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 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.

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

// CLASS DM
class DM {
// class variables
private:
float meters;
float centimeters;

public:
// constructor
DM(float f, float i) {
meters = f;
centimeters = i;
}
// getters
float getMeters() {
return meters;
}
float getCentimeters() {
return centimeters;
}
// setters
void setMeters(float f) {
meters = f;
}
void setCentimeters(float i) {
centimeters = i;
}
// other functions
void display() {
cout << meters << " Meters " << centimeters << " Centimeters" << endl;
}

// add function for diff classes
```

```
void add(DM dm) {
float f = dm.getMeters();
float i = dm.getCentimeters();
float m = meters + f;
float c = centimeters + i;
if (c >= 100) {
m++;
c -= 100;
}
meters = m;
centimeters = c;
display();
}
};

// CLASS DB
class DB {
// class variables
private:
float feet;
float inches;

public:
// constructor
DB(float f, float i) {
feet = f;
inches = i;
}
// getters
float getFeet() {
return feet;
}
float getInches() {
return inches;
}
// setters
void setFeet(float f) {
feet = f;
}
void setInches(float i) {
inches = i;
}
// other functions
void display() {
cout << feet << " feet " << inches << " inches" << endl;
}
```

```
void add(DB db) {
float f = db.getFeet();
float i = db.getInches();
float m = feet + f;
float c = inches + i;
if (c >= 12) {
m++;
c -= 12;
}
feet = m;
inches = c;
display();
}
};

// add function for diff classes
void add(DM dm, DB db) {
float f = dm.getMeters();
float i = dm.getCentimeters();
float m = f + db.getFeet()*0.3048;
float c = i + db.getInches()*2.54;
if (c >= 12) {
m++;
c -= 12;
}
cout << m << " Meters " << c << " Centimeters" << endl;
}

int main() {
// DM object
DM dm1(1, 1);
DM dm2(4, 5);
DB db1(2, 2);

// add dm1 and dm2
dm1.add(dm2);
// add dm1 and db1
add(dm1, db1);
db1.display();
return 0;
}

// Admission no: U19CS019
// Author: Naman Khater
```

```
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$ ./q1.out
5 Meters 6 Centimeters
5.6096 Meters 11.08 Centimeters
2 feet 2 inches
```

2. Find errors, if any, in the following C++ statements.
    1. long float x;
    2. char *cp = vp; // vp is a void pointer
    3. int code = three; // three is an enumerator
    4. int sp = new; // allocate memory with new
    5. enum (green, yellow, red);
    6. int const sp = total;
    7. const int array_size;
    8. for (i=1; int i<10; i++) cout << i << “/n”;
    9. int & number = 100;
    10. float *p = new int 1101;
    11. int public = 1000;
    12. char name[33] = “USA”;

Line No.	Error	Correction
1	Too many types	float x; or double x;
2	type must be matched	char *cp = (char*) vp;
3	no error	
4	syntax error	int*p=new int[10];
5	tag name missing	enum color (green, yellow, red)
6	address have to assign instead of content	int const*p = &total;
7	C++ requires a const to be initialized	const int array-size = 5;
8	undefined symbol i	for(int i=1;i<10;i++) cout<<i<<“/n”;
9	invalid variable name	int number = 100;
10	wrong data type	float *p = new float[10];
11	keyword can not be used as a variable name	int public1 = 1000;
12	array size of char must be larger than the number of characters in the string	chat 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.

```
#include <iostream>
#include <bits/stdc++.h>

using namespace std;

enum account_type {
CURRENT=1,
SAVINGS
};

// Class Account
class Account {
public:
string name;
long long int acc_no;
account_type type;
double balance;
double min_bal;
double penalty;
double interest_rate;
void deposit(double amount) {
balance += amount;
cout << "Deposited " << amount << " to " << name << "'s account.\n";
}

void withdraw(double amount) {
if (balance - amount < min_bal) {
balance -= amount + penalty;
```

```
cout << "Withdrew " << amount << " from " << name << "'s account. " << penalty << "
penalty imposed.\n";
} else {
balance -= amount;
cout << "Withdrew " << amount << " from " << name << "'s account.\n";
}
}

void display() {
for (int i = 0; i < 25; i++) cout << "-";
cout << "\nName: " << name << endl;
cout << "Account Number: " << acc_no << endl;
cout << "Account Type: " << type << endl;
cout << "Balance: " << balance << endl;
for (int i = 0; i < 25; i++) cout << "-";
cout << "\n";
}

void penalty_calc() {
if (balance < min_bal) {
balance -= penalty;
cout << "Penalty imposed.\n";
} else {
cout << "No penalty imposed.\n";
}
}

void compute_interest() {
cout << "Interest added: " << balance*interest_rate << endl;
balance += balance * interest_rate;
cout << "Current balance: " << balance << endl;
}
};

// Class Current Account
class CurrentAccount : public Account {
public:
CurrentAccount(string name) {
this->name = name;
this->type = CURRENT;
this->balance = 0;
this->min_bal = 5000;
this->penalty = 150;
this->interest_rate = 0;
long long int acc_no = 0;
for (int i = 0; i < 12; i++) {
```

```
acc_no *= 10;
acc_no += rand() % 10;
}
this->acc_no = acc_no;
}

void compute_interest() {
cout << "Interest is not applicable to current accounts.\n";
}
};

// Class Savings Account
class SavingsAccount : public Account {
public:
SavingsAccount(string name) {
this->name = name;
this->type = SAVINGS;
this->balance = 0;
this->min_bal = 0;
this->penalty = 0;
this->interest_rate = 0.05;
long long int acc_no = 0;
for (int i = 0; i < 12; i++) {
acc_no *= 10;
acc_no += rand() % 10;
}
this->acc_no = acc_no;
}

void penalty_calc() {
cout << "Penalty is not applicable to savings accounts.\n";
}

};

// Help (print instructions)
void help() {
cout << "Enter the following commands:\n";
cout << "1. deposit\n";
cout << "2. withdraw\n";
cout << "3. display\n";
cout << "4. compute_interest\n";
cout << "5. penalty_calc\n";
cout << "0. exit\n";
}
```

```
int main () {
// HELP
cout << "For help, type 'help'\n";
string fname, lname;
cout << "Enter full name: ";
cin >> fname >> lname;
cout << "Account type: (type 1 for current, 2 for savings): ";
int type;
cin >> type;
CurrentAccount *cacc;
SavingsAccount *sacc;
if (type == 1) {
cacc = new CurrentAccount(fname + " " + lname);
} else {
sacc = new SavingsAccount(fname + " " + lname);
}
int balanace;
cout << "Enter initial balance: ";
cin >> balanace;
if (type == 1) {
cacc->deposit(balanace);
cacc->display();
} else {
sacc->deposit(balanace);
sacc->display();
}
while (true) {
string input;
cout << "Enter command: ";
cin >> input;
if (input == "help") {
help();
} else if (input == "1") {
double amount;
cout << "Enter amount: ";
cin >> amount;
if (type == 1) {
cacc->deposit(amount);
cacc->display();
} else {
sacc->deposit(amount);
sacc->display();
}
} else if (input == "2") {
double amount;
cout << "Enter amount: ";
```

```
cin >> amount;
if (type == 1) {
cacc->withdraw(amount);
cacc->display();
} else {
sacc->withdraw(amount);
sacc->display();
}
} else if (input == "3") {
if (type == 1) {
cacc->display();
} else {
sacc->display();
}
} else if (input == "4") {
if (type == 1) {
cacc->compute_interest();
cacc->display();
} else {
sacc->compute_interest();
sacc->display();
}
} else if (input == "5") {
if (type == 1) {
cacc->penalty_calc();
cacc->display();
} else {
sacc->penalty_calc();
sacc->display();
}
} else if (input == "0") {
break;
} else {
cout << "Invalid command.\n";
}
}
return 0;
}
```

```
For help, type 'help'
Enter full name: Naman Khater
Account type: (type 1 for current, 2 for savings): 1
Enter initial balance: 10000
Deposited 10000 to Naman Khater's account.
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 1
Balance: 10000
------------------------
Enter command: help
Enter the following commands:
1. deposit
2. withdraw
3. display
4. compute_interest
5. penalty_calc
0. exit
Enter command: 2
Enter amount: 8000
Withdrew 8000 from Naman Khater's account. 150 penalty imposed.
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 1
Balance: 1850
------------------------
Enter command: 4
Interest is not applicable to current accounts.
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 1
Balance: 1850
------------------------
Enter command: 0
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$
```

```
For help, type 'help'
Enter full name: Naman Khater
Account type: (type 1 for current, 2 for savings): 2
Enter initial balance: 10000
Deposited 10000 to Naman Khater's account.
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 2
Balance: 10000
------------------------
Enter command: help
Enter the following commands:
1. deposit
2. withdraw
3. display
4. compute_interest
5. penalty_calc
0. exit
Enter command: 4
Interest added: 500
Current balance: 10500
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 2
Balance: 10500
------------------------
Enter command: 5
Penalty is not applicable to savings accounts.
------------------------
Name: Naman Khater
Account Number: 367535629127
Account Type: 2
Balance: 10500
------------------------
Enter command: 0
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$
```

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.

```
#include <bits/stdc++.h>
using namespace std;
class Staff
{
```

```
public:
int code;
string name;
void set_basic_info(int code, string name)
{
this->code = code;
this->name = name;
}
};
class Education : public Staff
{
public:
string qualification;
void set_qualification(string qualification)
{
this->qualification = qualification;
}
};
class Teacher : public Education
{
public:
string subject;
string publication;
void set_details(string subject, string publication)
{
this->subject = subject;
this->publication = publication;
}
void display()
{
cout << "\tCode: " << code << endl;
cout << "\tName: " << name << endl;
cout << "\tQualification: " << qualification << endl;
cout << "\tSubject: " << subject << endl;
cout << "\tPublication: " << publication << endl;
}
};
class Officer : public Education
{
public:
string grade;
void set_details(string grade)
{
this->grade = grade;
}
void display()
```

```
{
cout << "\tCode: " << code << endl;
cout << "\tName: " << name << endl;
cout << "\tQualification: " << qualification << endl;
cout << "\tGrade: " << grade << endl;
}
};
class Typist : public Staff
{
public:
float speed;
void set_speed(float speed)
{
this->speed = speed;
}
};
class Regular : public Typist
{
public:
void display()
{
cout << "\tCode: " << code << endl;
cout << "\tName: " << name << endl;
cout << "\tSpeed: " << speed << endl;
}
};
class Casual : public Typist
{
public:
float daily_wage;
void set_daily_wage(float wage)
{
this->daily_wage = wage;
}
void display()
{
cout << "\n\tName: " << this->name << ",\n\tCode: " << this->code << ",\n\tSpeed: "
<< this->speed << ",\n\tDaily wage: Rs. " << this->daily_wage << endl;
}
};

int main()
{
Teacher t;
t.set_basic_info(1, "Naman");
t.set_qualification("M.Tech");
```

```
t.set_details("CSE", "IEEE");
cout << "\nTeacher";
t.display();
Officer o;
o.set_basic_info(2, "Karan");
o.set_qualification("M.A");
o.set_details("A");
cout << "\nOfficer";
o.display();
Regular r;
r.set_basic_info(3, "Rohan");
r.set_speed(30);
cout << "\nRegular Typist";
r.display();
Casual c;
c.set_basic_info(4, "Rishabh");
c.set_speed(40);
c.set_daily_wage(200);
cout << "\nCasual Typist";
c.display();
return 0;
}
```

```
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$ g++ q4.cpp
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$ ./a.out

Teacher Code: 1
        Name: Naman
        Qualification: M.Tech
        Subject: CSE
        Publication: IEEE

Officer Code: 2
        Name: Karan
        Qualification: M.A
        Grade: A

Regular Typist  Code: 3
        Name: Rohan
        Speed: 30

Casual Typist
        Name: Rishabh,
        Code: 4,
        Speed: 40,
        Daily wage: Rs. 200
maxmax@madmax:~/Desktop/u19cs019_sem6/Principles_of_programming_language/lab2$
```