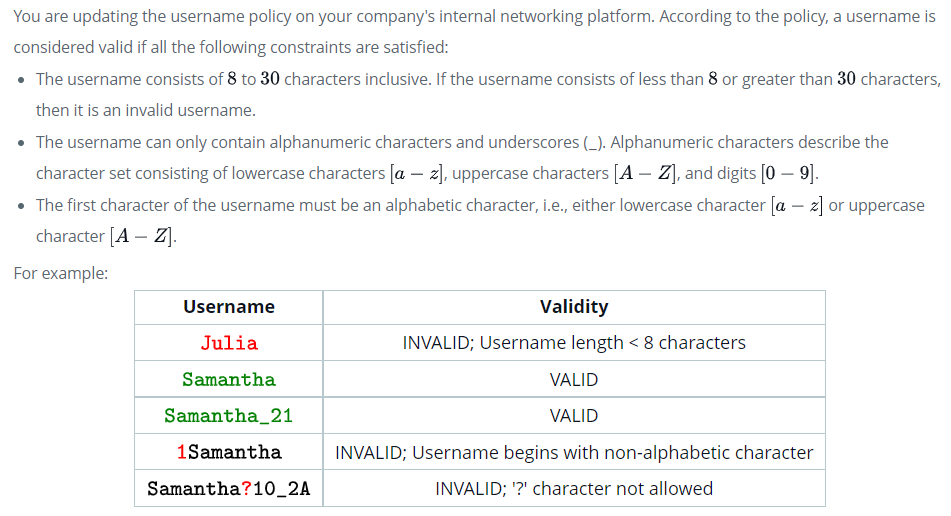
Principles of Programming Language (CS302)

Assignment - 3

**U19CS012**

1.) Write a code that performs **Username Validation** for a website.

**Code**

*#include* <iostream>

using namespace std;

*#define* username\_lower\_limit 8

*#define* username\_upper\_limit 30

string reason;

bool valid\_username(string username)

{

    int special = 0;

    int n = username.length();

*// Length of Username Constraint*

*if* (n < username\_lower\_limit || n > username\_upper\_limit || n <= 0)

    {

*if* (n < username\_lower\_limit)

        {

            string t("Username length < " + to\_string(username\_lower\_limit) + "characters\n");

            reason = t;

        }

*if* (n > username\_upper\_limit)

        {

            string t("Username length > " + to\_string(username\_upper\_limit) + "characters\n");

            reason = t;

        }

*return* false;

    }

*// Starting Character Constraint*

    char start\_ch = username[0];

*// If first character is not {[A-Z]/[a-z]}, then username is invalid*

*if* (!((start\_ch >= 'a' && start\_ch <= 'z') || (start\_ch >= 'A' && start\_ch <= 'Z')))

    {

        string t("First character is not {[A-Z]/[a-z]}\n");

        reason = t;

*return* false;

    }

*// All remaining Characters is not {\_/[A-Z]/[a-z]/[0-9]}, then username is invalid*

*for* (int i = 1; i < n; i++)

    {

        char ch = username[i];

*if* (!((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9') || ch == '\_'))

        {

            string t("Characters is Not from {\_/[A-Z]/[a-z]/[0-9]}, i.e. Special Character Used!\n");

            reason = t;

*return* false;

        }

    }

*// If all the 3 Constraints are Satisfied, then Username is Valid*

*return* true;

}

int main()

{

*// Number of Usernames to be Checked*

    int n;

    cout << "Enter the Number of Usernames to Check for Validity : ";

    cin >> n;

    string s[n];

*for* (int i = 0; i < n; i++)

    {

        cin >> s[i];

    }

    cout << endl;

*for* (int i = 0; i < n; i++)

    {

*if* (valid\_username(s[i]))

        {

            cout << s[i] << " -> Valid Username\n\n";

        }

*else*

        {

            cout << s[i] << " -> Invalid Username\n";

            cout << "Reason : " << reason << endl;

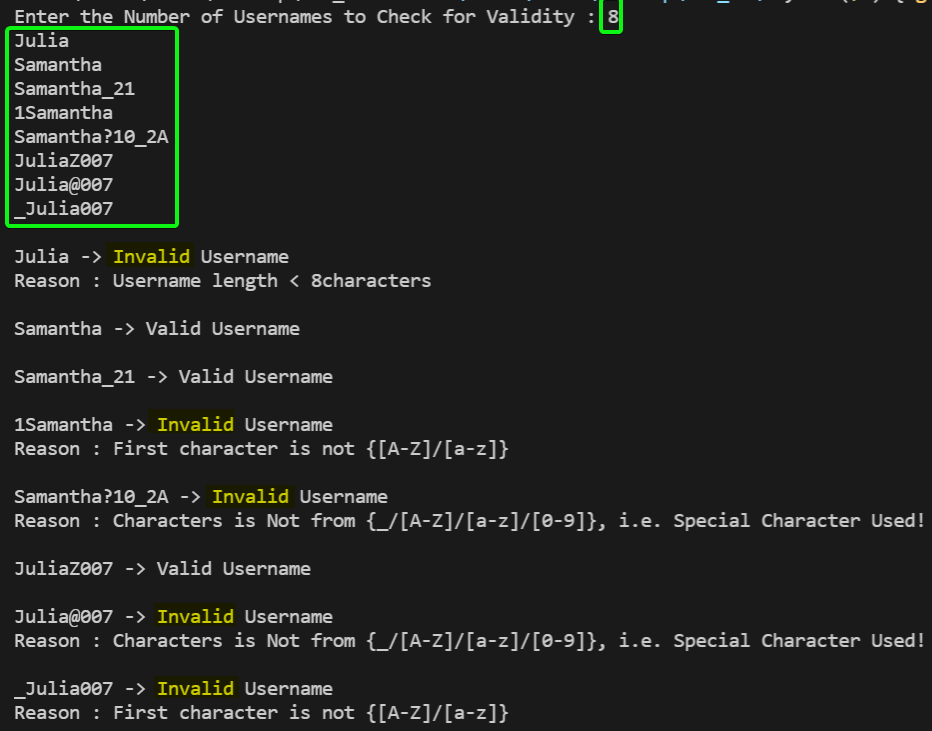
        }

    }

*return* 0;

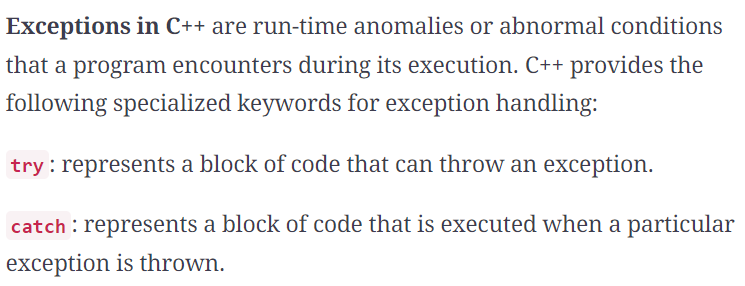
}

**Output**



2.) You are required to **handle error messages** while working with a small computational server that performs complex calculations. It has a function that takes **2 large numbers** as its input and returns a numeric result. Unfortunately, there are various exceptions that may occur during execution. Write a program so that it **prints appropriate error messages**. The expected behavior is defined as follows:

* If the compute function runs fine with the given arguments, then print the result of the function call.
* If it fails to allocate the memory that it needs, print Not enough memory.
* If any other standard C++ exception occurs, print Exception: S where S is the exception's error message.
* If any non-standard exception occurs, print Other Exceptions.



**Code**

*#include* <iostream>

*#include* <vector>

*// exception is for the user to inherit and define their own exceptions.*

*#include* <exception>

*// stdexcept is for catching and handling the standard exceptions*

*#include* <stdexcept>

*// [U19CS012] BHAGYA VINOD RANA*

using namespace std;

typedef long long int ll;

ll complex\_function(ll a, ll b)

{

*if* (b < 0)

*throw* invalid\_argument("B is Negative\n");

*// Allocated a Vector of Size 'b' -> for bad\_alloc Exception*

    vector<ll> vec(b, 1);

*// For other Exception Code*

*if* (!(a ^ b))

    {

*throw* int(a ^ b);

    }

    cout << "Output of Complex computation : ";

*return* (vec[b - 1]) ^ a | b;

}

int main()

{

    ll a, b;

    cout << "Enter Two Numbers for Complex Computation : ";

    cin >> a >> b;

*// Try Catch Block for Exception Handling*

*try*

    {

        cout << complex\_function(a, b) << "\n";

    }

*catch* (bad\_alloc &ba)

    {

*// Exception thrown on failure allocating memory*

        cerr << "bad\_alloc Exception caught : " << ba.what() << endl;

        cerr << "Not Enough Memory\n";

    }

*catch* (exception &e)

    {

        cerr << "Exception : " << e.what() << endl;

    }

*catch* (int e)

    {

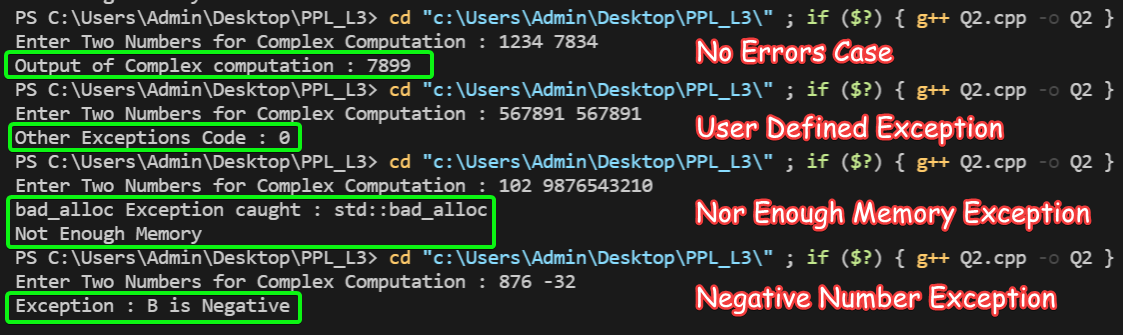
        cerr << "Other Exceptions Code : " << e << endl;

    }

*return* 0;

}

**Output**



3.) Create a class Polar that represents the points on the plane as **polar coordinates** (radius and angles).

Create an **overloaded + operator** for addition of two Polar quantities. "Adding" two points on the plane can be accomplished by adding their X coordinates and then adding their Y coordinates. This gives the X and Y coordinates of the "answer."

Thus you'll need to convert two sets of polar coordinates to rectangular coordinates, add them, then convert the resulting rectangular representation back to polar. You need to use the following trigonometric formulae:

x = r\*cos(a);

y = r\*sin(a);

a = atan(y/x); //arc tangent

r = sqrt(x\*x + y\*y);

**Code**

*#include* <iostream>

*// for sin(), cos(), tan(), sqrt()*

*#include* <cmath>

*// [U19CS012] BHAGYA VINOD RANA*

using namespace std;

const double pi = 3.14159265358979323846;

*// Polar Class with '+' Operator Overloaded*

class polar

{

private:

    double radius;

*// theeta in degrees*

    double angle;

public:

*// Default Constructor*

    polar() : radius(0.0), angle(0.0) {}

*// Parameterized Constructor*

    polar(double r, double t)

    {

        radius = r;

        angle = t;

    }

*// For Displaying the Output in Polar Form (r,0)*

    void display\_polar\_form()

    {

        cout << "Radius (R) = " << radius << " , Theeta (0) = " << angle << " Degree(s)\n";

    }

*// Overload the '+' Operator*

    polar operator+(polar a)

    {

*// Final r' and theta 0°*

        double r, t;

        double x1, x2, y1, y2, x3, y3;

*// Trigonometry F(x) -> single mandatory argument in radians.*

*// x = r\*cos(a);*

        x1 = radius \* cos(pi \* angle / 180);

        x2 = a.radius \* cos(pi \* a.angle / 180);

*// y = r\*sin(a);*

        y1 = radius \* sin(pi \* angle / 180);

        y2 = a.radius \* sin(pi \* a.angle / 180);

*// Add them*

        x3 = x1 + x2;

        y3 = y1 + y2;

*// Radius of Final r' = sqrt((x')\*(x') + (y')\*(y'));*

        r = sqrt((x3 \* x3) + (y3 \* y3));

*// 0 = tan inverse of (y'/x')*

        t = atan(y3 / x3);

*// convert Back to degrees*

        t = t \* 180 / pi;

*return* polar(r, t);

    }

};

int main()

{

*// Number of Points*

    int n;

    cout << "Enter the Number of Polar Points : ";

    cin >> n;

    double rad[n], angle[n];

    polar P[n];

*for* (int i = 0; i < n; i++)

    {

        cout << "Enter the Polar Co-Ordinates of Point " << i + 1 << " [ Radius, Angle(in degree) ] : \n";

        cin >> rad[i] >> angle[i];

        P[i] = polar(rad[i], angle[i]);

    }

    polar resultant(0, 0);

*for* (int i = 0; i < n; i++)

    {

        P[i].display\_polar\_form();

        resultant = resultant + P[i];

    }

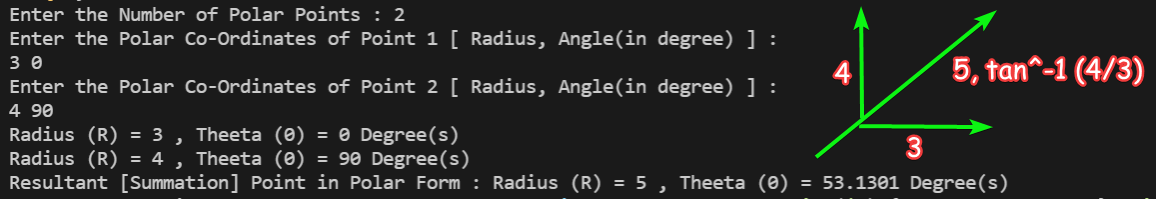
    cout << "Resultant [Summation] Point in Polar Form : ";

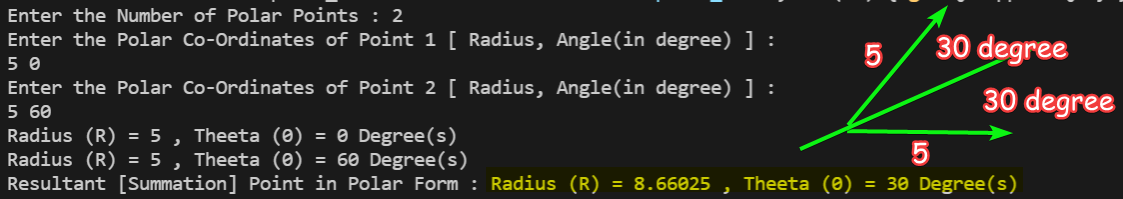
    resultant.display\_polar\_form();

*return* 0;

}

**Output**





4.) A file contains a list of telephone numbers in the following form:

John 2347038256

Ken 9841920261

The names contain only one word and the **names** and **telephone numbers** are separated by white spaces. Write a program to read a file and display its contents in two columns. *The* ***names*** *should be* ***left justified*** and the **number** ***right justified***.

**Code**

*#include* <iostream>

*// For setw()*

*#include* <iomanip>

*// For String*

*#include* <string>

*// For File Reading*

*#include* <fstream>

*// U19CS012 [BHAGYA VINOD RANA]*

using namespace std;

int main()

{

    string file\_name;

    cout << "Enter the Name of the File to be Formatted [Name No] : ";

    cin >> file\_name;

*// ifstream: Stream class to read from files*

    ifstream ifstream\_obj;

    ifstream\_obj.open(file\_name);

    string name;

    long long int telephone\_no;

*if* (ifstream\_obj)

    {

*while* (ifstream\_obj >> name >> telephone\_no)

        {

*// Adjust output to the left*

            cout << left << setw(15) << name;

*// Adjust output to the right*

            cout << right << setw(15) << telephone\_no;

            cout << endl;

        }

        ifstream\_obj.close();

    }

*else*

    {

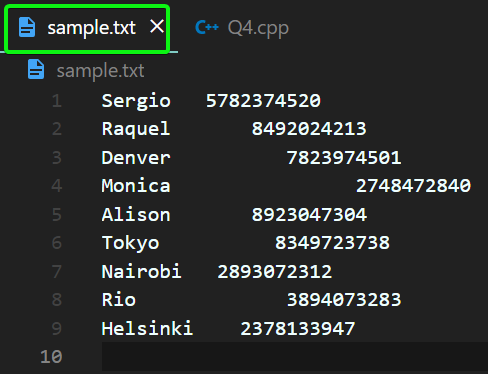
        cout << file\_name << " named File Does Not Exist!\n";

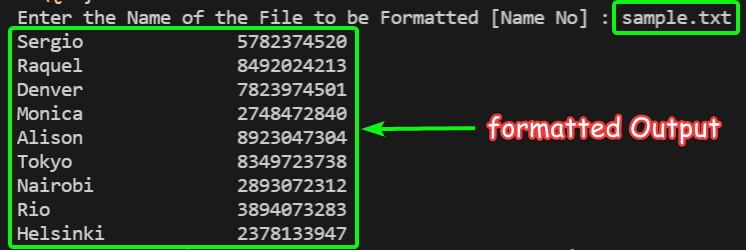
    }

*return* 0;

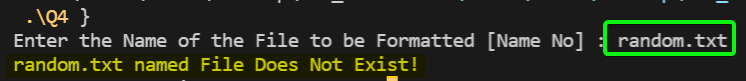
}

**Output**





If the File Does Not exist, it Reports an Error.



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA