Name: Brijesh Rameshbhai Rohit

Admission number: U19CS009

PRINCIPLES OF PROGRAMMING LANGUAGES

ASSIGNMENT 1

1. Create two classes DM and DB which store the value of distances. DM stores distances in meters and centimeters and DB in feet and inches. Write a program that can read values for the class objects and add one object of DM with another object of DB. Use a friend function to carry out the addition operation. The object that stores the results may be a DM object or DB object, depending on the units in which the results are required. The display should be in the format of feet and inches or meters and centimeters depending on the object on display.

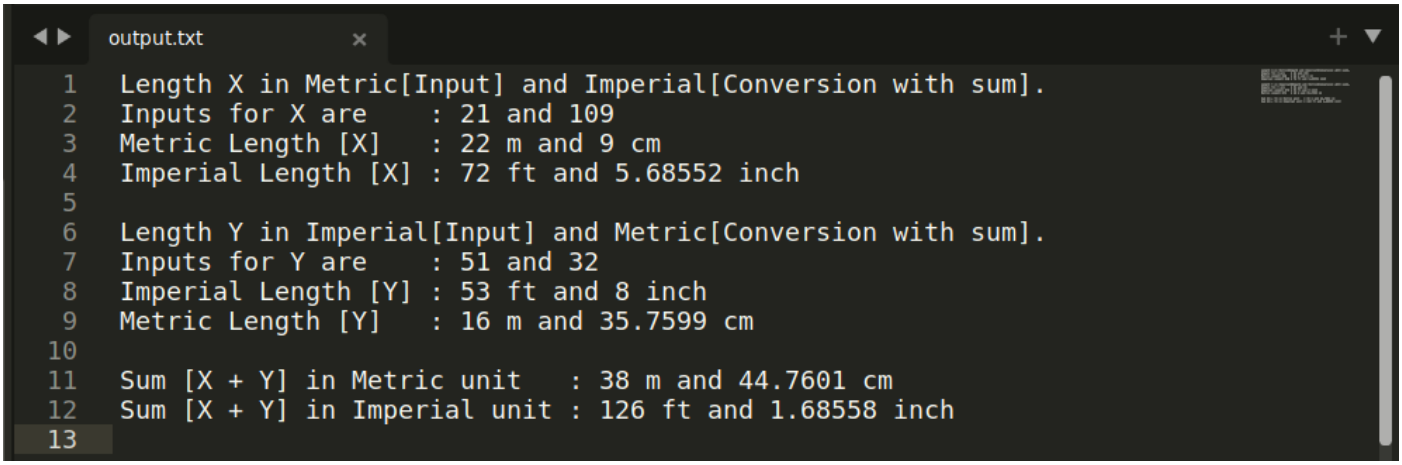
Code :

```
#include<bits/stdc++.h>
using namespace std;
class DB;
class DM {
    float meter;
    float centimeter;
    friend DM addinDM(DM, DB);
    friend DB addinDB(DB, DM);
public:
    DM(float m = 0, float cm = 0)
    {
        if (cm >= 100)
        {
            m += floor(cm / 100);
            cm /= 100;
            cm -= floor(cm);
            cm *= 100;
        }
        this->meter = m;
        this->centimeter = cm;
    }
    float inMeter()
    {
        return this->meter;
    }
    float inCM()
    {
        return this->centimeter;
    }
    void setMeter(float val)
    {
        this->meter = val;
    }
    void setCentimeter(float val)
    {
        this->centimeter = val;
    }
};
```

```
class DB {
    float feet;
    float inch;
    friend DM addinDM(DM, DB);
    friend DB addinDB(DB, DM);
public:
    DB(float ft = 0, float in = 0)
    {
        if (in >= 12)
        {
            ft += floor(in / 12);
            in /= 12;
            in -= floor(in);
            in *= 12;
        }
        this->feet = ft;
        this->inch = in;
    }
    float inFeet()
    {
        return this->feet;
    }
    float inInch()
    {
        return this->inch;
    }
    void setFeet(float val)
    {
        this->feet = val;
    }
    void setInch(float val)
    {
        this->inch = val;
    }
};
DM addinDM(DM obj1, DB obj2)
{
    float x , y;
    //conversion to metric unit
    y = obj2.inch * 2.54;
    y += obj2.feet * 30.48 + obj1.centimeter + (obj1.meter * 100);
    x = floor(y / 100);
    y = (y / 100) - floor(y / 100);
    y *= 100;
    //creating an object in mettric unit after addition
    DM temp(x, y);
    return temp;
}
DB addinDB(DB obj1, DM obj2)
{
    float x , y;
    //conversion to imperial unit
    y = obj2.centimeter * 0.393701;
```

```
    y += obj2.meter * 39.3701 + obj1.inch + (obj1.feet * 12);
    x = floor(y / 12);
    y = (y / 12) - floor(y / 12);
    y *= 12;
    //creating an object in imperial unit after addition
    DB temp(x, y);
    return temp;
}
int main()
{
#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("error.txt", "w", stderr);
    freopen("output.txt", "w", stdout);
#endif
    float m, cm, f, i;
    cin >> m >> cm >> f >> i;
    //Metric length X
    cout << "Length X in Metric[Input] and Imperial[Conversion with sum].\n";
         cout << "Inputs for X are    : " << m << " and " << cm << endl;
    DM metriclen(m , cm);
    DB temp1;
    cout << "Metric Length [X]   : " << metriclen.inMeter() << " m and " <<
metriclen.inCM() << " cm" << endl;
    DB metricinimperial = addinDB(temp1, metriclen);
    cout << "Imperial Length [X] : " << metricinimperial.inFeet() << " ft and "
<< metricinimperial.inInch() << " inch" << endl;
    //Imperial length
    cout << "\nLength Y in Imperial[Input] and Metric[Conversion with sum].\n";
         cout << "Inputs for Y are    : " << f << " and " << i << endl;
    DB imperiallen(f, i);
    DM temp2;
    cout << "Imperial Length [Y] : " << imperiallen.inFeet() << " ft and " <<
imperiallen.inInch() << " inch" << endl;
    DM imperialinmetric = addinDM(temp2, imperiallen);
    cout << "Metric Length [Y]   : " << imperialinmetric.inMeter() << " m and "
<< imperialinmetric.inCM() << " cm" << endl;
    DM metricsum = addinDM(metriclen, imperiallen);
    DB imperialsum = addinDB(imperiallen, metriclen);
    cout << "\nSum [X + Y] in Metric unit   : " << metricsum.inMeter() << " m
and " << metricsum.inCM() << " cm" << endl;
    cout << "Sum [X + Y] in Imperial unit : " << imperialsum.inFeet() << " ft
and " << imperialsum.inInch() << " inch" << endl;
    return 0;
}
```

Output:



2.Find errors, if any, in the following C++ statements.

a) long float x;

Error : 'long float' is not a data type in C++;

b) char *cp = vp; // vp is a void pointer

Error : a null pointer cannot be assigned to char pointer

c) int code = three; // three is an enumerator

Compiled Successfully.

d) int sp = new; // allocate memory with new

Error : which type of allocation not mentioned.

e) enum (green, yellow, red);

Error : syntax error, no name for enumerated values, also pair of bracket used is wrong, curly braces should be used instead of parenthesis.

f) int const sp = total;

Compiled Successfully.

g) const int array_size;

Error : when a const variable is declared, it must be initialized at the same time.

h) for (i=1; int i<10; i++) cout << i << "/n";

Error : if i was not declared, than error will be given in the condition part, and will throw error : "redeclaration of int i", else i was not declared.

i) int & number = 100;

Error : references cannot be made to constant value.

j) float *p = new int 1101;

Error : Invalid data specifier, cannot convert int pointer to constant to float pointer in initialization.

k) int public = 1000;

Error : Invalid identifier, as public is an access specifier and cannot be used as identifier.

l) char name[33] = “USA”;

Compiled Successfully.

3. Assume that a bank maintains two kinds of accounts for customers, one called a savings account and the other as a current account. The savings account provides simple interest and withdrawal facilities but no cheque book facility. The current account provides a check book facility but no interest. Current account holders should also maintain a minimum balance and if the balance falls below this level, a service charge is imposed.

Create a class account that stores customer name, account number and type of account. From this derive the classes cur_acct and sav_acct to make them more specific to their requirements. Include necessary member functions in order to achieve the following tasks:

a. Accept deposits from a customer and update the balance.
b. Display the balance.
c. Compute and deposit interest.
d. Permit withdrawal and update the balance.
e. Check for the minimum balance, impose penalty, necessary and update the balance.
f. Do not use any constructors. Use member functions to initialize the class members.

Code :