**Name: Brijesh Rameshbhai Rohit**

**Admission number: U19CS009**

**PRINCIPLES OF PROGRAMMING LANGUAGES**

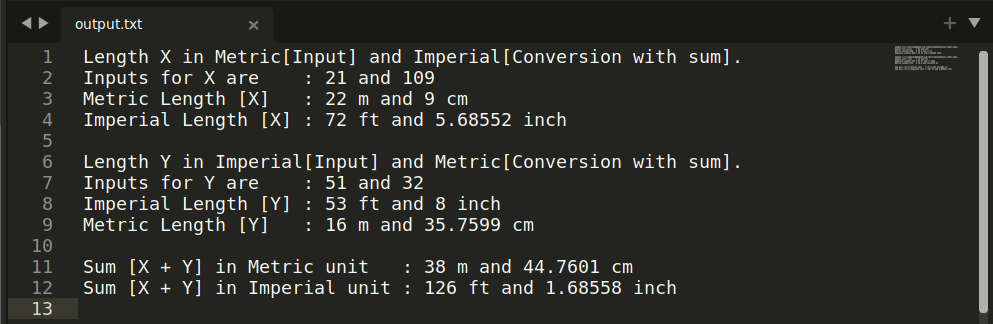
**ASSIGNMENT 1**

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.**

1. **long float x;**

**Error** : ‘long float’ is not a data type in C++;

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

**Error** : a null pointer cannot be assigned to char pointer

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

**Compiled Successfully***.*

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

**Error** : which type of allocation not mentioned.

1. **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.

1. **int const sp = total;**

**Compiled Successfully***.*

1. **const int array\_size;**

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

1. **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.

1. **int & number = 100;**

**Error** : references cannot be made to constant value.

1. **float \*p = new int 1101;**

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

1. **int public = 1000;**

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

1. **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:

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 :

|  |
| --- |
| //U19CS038 SUMIT SHETTY  #include<iostream>  using namespace std;  class account{  protected:  string customer\_name;  long long int account\_number;  string account\_type;  public:  void customerName(string name) {  this->customer\_name = name;  }  void accountNo(long long int acctno) {  this->account\_number = acctno;  }  void acct\_type(string acct) {  this->account\_type = acct;  }  };  class cur\_acct : public account{  static double min\_balance;  static double srv\_charge;  double act\_balance;  public:  void minBalance(double money) {  min\_balance = money;  }  void initCharge(double money) {  srv\_charge = money;  }  void initBalance(double bal) {  this->act\_balance = bal;  }  void money\_deposit(double money) {  this->act\_balance += money;  }  void checkMinBalance() {  if(this->act\_balance<min\_balance) {  cout << "Please maintain minimum balance!!\n";  if(this->act\_balance > srv\_charge){  this->act\_balance -= srv\_charge;  cout << "Amount 50 charged for not maintaining minimum balance!!\n";  }  else  this->act\_balance = 0;  }  else{  cout << "Remaining balance : " << this->act\_balance << "\n";  }  }  void withdrawal(double money) {  if(money <= this->act\_balance)  {  this->act\_balance -= money;  checkMinBalance();  cout << "Money Withdrawal Successful !!!\n\n";  }  else{  cout << "You do not have enough balance in your account!!\n\n";  }  }  void displayBalance() {  cout <<"\nAccount Balance is : "<< (double)this->act\_balance <<"\n\n";  if(this->act\_balance < min\_balance)  {  cout << "Please maintain minimum balance!!!\n\n";  }  }  void acctInfo() {  cout << "\n----------Account Information----------------";  cout << "\nAccount holder : " << this->customer\_name;  cout << "\nAccount no. : " << this->account\_number;  cout << "\nAccount type : " << this->account\_type;  cout << "\nCurrent Balance : " << (double)this->act\_balance;  cout << "\nHaving Chequebook : Yes";  cout << "\n\n" ;  }  };  double cur\_acct::min\_balance{ 500.00 };  double cur\_acct::srv\_charge{ 50.00 };  class sav\_acct : public account{  static double rate;  double interest;  double act\_balance;  public:  void initBalance(double bal) {  this->act\_balance = bal;  }  void money\_deposit(double money) {  this->act\_balance += money;  }  void withdrawal(double money) {  if(money <= this->act\_balance) {  this->act\_balance -= money;  cout << "Money Withdrawal Successful !!!\n";  cout << "Remaining balance : " << this->act\_balance << "\n\n";  }  else  cout << "You do not have enough balance in your account!!\n";  }  void computeInterest(int years) {  interest = (this->act\_balance)\*(rate)\*(years);  cout << "Calculated interest : "<< interest << "\n";  cout << "Final amount after "<< years << "years will be : " << this->act\_balance + interest;  cout << "\n\n";  }  void displayBalance() {  cout << "Account Balance is : " << this->act\_balance <<"\n\n";  }  void acctInfo() {  cout << "\n----------Account Information----------------";  cout << "\nAccount holder : " << this->customer\_name;  cout << "\nAccount no. : " << this->account\_number;  cout << "\nAccount type : " << this->account\_type;  cout << "\nCurrent Balance : " << this->act\_balance;  cout << "\nHaving Chequebook : No";  cout << "\n\n" ;  }  };  double sav\_acct::rate{6.00};  int main()  {  while(1)  {  int accounttype;  cout << "Account types : \n1 : Savings\n2 : Current\nEnter your choice : ";  cin >> accounttype;  if(accounttype == 1)  {  int amt;  sav\_acct sav;  string name;  sav.acct\_type("Savings");  cout << "Enter Account name : ";  cin >> name;  sav.customerName(name);  sav.accountNo(0ll + 311099181000 + rand()%1000);  cout << "Deposit initial balance (greater than zero): ";  cin >> amt;  sav.initBalance(amt);  while(1)  {  cout << "Following operations are allowed : ";  cout << "\n1.Deposit money\n2.Withraw Money\n3.Check Balance in your account\n";  cout << "4.Interest calculation\n5.Account Information\n";  int i;  cout << "Choose any operation you want to perform : ";  cin >> i;  if(i==1) {  double money;  cout << "\nEnter the amount to deposit : ";  cin >> money;  sav.money\_deposit(money);  cout << "Successfully money deposited!!!\n\n";  }  else if(i==2) {  double money;  cout << "\nEnter the amount for withrawal : ";  cin >> money;  sav.withdrawal(money);  }  else if(i==3) {  sav.displayBalance();  }  else if(i==4) {  int years;  cout << "Enter no of years : ";  cin >> years;  sav.computeInterest(years);  }  else if(i==5){  sav.acctInfo();  }  else{  cout << "Invalid Operation\n\n";  break;  }  }  }  else if (accounttype == 2){  string name;  double amt;  cur\_acct curr;  curr.acct\_type("Current");  cout << "\nEnter Account name : ";  cin >> name;  curr.customerName(name);  curr.accountNo(0ll + 311099181000 + rand()%1000);  cout << "\nMinimum Balance in current account : 500.00";  cout << "\nService Charge : 50.00\n";  cout << "\nDeposit initial balance (should be more than minimum balance) : ";  cin >> amt;  curr.initBalance(amt);  while(1)  {  cout << "Following operations are allowed : ";  cout << "\n1.Deposit money\n2.Withraw Money\n3.Check Balance in your account\n";  cout << "4.Account Information\n";  int i;  cout << "Choose any operation you want to perform : ";  cin >> i;  if(i==1) {  double money;  cout << "\nEnter the amount to deposit : ";  cin >> money;  curr.money\_deposit(money);  cout << "Successfully money deposited!!!\n\n";  }  else if(i==2) {  double money;  cout << "\nEnter the amount for withrawal : ";  cin >> money;  curr.withdrawal(money);  }  else if(i==3) {  curr.displayBalance();  }  else if(i==4){  curr.acctInfo();  }  else{  cout << "Invalid Operation\n\n";  break;  }  }  }  int ch;  cout << "Do you want to perform further operation (1:Yes 0:No) ? ";  cin >> ch;  if(ch != 1)  return 0;  }  return 0;  } |