DSA ASSIGNMENT 3

QUESTION 1

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

int main(){

   cout<<"Enter the size of Array: ";
   int size;
   cin>>size;

   int arr[size];
   int top=-1;

   while(true){
                  cout<<("\nWhat do you want to perform: \n");
                  cout<<("1.Push\n");
                  cout<<("2.Pop\n");
                  cout<<("3.isEmpty\n");
                  cout<<("4.isFull\n");
                  cout<<("5.Display\n");
      cout<<("6.Peek\n");
                  cout<<("7.Exit\n");
                  int num;
                  cin>>num;
                  switch (num) {
```

```
                case 1:{
int num_push;
                cout<<"Enter the number ot push: ";
cin>>num_push;
if(top==size-1){
   cout<<"Stack is full already";
}
else{
   top++;
   arr[top]=num_push;
}
break;
                }
                case 2:{
if(top==-1){
   cout<<"Stack is already empty";
}
else{
   top--;
   cout<<"Removed element: "<<arr[top+1];
}
break;
                }
                case 3:{
if(top==-1){
   cout<<"Stack is empty";
}
else{
```

```
    cout<<"Stack is not empty";
  }
  break;
                  }
                  case 4:{
  if(top==size-1){
    cout<<"Stack is full";
  }
  else{
    cout<<"Stack is not full";
  }
                     break;
                  }
                  case 5:{
  cout<<"\nThe stacks is: ";
  for(int i=0;i<=top;i++){
    cout<<arr[i]<<"\t";
  }
  break;

                  }
                  case 6:{
  cout<<arr[top];
  break;
                  }
case 7:{
  cout << "Exiting";
                     return 0;
```

```
        }
                        default:
                                cout<<"Invalid code! Try again";
                }
    }


    return 0;
}
```

Output

```
Enter the size of Array: 6

What do you want to perform:
1.Push
2.Pop
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
1
Enter the number ot push: 5

What do you want to perform:
1.Push
2.Pop
3.isEmpty
4.isFull
5.Display
6.Peek
7.Exit
5

The stacks is: 5
```

QUESTION 2

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

int main(){

   string word;
   stack<char> reverseWord;

   cout<<"Enter the string you want to reverse: ";
   cin>>word;

   for(char ch:word){
      reverseWord.push(ch);
   }

   cout<<"Reversed string: ";
   while(!reverseWord.empty()){
      cout<<reverseWord.top();
      reverseWord.pop();
   }

   return 0;
}
```

Output

```
Enter the string you want to reverse: DataStructures
Reversed string: serutcurtSataD

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

QUESTION 3

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

bool isBalanced(string exp){
   stack<char> pt;

   for(char ch:exp){
      if(ch=='{' || ch=='(' || ch=='['){
         pt.push(ch);
      }
      else if(ch=='}' || ch==')' || ch==']'){
         if(pt.empty()){
            return false;
         }
         char top=pt.top();
         pt.pop();

         if(ch=='}' && top!='{' || ch==')' && top!='(' || ch==']' && top!='['){
            return false;
         }
      }
   }
```

```
    return pt.empty();
}

int main(){

    string word;
    stack<char> pt;

    cout<<"Enter the expression: ";
    cin>> word;

    bool balanced=isBalanced(word);

    if(balanced){
        cout<<"The expression is balanced";
    }
    else{
        cout<<"Expression is not balanced";
    }

    return 0;
}
```

Output

```
Enter the expression: (a+b) * {c/[d-e]}
The expression is balnced

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

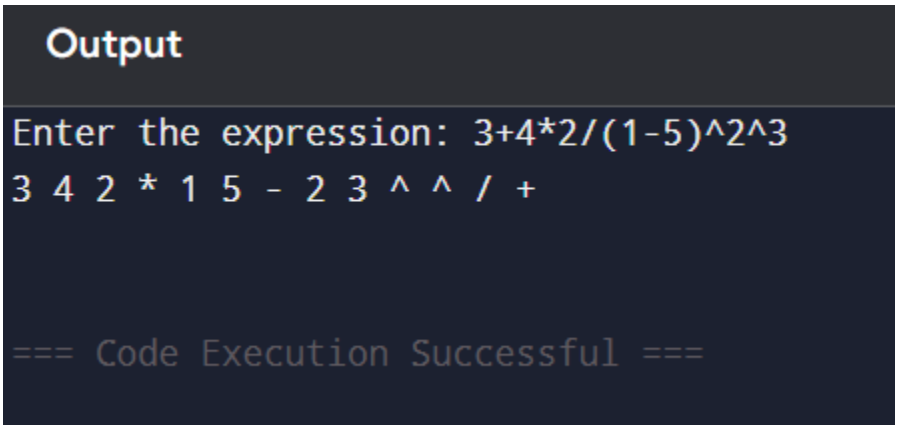
QUESTION 4

```
#include <iostream>
#include <stack>
#include <string>
#include <cctype>

using namespace std;

int prec(char op) {
   if (op == '+' || op == '-') return 1;
   if (op == '*' || op == '/') return 2;
   if (op == '^')              return 3;
   return -1; // non-operator
}

bool isOp(char c) {
   return c=='+' || c=='-' || c=='*' || c=='/' || c=='^';
}

// '^' is right-associative; others are left-associative.
bool isRightAssoc(char op) { return op == '^'; }

int main() {
   cout << "Enter the expression: ";
   string line;
   if (!getline(cin, line)) return 0;
```

```
    string out;
    stack<char> ops;

    auto flushOp = [&](char op){
        out += op;
        out += ' ';
    };

    for (size_t i = 0; i < line.size(); ++i) {
        char c = line[i];

        // skip spaces
        if (isspace(static_cast<unsigned char>(c))) continue;

        // numbers/identifiers (alnum and underscore) — emit as one token
        if (isdigit(static_cast<unsigned char>(c)) || isalpha(static_cast<unsigned char>(c)) || c ==
'_') {
            // collect the whole token
            string token;
            while (i < line.size()) {
                char d = line[i];
                if (isalnum(static_cast<unsigned char>(d)) || d == '_') {
                    token += d;
                    ++i;
                } else {
                    break;
                }
            }
            --i; // step back one because for-loop will ++i
```

```
    out += token;
    out += ' ';
}
else if (c == '(') {
    ops.push(c);
}
else if (c == ')') {
    bool foundOpen = false;
    while (!ops.empty()) {
        char t = ops.top(); ops.pop();
        if (t == '(') { foundOpen = true; break; }
        flushOp(t);
    }
    if (!foundOpen) {
        cerr << "Error: mismatched parentheses.\n";
        return 1;
    }
}
else if (isOp(c)) {
    // pop while top has higher precedence, or same precedence and current is left-assoc
    while (!ops.empty() && isOp(ops.top())) {
        char top = ops.top();
        int pt = prec(top), pc = prec(c);
        bool shouldPop = (pt > pc) || (pt == pc && !isRightAssoc(c));
        if (!shouldPop) break;
        ops.pop();
        flushOp(top);
    }
```

```
            ops.push(c);
        }
        else {
            cerr << "Error: invalid character '" << c << "'.\n";
            return 1;
        }
    }

    // pop remaining operators
    while (!ops.empty()) {
        if (ops.top() == '(' || ops.top() == ')') {
            cerr << "Error: mismatched parentheses.\n";
            return 1;
        }
        flushOp(ops.top());
        ops.pop();
    }

    // trim trailing space if you want (not necessary)
    cout << out << "\n";
    return 0;
}
```


Output
Enter the expression: 3+4*2/(1-5)^2^3
3 4 2 * 1 5 - 2 3 ^ ^ / +
=== Code Execution Successful ===

QUESTION 5

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



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

int main(){

   string expression;
   cout<<"Enter the expression: ";
   cin>>expression;
   stack<int> st;

   for(char c:expression){
```

```
if(isdigit(c)){
    st.push(c-'0');
}

else if(isOperator(c)){
    int num1= st.top();
    st.pop();
    int num2= st.top();
    st.pop();

    if(c=='+'){
        st.push(num2+num1);
    }
    else if(c=='-'){
        st.push(num2-num1);
    }
    else if(c=='*'){
        st.push(num2*num1);
    }
    else if(c=='/'){
        st.push(num2/num1);
    }
    else if(c=='^'){
        st.push(pow(num2,num1));
    }
}
```

```
    }
    int result=st.top();
    st.pop();
    if(!st.empty()){
        cout<<"Error!";
    }
    cout<<"Result: "<<result;

    return 0;
}
```

Output

```
Enter the expression: 23*54*+9-
Result: 17

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