == '+'|| ch == '-'|| ch == '\*'|| ch == '/' || ch == '^')

      return 1;    //character is an operator

   return -1;   //not an operator

}

int isOperand(char ch) {

   if(ch >= '0' && ch <= '9')

      return 1;    //character is an operand

   return -1;   //not an operand

}

float operation(int a, int b, char op) {

   //Perform operation

   if(op == '+')

      return b+a;

   else if(op == '-')

      return b-a;

   else if(op == '\*')

      return b\*a;

   else if(op == '/')

      return b/a;

   else if(op == '^')

      return pow(b,a);    //find b^a

   else

      return INT\_MIN;    //return negative infinity

}

float postfixexpression()

{

     string postfix;

    cout<<"\n Enter the Expression you want to solve \n";

    cin>>postfix;

    int a, b;

   stack<float> stk;

   string::iterator it;

   for(it=postfix.begin(); it!=postfix.end(); it++) {

      //read elements and perform postfix evaluation

      if(isOperator(\*it) != -1) {

         a = stk.top();

         stk.pop();

         b = stk.top();

         stk.pop();

         stk.push(operation(a, b, \*it));

      }else if(isOperand(\*it) > 0) {

         stk.push(scanNum(\*it));

      }

   }

   cout << "The result is: "<<stk.top()<<"\n";

   return 0;

}

int main()

{

    int choice, n;

    do{

    cout << "\n Menu \n 1.Balanced Parenthesis \n 2.Infix to postfix \n 3.Solving postfix expression \n 4.Exit \n\n\n Enter your choice: \t";

    cin >> choice;

    switch (choice)

    {

    case 1:

        Balancedp();

        break;

    case 2:

        infixtopostfix();

        break;

    case 3:

        postfixexpression();

        break;

    case 4:

        cout << "\n successful exit \n";

        break;

    default:

        cout << "\n Invalid Choice \n";

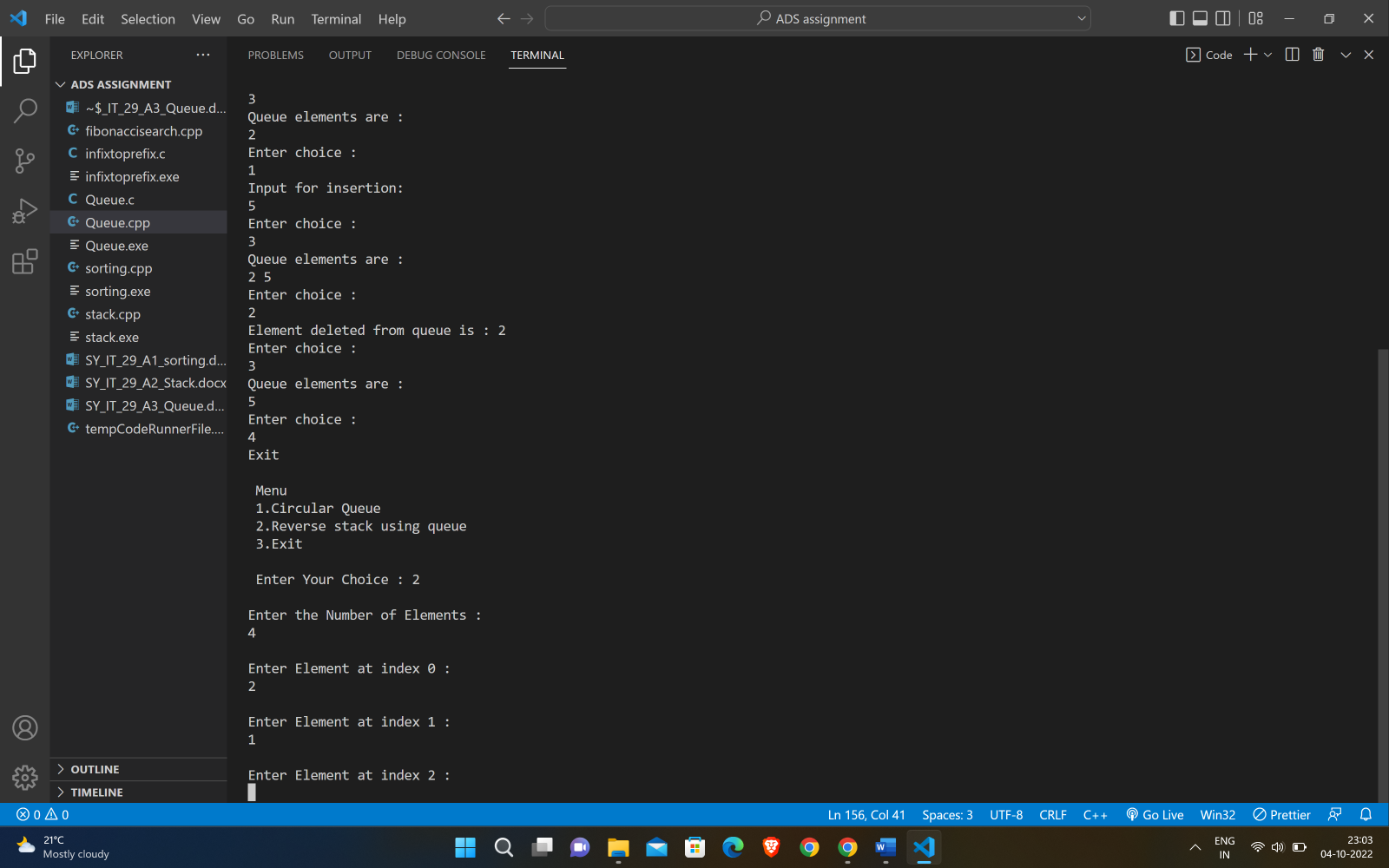
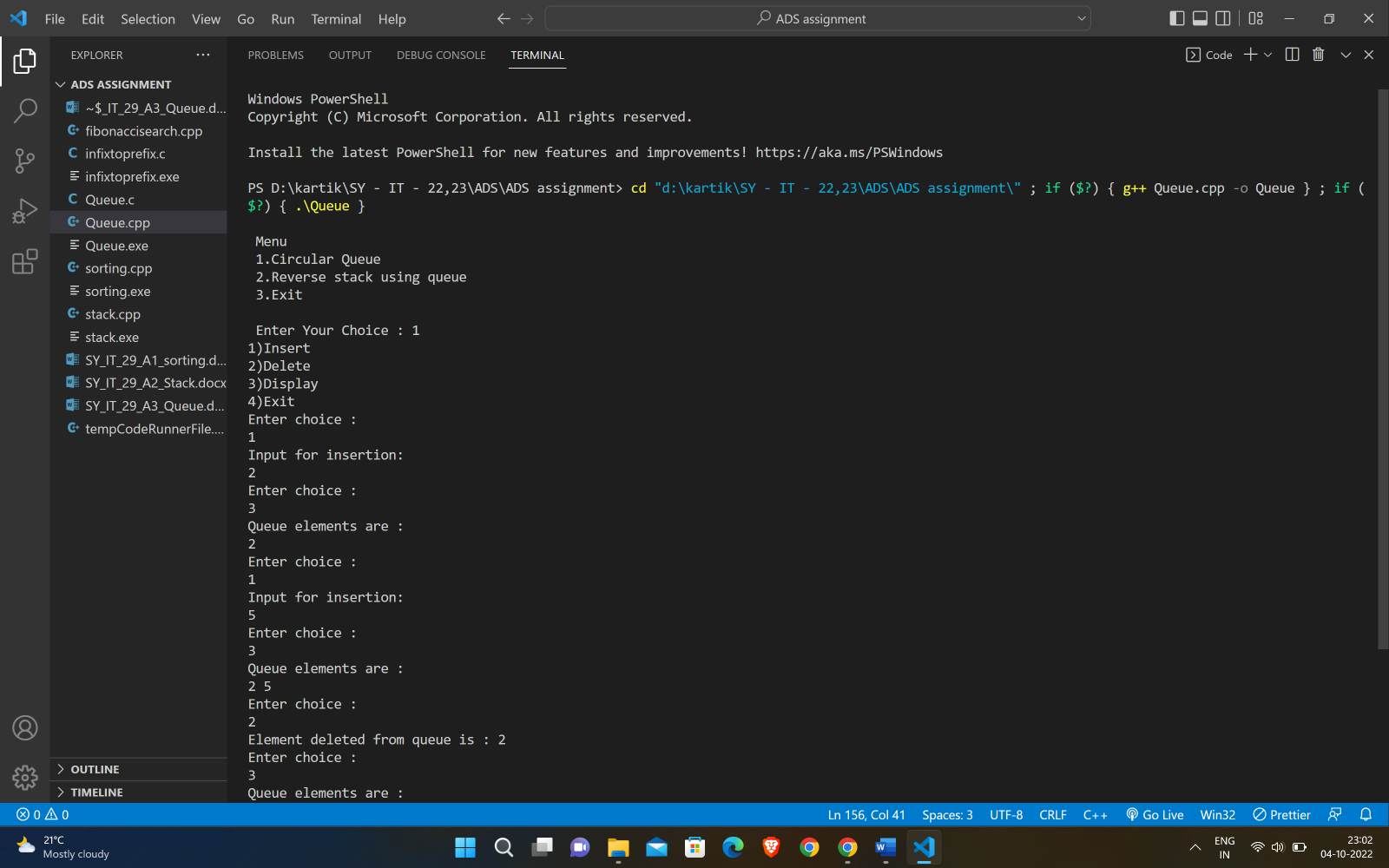
        break;

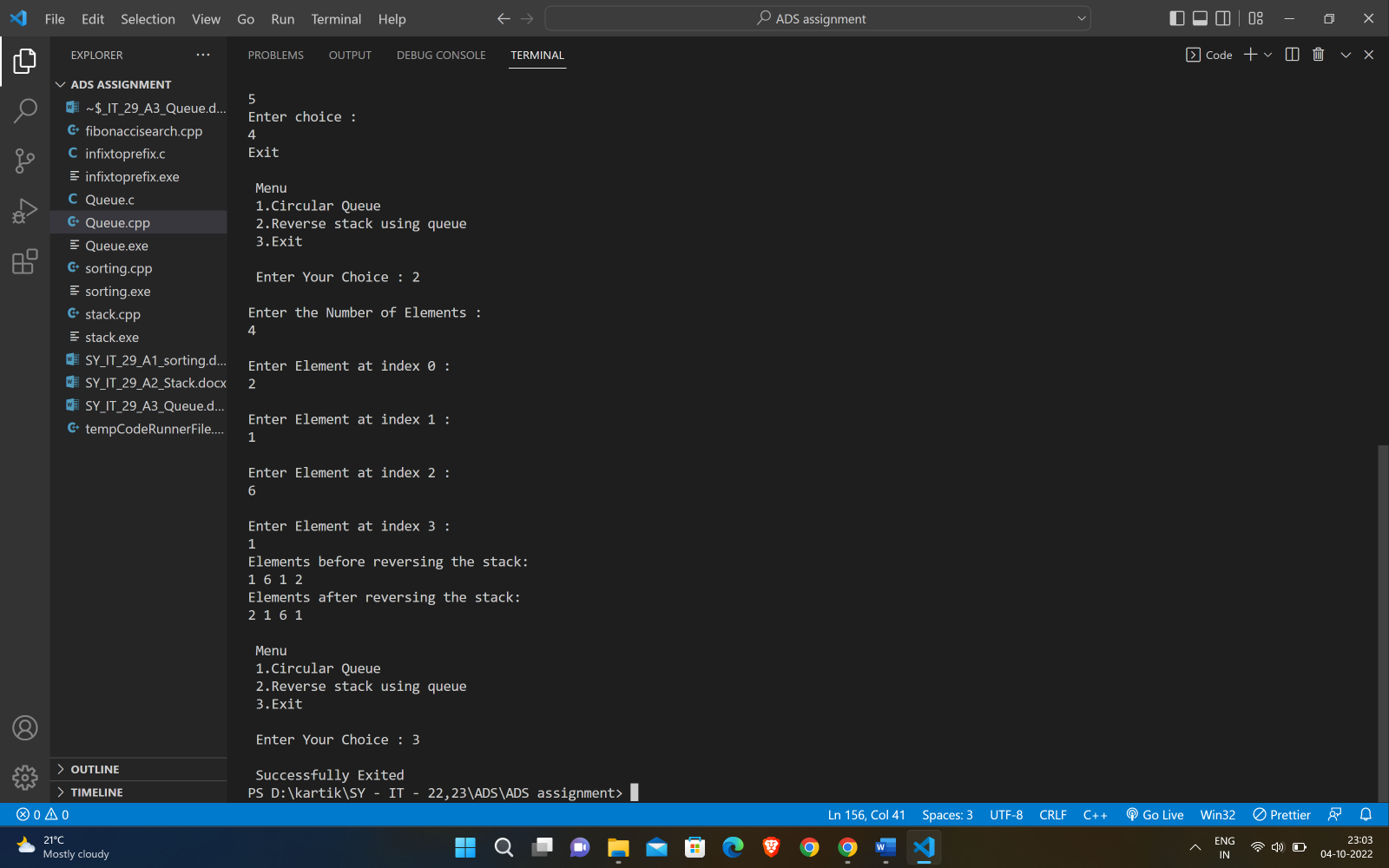
    }

    }while(choice != 4);

}

**Input / output:**

****

****