Principles of Programming Language (CS302)

Assignment - 1

**U19CS012**

1.) Create two classes DM and DB which store the value of distances.

* DM stores distances in metres 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 metres and centimeters depending on the object on display.

**Code**

*#include* <iostream>

using namespace std;

*// [U19CS012 - BHAGYA VINOD RANA]*

*// DB Class {Stores the Distance in Feet and Inches}*

class DB;

*// DM Class {Stores the Distance in Metres and Centimeters}*

class DM;

*// DM Class {Stores the Distance in Metres and Centimeters}*

class DM

{

    double meter, centi;

public:

    void getdata()

    {

        cout << "\nEnter the Distance in (Meter-Centimeter) : ";

        cin >> meter >> centi;

    }

    void display()

    {

        cout << "\nThe Distance is : ";

        cout << meter << " Meters and " << centi << " Centimeter.";

    }

    friend void add(DM &, DB &);

};

*// DB Class {Stores the Distance in Feet and Inches}*

class DB

{

    double inch, feet;

public:

    void getdata()

    {

        cout << "\nEnter the Distance in (Feet-Inch) : ";

        cin >> feet >> inch;

    }

    void display()

    {

        cout << "\nThe Distance is : ";

        cout << feet << " Feet and " << inch << " Inch.";

    }

    friend void add(DM &, DB &);

};

*// Friend Functoin to Carry Out Addition Operation*

void add(DM &a, DB &b);

int main()

{

    DM a;

    DB b;

*// Read Values from Class Objects*

    a.getdata();

    b.getdata();

*// Call the Friend Function to Add Both the Objects in Different Units*

    add(a, b);

}

*// Friend Functoin to Carry Out Addition Operation*

void add(DM &a, DB &b)

{

    int ch;

    cout << "\nEnter 1 -> Meter-Centi Output : ";

    cout << "\nEnter 2 -> Feet-Inch Output : ";

    cout << "\nEnter your choice : ";

    cin >> ch;

*if* (ch == 1)

    {

        DM d;

*// Convert all to Common 'cm' Denominator*

*// 1 Meter = 100 cm, 1 cm = 1 cm, 1 Feet = 30.48 cm, & Round Off to Nearest cm*

        int c = ((a.meter \* 100) + (a.centi) + (b.feet \* 30.48) + (b.inch \* 2.54));

*if* (c >= 100)

        {

            d.meter = c / 100;

            d.centi = c % 100;

        }

*else*

        {

            d.meter = 0;

            d.centi = c;

        }

        d.display();

    }

*else*

    {

        DB d;

*// Convert all to Common 'inches' Denominator*

*// 1 Meter = 39.3701 inch, 1 cm = 0.3937 inch, 1 Feet = 12 inch, & Round Off to Nearest inch*

        int i = ((a.meter \* 39.3701) + (a.centi \* 0.393701) + (b.feet \* 12) + (b.inch));

*if* (i >= 12)

        {

            d.feet = i / 12;

            d.inch = i % 12;

        }

*else*

        {

            d.feet = 0;

            d.inch = i;

        }

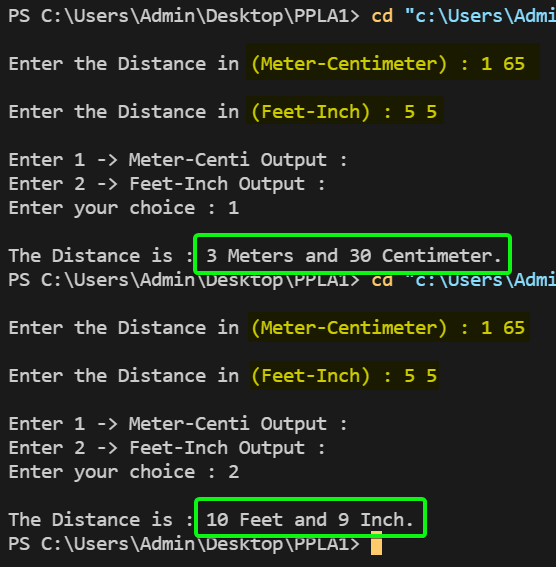
        d.display();

    }

}

**Output**

|  |  |  |  |
| --- | --- | --- | --- |
| Input | meter-centi | feet-inch | centimeter |
| Object1 | 1 m, 65 cm | 5 feet, 5 inch | 165 cm |
| Object2 | 1 m, 65 cm | 5 feet, 5 inch | 165 cm |
| Total | 3 m, 30 cm | 10 feet, 9 inch | 330 cm |



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

1. long float x;

Error – **Yes**, Too Many Datatypes

Correction – long x; or float x;

1. char \*cp = vp; // vp is a void pointer

Error – **Yes,** PointerType must be same on both side

Correction - char \*cp = (char\*) vp;

1. int code = three; // three is an enumerator

Error - **No**

1. int sp = new; // allocate memory with new

Error – **Yes,** syntax Error

Correction – int \*p=new int[10];

1. enum (green, yellow, red);

Error – **Yes**, tag name missing.

Correction – enum **color**(green,yellow,red);

1. int const sp = total;

Error – **Yes**, address have to assign instead of content

Correction - int const\* p = &total;

1. const int array\_size;

Error – **Yes**, C++ requires a const to be initialized at time of defination

Correction - const int array\_size = 5;

1. for (i=1; int i<10; i++) cout << i << “/n”;

Error – **Yes,** undefined symbol i

Correction - for (int i=1; int i<10; i++) cout << i << “/n”;

1. int &number = 100;

Error – **Yes,** invalid variable name

Correction **-** int number = 100;

1. float \*p = new int 1101;

Error – **Yes**, wrong data type

Correction - float \*p = new float[10];

1. int public = 1000;

Error – **Yes**, keyword can not be used as a variable name.

Correction - int public1 = 1000;

1. char name[33] = “USA”;

Error – **Yes,** array size of char must be larger than the number of characters in the string.

Correction - char name[4] = “USA”;

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:

1. **Accept deposits** from a customer and update the balance.
2. **Display** the balance.
3. **Compute** and deposit interest.
4. **Permit withdrawal** and update the balance.
5. Check for the **minimum balance**, impose penalty, necessary and update the balance.
6. Do not use any constructors. Use **member functions** to initialize the class members.

**Code**

*#include* <iostream>

*#include* <string.h>

*#include* <string>

*// [U19CS012 BHAGYA VINOD RANA]*

*// Miinimum Balance*

*#define* minimum 500

*// Service Charge in case if amount is less than minimum balance*

*#define* service\_charge 100

*// Rate of Interest*

*#define* r 0.10

using namespace std;

*// Account Class*

class account

{

protected:

*// Customer Name*

    string name;

*// Account Number*

    int ac\_number;

*// Account Type*

    string ac\_type;

public:

*// Member Function to Create Account of type 't'*

    void create\_acc();

};

*// Current Account Derived from Account Class*

class cur\_acct : public account

{

private:

    double balance;

public:

    void deposit(double d);

    void withdraw(double w);

    void display();

};

*// Saving Account Derived from Account Class*

class sav\_acct : public account

{

    double balance;

    int d, m, y;

public:

    void deposit(double d);

    void withdraw(double w);

    void display();

    void set\_date(int a, int b, int c)

    {

        d = a;

        m = b;

        y = c;

    }

    void interest();

};

*// --------------------------------------------------------------------------------------------*

*// Main Function*

int main()

{

    sav\_acct raju;

    raju.create\_acc();

*// Accept Deposits*

    double d;

    cout << " Enter your Deposit Amount : ";

    cin >> d;

    raju.deposit(d);

    raju.display();

    int t;

    cout << "\n press 1 to see your Interest : \n"

         << " press 0 to skip : ";

    cin >> t;

*if* (t == 1)

        raju.interest();

*// Permit Withdrawal and update balance*

    cout << "\n Enter your Withdrawal Amount :";

    double w;

    cin >> w;

    raju.withdraw(w);

    raju.display();

*return* 0;

}

*// -------------------------------MEMBER F(X) OF ACCOUNT CLASS-----------------------------------*

*// Member Function to Create Account of type 't'*

void account::create\_acc()

{

    cout << " Enter Customer Name : ";

    cin >> name;

    cout << "Account Type" << endl;

    cout << " 1 -> Saving\n 2 -> Current\n ";

    cout << "Enter Account Type {1/2} : ";

    int ch;

    cin >> ch;

*if* (ch == 1)

        ac\_type = "savings";

*else*

        ac\_type = "current";

    string s;

*do*

    {

        cout << " Enter Account Number [8-digits] : ";

        cin >> ac\_number;

        s = to\_string(ac\_number);

*if* (s.length() != 8)

            cout << "Please Enter Valid Account Number!\n";

    } *while* (s.length() != 8);

    cout << "\nAccount Successfully Made!\n\n";

}

*// -----------------------------MEMBER F(X) OF CURR ACCOUNT CLASS--------------------------------*

void cur\_acct::deposit(double d)

{

    balance += d;

}

void cur\_acct::withdraw(double w)

{

*if* (balance < w)

        cout << " Sorry! Insufficient Balance!\n";

*else*

    {

        balance -= w;

*if* (balance < minimum)

        {

            cout << "\n Your current balance is :" << balance << " which is less than" << minimum << "\n your account is discharged by " << service\_charge << "Rs \n"

                 << " You must store " << minimum << "Rs to avoid discharge\n "

                 << " Do you want to Withdraw ? Press 1 -> YES OR Press 0 -> NO \n"

                 << " What is your Choice ?";

            int opt;

            cin >> opt;

*if* (opt == 0)

                balance += w;

        }

    }

}

void cur\_acct::display()

{

    cout << "\n Account Balance = " << balance << "\n";

}

*// -----------------------------MEMBER F(X) OF SAVING ACCOUNT CLASS------------------------------*

void sav\_acct::deposit(double d)

{

    int x, y, z;

    cout << " Enter Date of Deposit (i,e day,month,year) : ";

    cin >> x >> y >> z;

    set\_date(x, y, z);

    balance = d;

}

void sav\_acct::withdraw(double w)

{

*if* (balance < w)

        cout << " Sorry! Insufficient Balance!\n";

*else*

    {

        balance -= w;

*if* (balance < minimum)

        {

            cout << "\n Your current balance is :" << balance << " which is less than" << minimum << "\n your account is discharged by " << service\_charge << "Rs \n"

                 << " You must store " << minimum << "Rs to avoid discharge\n "

                 << " Do you want to Withdraw ? Press 1 -> YES OR Press 0 -> NO \n"

                 << " What is your Choice ?";

            int opt;

            cin >> opt;

*if* (opt == 0)

                balance += w;

        }

    }

}

void sav\_acct::display()

{

    cout << "\n Account Balance : " << balance << endl;

}

void sav\_acct::interest()

{

*// No of Days in Different Month of Years*

    int D[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    int d1, y1, m1;

    cout << " Enter Today's Date (i,e day,month,year) : ";

    cin >> d1 >> m1 >> y1;

    int iday, fday;

    iday = d;

    fday = d1;

*for* (int i = 0; i < m1; i++)

        fday += D[i];

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

        iday += D[i];

    int tday;

*// Final - Initial Days = Total Interest Days*

    tday = fday - iday;

    double ty;

    ty = double(tday) / 365 + (y1 - y);

    double intrst;

*// SI = (P\*R\*T)*

    intrst = balance \* r \* ty;

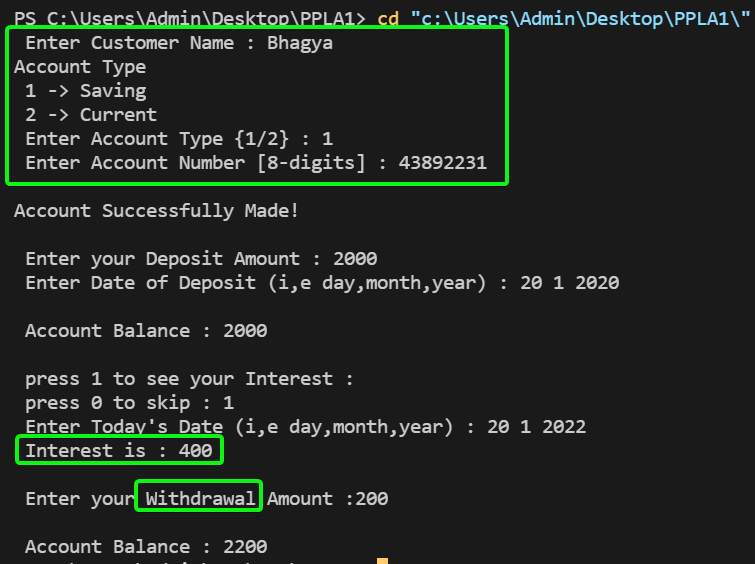
    cout << " Interest is : " << intrst << "\n";

*// Add interest to Balance Amount*

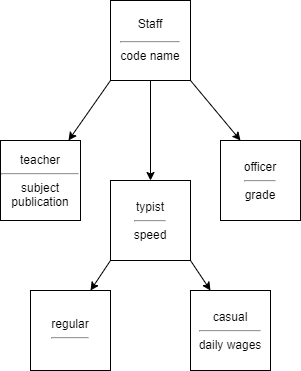
    balance += intrst;

}

**Output**



4.) An educational institution wishes to maintain a database of its Employees. The database is divided into a number of classes whose **hierarchical relationships** are shown in the following figure. The figure also shows the minimum information required for each class. Specify **all classes and define functions** to create the database and retrieve individual information as and when required.



The database created does not include educational information of the staff. It has been decided to add this information to **teachers and officers** (and not for typists) which will help management in decision making with regard to training, promotions etc.

Add another data class called **education** that holds two pieces of educational information namely **highest qualification** in general education and highest professional qualification. This class should be inherited by the class's teacher and officer.

**Code**

*#include* <iostream>

*#include* <iomanip>

*#include* <string>

*#include* <string.h>

using namespace std;

*// Staff Class*

class staff

{

protected:

*// Staff Code & Name*

    int code;

    string name;

public:

    void set\_info(string n, int c)

    {

        name = n;

        code = c;

    }

};

*// Education added for Staff*

class education : public staff

{

protected:

    string quali;

public:

    void set\_qualification(string q) { quali = q; }

};

*// Teacher Class*

class teacher : public education

{

protected:

*// Subject and Publication*

    string sub, publication;

public:

*// To Intialize the Teacher's Details*

    void set\_details(string s, string p)

    {

        sub = s;

        publication = p;

    }

*// To Display the Teachers Information*

    void show()

    {

        cout << " Name " << setw(8) << " Code " << setw(15)

             << " Subject " << setw(22) << " Publication "

             << setw(25) << " Qualification " << endl

             << name << setw(8) << code << setw(25)

             << sub << setw(18) << publication << setw(25) << quali << endl;

    }

};

*// Officer's Class*

class officer : public education

{

*// Officer Grade*

    string grade;

public:

    void set\_details(string g)

    {

        grade = g;

    }

*// To Display the Officers Information*

    void show()

    {

        cout << " Name " << setw(15) << " Code " << setw(15) << " Category "

             << setw(22) << " Qualification " << endl

             << name << setw(10)

             << code << setw(15) << grade << setw(25) << quali << endl

             << endl;

    }

};

*// Typist Class*

class typist : public staff

{

protected:

    float speed;

public:

    void set\_speed(float s)

    {

        speed = s;

    }

};

*// Regular Typist which inherits Publicly from Typist Class*

class regular : public typist

{

protected:

    float wage;

public:

    void set\_wage(float w) { wage = w; }

    void show()

    {

        cout << " Name " << setw(10) << " Code " << setw(10) << " Speed "

             << setw(10) << " Wage " << endl

             << name << setw(10) << code

             << setw(15) << speed << setw(15) << wage << endl

             << endl;

    }

};

*// Casual Typist which inherits Publicly from Typist Class*

class causal : public typist

{

    float wage;

public:

    void set\_wage(float w) { wage = w; }

    void show()

    {

        cout << " Name " << setw(16) << " Code " << setw(15) << " Speed "

             << setw(15) << " Wage " << endl

             << name << setw(10) << code

             << setw(15) << speed << setw(15) << wage << endl

             << endl;

    }

};

int main()

{

*// Teacher*

    teacher t;

    t.set\_info("Akbar", 710);

    t.set\_details("Programming with c++", "Tata McGraw Hill");

    t.set\_qualification("PHD from Standford");

*// Officer*

    officer o;

    o.set\_info("Ramesh", 155);

    o.set\_details("First class");

    o.set\_qualification("2 years experienced");

*// Regular Typist*

    regular rt;

    rt.set\_info("Rohan", 310);

    rt.set\_speed(85);

    rt.set\_wage(25000);

*// Casual Typist*

    causal ct;

    ct.set\_info("Jethalal", 205);

    ct.set\_speed(60);

    ct.set\_wage(20000);

    cout << "\nTeacher Info : " << endl;

    t.show();

    cout << "\nOfficer Info : " << endl;

    o.show();

    cout << "\nRegular Typist Info : " << endl;

    rt.show();

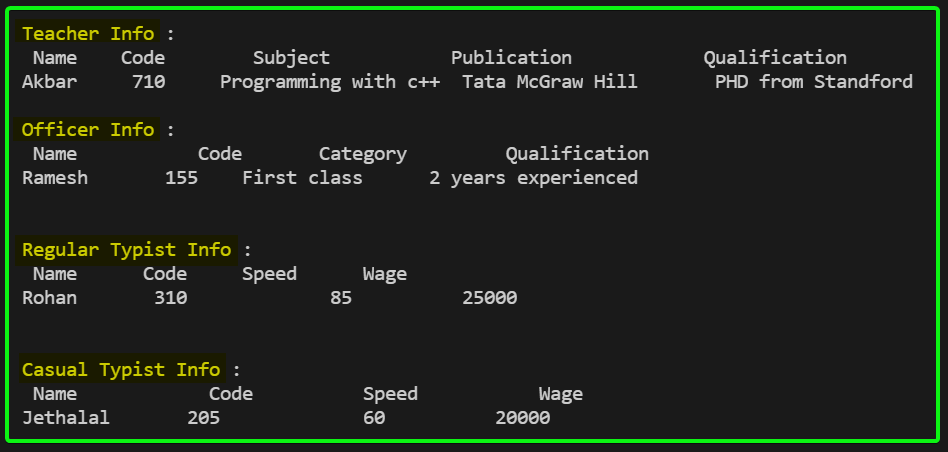
    cout << "\nCasual Typist Info : " << endl;

    ct.show();

*return* 0;

}

**Output**



**SUBMITTED BY**: U19CS012

BHAGYA VINOD RANA