Q1. Write a program to merge the contents of two given flies into a third file.

Code :

```
#include <iostream>
#include <fstream>
using namespace std;
/*
   Contents of File1.txt:
   Name : Mavia Khalid
   Course : MCA

   Contents of File2.txt:
   Roll No. : 22MCA026
   University : Jamia Millia Islamia
*/
int main()
{
   ifstream fin;
   fin.open("File1.txt");
   string line, merged = "";
   while (getline(fin, line))
   {
      merged = merged + line + '\n';
   }
   fin.close();

   fin.open("File2.txt");
   while (getline(fin, line))
   {
      merged = merged + line + '\n';
```

```
    }
    fin.close();
    ofstream fout;
    fout.open("merged.txt");
    fout << merged;
    fout.close();
/*
    Output :
    Contents of merged.txt:
    Name : Mavia Khalid
    Course : MCA
    Roll No. : 22MCA026
    University : Jamia Millia Islamia
*/
}
```

Q2. Write a function in C++ to count and display the number of lines not starting with alphabet 'A' present in a text file "STORY.TXT".

Contents of story.txt:

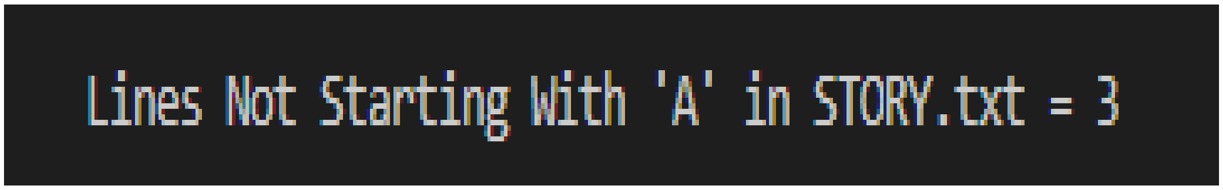
```
The rose is red.
 A girl is playing there.
Numbers are not allowed in the password.
There is a playground.
An aeroplane is in the sky.
```

Code :

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
```

```
{
   ifstream fin;
   string line;
   int count = 0;
   fin.open("STORY.txt");
   while (getline(fin, line))
   {
      if (line[0] != 'A')
         count++;
   }
   fin.close();
   cout << "\n\t Lines Not Starting With 'A' in STORY.txt = " << count << endl << endl;
   return 0;
}
```

OUTPUT :



```
Lines Not Starting With 'A' in STORY.txt = 3
```

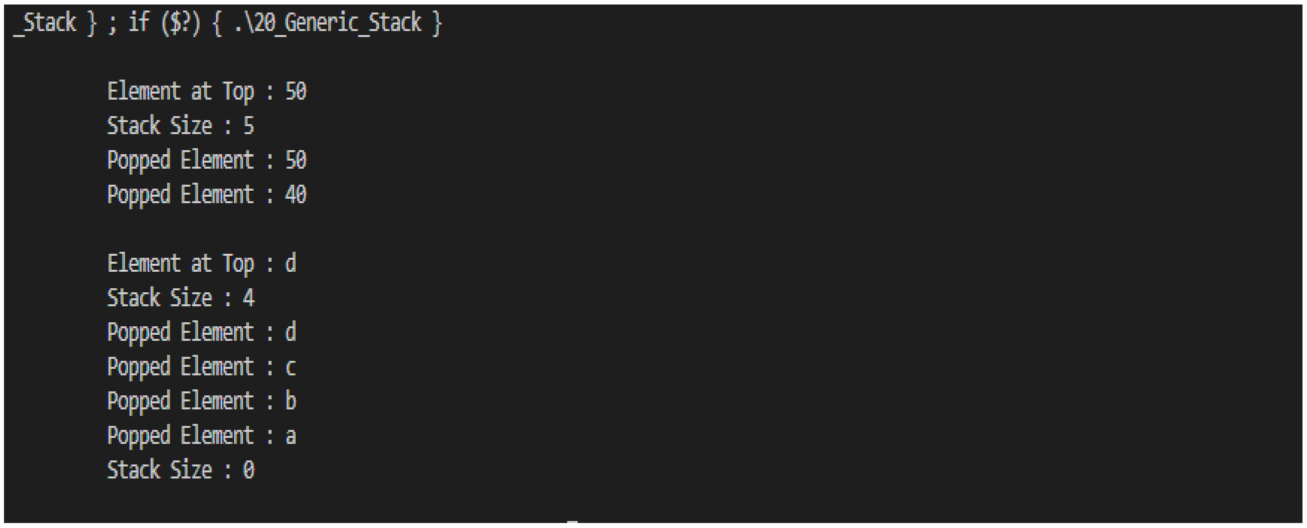
Q3. Write a program using generic stack class to implement all possible stack operations using pointers.

Code :

```
#include<iostream>
using namespace std;
template<class type>
class Stack
```

```
{
    type *arr;
    int top;
    int size;
    public:
    Stack(int size)
    {
        this->size = size;
        arr = new type[size];
        top = -1;
    }
    bool isEmpty()
    {
        if(top == -1)
        {
            return true;
        }
        return false;
    }
    int getSize()
    {
        return top+1;
    }
    type getTop()
    {
        if(top > -1)
        {
            return *(arr+top);
        }
        cout << "\n\t Stack Underflow ";
        return -1;
    }
    void push(type data)
    {
        if(top >= size-1)
        {
            cout << "\n\t Stack Overflow ";
            return;
        }
        top++;
        *(arr+top) = data;
    }
    type pop()
    {
        if(top == -1)
        {
            cout << "\n\t Stack Underflow ";
            return -1;
        }
        type element = *(arr+top);
        top--;
        return element;
    }
};
int main()
{
    Stack<int> s1(30);
    Stack<char> s2(30);
    s1.push(10);
    s1.push(20);
    s1.push(30);
    s1.push(40);
    s1.push(50);

    cout << "\n\t Element at Top : " << s1.getTop() << endl;
    cout << "\t Stack Size : " << s1.getSize() << endl;
    cout << "\t Popped Element : " << s1.pop() << endl;
    cout << "\t Popped Element : " << s1.pop() << endl;

    s2.push('a');
    s2.push('b');
    s2.push('c');
    s2.push('d');

    cout << "\n\t Element at Top : " << s2.getTop() << endl;
    cout << "\t Stack Size : " << s2.getSize();

    for(int i = 0; i < 4; i++)
    {
        cout << "\n\t Popped Element : " << s2.pop();
    }
    cout << "\n\t Stack Size : " << s2.getSize() << endl;

    cout << endl;
    return 0;
}
```

OUTPUT :



```
_Stack } ; if ($?) { .\20_Generic_Stack }

        Element at Top : 50
        Stack Size : 5
        Popped Element : 50
        Popped Element : 40

        Element at Top : d
        Stack Size : 4
        Popped Element : d
        Popped Element : c
        Popped Element : b
        Popped Element : a
        Stack Size : 0
```

Q4. Write a program of your choice to handle the occurring exceptions in the program using multiple catch statements.

Code :

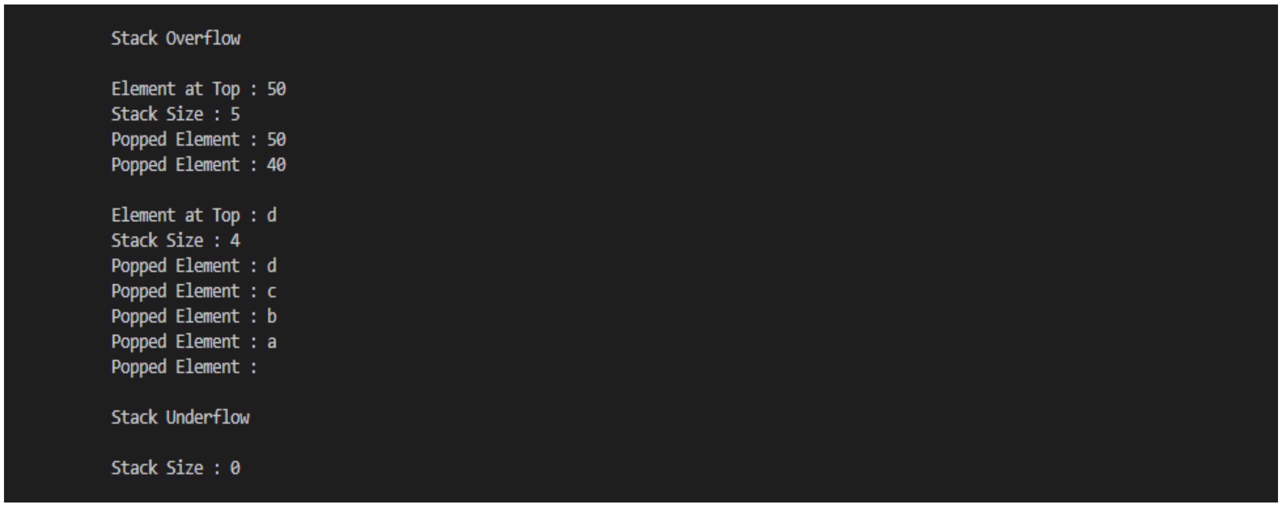
```
#include <iostream>

using namespace std;

class error

{

  string exception;


public:

  error(string e)

  {

    exception = e;

  }

  friend ostream &operator<<(ostream &out, error e)

  {

    return out << e.exception;

  }

};

template <class type>
```

```
class Stack
{
   type *arr;
   int top;
   int size;

public:
   Stack(int size)
   {
      this->size = size;
      arr = new type[size];
      top = -1;
   }
   bool isEmpty()
   {
      if (top == -1)
      {
         return true;
      }
      return false;
   }
   int getSize()
   {
      return top + 1;
   }
   type getTop()
   {
      if (top > -1)
      {
            return *(arr + top);
        }
        throw(error("\n\t Stack Underflow "));
        return -1;
    }
    void push(type data)
    {
        if (top >= size - 1)
        {
            throw(error("\n\t Stack Overflow "));
            return;
        }
        top++;
        *(arr + top) = data;
    }
    type pop()
    {
        if (top == -1)
        {
            throw(error("\n\t Stack Underflow "));
            return -1;
        }
        type element = *(arr + top);
        top--;
        return element;
    }
};
int main()
{
```

```
Stack<int> s1(5);

Stack<char> s2(30);

try

{

   s1.push(10);

   s1.push(20);

   s1.push(30);

   s1.push(40);

   s1.push(50);

   s1.push(60);

}

catch (error &exception)

{

   cout << exception << endl;

}


cout << "\n\t Element at Top : " << s1.getTop() << endl;

cout << "\t Stack Size : " << s1.getSize() << endl;

cout << "\t Popped Element : " << s1.pop() << endl;

cout << "\t Popped Element : " << s1.pop() << endl;


s2.push('a');

s2.push('b');

s2.push('c');

s2.push('d');


cout << "\n\t Element at Top : " << s2.getTop() << endl;

cout << "\t Stack Size : " << s2.getSize();
```

```
    for (int i = 0; i < 4; i++)

    {

        cout << "\n\t Popped Element : " << s2.pop();

    }

    try

    {

        cout << "\n\t Popped Element : " << s2.pop();

    }

    catch(error &exception)

    {

        cout << endl << exception << endl;

    }


    cout << "\n\t Stack Size : " << s2.getSize() << endl;


    cout << endl;

    return 0;

}
```

OUTPUT :



```
Stack Overflow

Element at Top : 50
Stack Size : 5
Popped Element : 50
Popped Element : 40

Element at Top : d
Stack Size : 4
Popped Element : d
Popped Element : c
Popped Element : b
Popped Element : a
Popped Element :

Stack Underflow

Stack Size : 0
```