LABORATORY EXERCISES

AEC C++

**LAB PROGRAM 1:**

Given that an EMPLOYEE class contains following members: Data members : Employee\_Number, Employee\_Name, Basic, DA, IT, Net\_Salary Member functions: to read the data, to calculate Net\_Salary and to print data members. Write a C++ program to read the data of N employees and compute Net\_Salary of each employee. (Dearness Allowance (DA) = 52% of Basic and Income Tax

(IT) = 30% of the gross salary . Net\_Salary=gross-IT).

#include<iostream>

using namespace std;

class EMPLOYEE

{

int emp\_num;

string emp\_name;

double Basic,DA,IT,Net\_Salary;

private:void read()

{

cout<<"\nEnter employee name:";

cin>>emp\_name;

cout<<"\n Enter employee num:";

cin>>emp\_num;

cout<<"\n Enter Basic salary:";

cin>>Basic;

}

private: void calc()

{

DA=(52.0/100)\*Basic;

double gross=Basic+DA;

IT=(30.0/100)\*gross;

Net\_Salary=gross-IT;

}

public: void display()

{

read();

calc();

cout<<"\n Employee name: "<<emp\_name;

cout<<"\n Employee num: "<<emp\_num;

cout<<"\n Basic salary: "<<Basic;

cout<<"\n DA: "<<DA<<"\n IT:"<<IT<<"\nNet salary: "<<Net\_Salary;

}

};

int main()

{

int N;

cout<<"\nEnter number of employees:";

cin>>N;

EMPLOYEE emp[N];

for(int i=0;i<N;i++)

{

emp[i].display();

}

}

2. Write a C++ program to create a class called DATE. Accept two valid dates in the form dd/mm/yy. Implement the following operations by overloading the operators + and After every operation display the results by overloading the operator <<.

i. no\_of\_days = d1 – d2; where d1 and d2 are DATE objects, d1 >=d2 and no\_of\_days is an integer. ii. d2 = d1 + no\_of\_days; where d1 is a DATE object and no\_of\_days is an integer.

CODE:

#include <iostream>

#include <sstream>

using namespace std;

class DATE {

private:

int day, month, year;

public:

DATE(int d, int m, int y) :

day(d), month(m), year(y) {}

friend ostream& operator<<(ostream& out, const DATE& date);

friend int operator-(const DATE& d1, const DATE& d2);

friend DATE operator+(const DATE& d, int num);

};

ostream& operator<<(ostream& out, const DATE& date) {

out << date.day << "/" << date.month << "/" << date.year;

return out;

}

int operator-(const DATE& d1, const DATE& d2) {

// Convert both dates to days

int days1 = d1.year \* 365 + d1.month \* 30 + d1.day;

int days2 = d2.year \* 365 + d2.month \* 30 + d2.day;

// Calculate the difference return days1 - days2;

return days1 - days2;

}

DATE operator+(const DATE& d, int num)

{

DATE result = d; // Add 'num' days to the date

result.day += num;

while (result.day > 30) {

result.day -= 30;

result.month++;

if (result.month > 12)

{

result.month = 1;

result.year++;

}

}

return result;

}

int main() {

int day, month, year;

cout << "Enter first date (dd//mm//yyyy): ";

cin >> day >> month >> year;

DATE d1(day, month, year);

cout << "Enter second date (dd//mm//yyyy): ";

cin >> day >> month >> year;

DATE d2(day, month, year);

int no\_of\_days = d1 - d2;

cout << "Difference between d1 and d2: " << no\_of\_days << "days" << endl;

DATE new\_date = d1 + no\_of\_days;

cout << "New date after adding no\_of\_days to d1: " << new\_date << endl;

return 0;

}

**OUTPUT:**

Enter first date (dd//mm//yyyy): 12 10 2024

Enter second date (dd//mm//yyyy): 10 10 2024

Difference between d1 and d2: 2days

New date after adding no\_of\_days to d1: 14/10/2024

3. Write a C++ program to create a class called NUMBER with an Integer data member and member function to set the value for this data member. Derive three classes from this base class called HEXADECIMAL, DECIMAL and OCTAL.

Include a function display() in all these derived classes to display the value of base class data member in hexadecimal, decimal and octal respectively. use multilevel inheritance.

#include<iostream>

using namespace std;

class number

{

public:

int num;

public:

void setNum(int n)

{

num = n;

}

};

class HEXADECIMAL: public number{

public:

void display()

{

cout << "HexaDecimal Value is : "<<hex << num ;}

};

class DECIMAL:public number{

public:

void display()

{ cout << "\nDecimal Value is : " << dec << num;}

};

class OCTAL:public number{

public:

void display()

{

cout << "\nOctal Value is : " << oct << num;}

};

int main()

{

number obj1;

HEXADECIMAL hObj;

DECIMAL dObj;

OCTAL oObj;

obj1.setNum(11);

hObj.setNum(obj1.num);

dObj.setNum(obj1.num);

oObj.setNum(obj1.num);

cout << endl << "Values Displayed in Respective Numbers Systems" << endl;

hObj.display();

dObj.display();

oObj.display();

return 0;

}

4. Design three classes’ student test and results, where result inherited from test and test is inherited from students. Write possible functions to initialize the values. Also write a main function for execution by creating objects.

#include<iostream>

#include<string>

using namespace std;

class Student {

protected:

string name;

int rollNo;

public:

Student(string name, int rollNo) : name(name), rollNo(rollNo){}

void displayStudentInfo() {

cout << "Name: " << name << endl;

cout << "Roll No: " << rollNo << endl;

}

};

// Test class (inherits from Student)

class Test : public Student {

protected:int marks[6];

public:

Test(string name, int rollNo, const int marks[]) : Student(name, rollNo) {

for (int i = 0; i < 6; i++) {

this->marks[i] = marks[i];

}

}

void displayMarks(){

cout << "Marks: ";

for (int i = 0; i < 6; i++) {

cout<< marks[i] << " ";}

cout << endl;

}

};

// Result class (inherits from Test)

class Result: public Test {

private:int totalMarks;

float percentage;

public:

Result(string name, int rollNo, const int marks[]) : Test(name, rollNo, marks) {

calculateTotalAndPercentage();

}

void calculateTotalAndPercentage() {

totalMarks = 0;

for (int i = 0; i < 6; i++) {

totalMarks += marks[i];

}

percentage = (float)totalMarks / 600 \* 100;

}

void displayResult() {

displayStudentInfo();

displayMarks();

cout << "Total Marks: " <<totalMarks << endl;

cout << "Percentage: " << percentage << "%" << endl;

}

};

int main()

{

int marks[] = {85, 92, 78, 89,95, 81};

Result s("John Doe", 123, marks);

s.displayResult();

Result t("J", 124, marks);

t.displayResult();

return 0;

}

# 5.Design a class bank account with current and saving bank account as inherited classes.Class bank account should have following data members: Account number, Name, Balance Amount, and member functions. To initialize the value to deposit and withdrawal amount after checking the minimum balance.

# #include <iostream>

# #include <string>

# using namespace std;

# class BankAccount {

# protected:

# int accountNumber;

# string name;

# double balanceAmount;

# public:

# BankAccount(int accountNumber, const string& name,double balanceAmount): accountNumber(accountNumber),name(name), balanceAmount(balanceAmount) {}

# virtual void deposit(double amount) {

# balanceAmount += amount;

# cout << "Deposit successful. New balance: " << balanceAmount << endl;

# }

# virtual void withdraw(double amount) {

# if (balanceAmount >= amount) {

# balanceAmount -= amount;

# cout << "Withdrawal successful. New balance: "<< balanceAmount << endl;

# }

# else {

# cout << "Insufficientbalance. Cannot withdraw." << endl;

# }

# }

# virtual ~BankAccount() {}

# };

# class SavingsAccount : public BankAccount {

# private:double minimumBalance;

# public:

# SavingsAccount(int accountNumber, const string& name, double balanceAmount, double minimumBalance): BankAccount(accountNumber, name, balanceAmount), minimumBalance(minimumBalance) {}

# void withdraw(double amount) override {

# if (balanceAmount - amount >= minimumBalance) {

# BankAccount::withdraw(amount);

# }

# else {

# cout << "Cannot withdraw. Balance would fall below minimum balance." << endl;

# }

# }

# };

# class CurrentAccount : public BankAccount {

# private:double overdraftLimit;

# public:

# CurrentAccount(int accountNumber, const string& name, double balanceAmount, double overdraftLimit): BankAccount(accountNumber, name, balanceAmount), overdraftLimit(overdraftLimit)

# {}

# void withdraw(double amount) override {

# double allowedWithdrawal = balanceAmount + overdraftLimit;

# if (amount <= allowedWithdrawal) {

# balanceAmount -= amount;

# cout << "Withdrawal successful. New balance: " << balanceAmount << endl;

# }

# else {

# cout << "Cannot withdraw. Overdraft limit would be exceeded." << endl;

# }

# } };

# int main() {

# // Create a savings account with minimum balance

# SavingsAccount savingsAccount(12345, "John Doe", 5000,1000);

# // Deposit and withdraw from savings account

# savingsAccount.deposit(1000);

# savingsAccount.withdraw(3000);

# // Create a current account with limit

# CurrentAccount currentAccount(67890, "Jane Smith", 2000, 500);

# // Deposit and withdraw from current account

# currentAccount.deposit(1500);

# currentAccount.withdraw(3000);

# return 0;

# }

# Output:

# Deposit successful. New balance: 6000

# Withdrawal successful. New balance: 3000

# Deposit successful. New balance: 3500

# Withdrawal successful. New balance: 500

6.Write a C++ program for designing a calculator for addition, subtraction, multiplication, and division.

#include <iostream>

using namespace std;

// Template Calculator Class template <typename T> class Calculator {

public:

T add(T a, T b) { return a + b;

}

T subtract(T a, T b) { return a - b;

}

T multiply(T a, T b) {

return a \* b;

}

T divide(T a, T b) { if (b != 0) {

return a / b;

} else {

cout << "Error: Division by zero is not allowed." << endl; return 0;

}

}

};

int main() {

Calculator<double> calc; // Change the type here if needed (e.g., int, float, double)

double num1, num2;

char operation;

cout << "Enter first number: "; cin >> num1;

cout << "Enter second number: "; cin >> num2;

cout << "Select operation (+, -, \*, /): ";

cin >> operation; switch (operation) {

case '+':

cout << "Result: " << calc.add(num1, num2) << endl; break;

case '-':

cout << "Result: " << calc.subtract(num1, num2) << endl; break;

case '\*':

cout << "Result: " << calc.multiply(num1, num2) << endl; break;

case '/':

cout << "Result: " << calc.divide(num1, num2) << endl; break;

default:

cout << "Invalid operation." << endl; break;

}

return 0;}

**7.Construct a C++ program to read the content from sample.txt and print the same on the output screen**.

#include <iostream>

#include<fstream>

using namespace std;

int main()

{

char product[20]; int price;

cout<<"Enter product name and price : \n ";

cin>>product>>price;

ofstream outfile(“product.txt");

if(!outfile)

{

cout<<"Error couldn't open the file for writing \n";

return 1;

}

outfile<<product<<endl;

outfile<<price<<endl;

ifstream infile(“product.txt");

if(!infile)

{

cout<<"Error couldn't open the file for reading \n"; return 1;

}

infile>>product;

infile>>price;

cout<<"The product read : "<<product<<endl;

cout<<"The price read : "<<price << endl;

return 0;

}

8.Write a program to calculate final velocity using the formula: v=u+a\*t, with initial velocity, acceleration and time as input.

#include <iostream>

using namespace std;

int main() {

float v,u,a,t ;

cout<<"Enter initial velocity \n" ;

cin>> u ;

cout<<"Enter acceleration \n" ;

cin>> a ;

cout<<"Enter time \n";

cin>> t ;

v = u + a\*t ;

cout<<v ; ;

return 0;

}

Output:

Enter initial velocity

2

Enter acceleration

4

Enter time

5

OUTPUT22

9.Write a program to swap two characters of different data types using function overloading concept.

#include<iostream>

using namespace std;

void swap(int ,float );

int main()

{

int a;

float b;

cout<<"Enter the Two Numbers to Swap in C++: ";

cin>>a>>b;

cout<<"\nAfter Swapping of Two Numbers:";

swap(a,b);

return 0;

}

void swap(int x,float y)

{

int z;

z=x;

x=y;

y=z;

cout<<" "<<x<<" "<<y;

}

Output:

Enter the Two Numbers to Swap in C++: 2 3

After Swapping of Two Numbers: 3 2

10.Write a program to change the sign of an operands using unary operator overloading concept.

#include <iostream>

using namespace std;

class Count {

private:

int value;

public:

// Constructor to initialize count to 5

Count() : value(5) {}

// Overload ++ when used as prefix

void operator ++ () {

++value;

}

void display() {

cout << "Count: " << value << endl;

}

};

int main() {

Count count1;

// Call the "void operator ++ ()" function

++count1;

count1.display();

return 0;

}

Output:

Count: 6

11.Write a program to add two complex numbers using binary operator overloading concept

#include <iostream>

using namespace std;

class Complex

{

private:

float real;

float imag;

public:

Complex(): real(0), imag(0){ }

void input()

{

cout << "Enter real and imaginary parts respectively: ";

cin >> real;

cin >> imag;

}

Complex operator + (Complex c2)

{

Complex temp;

temp.real = real + c2.real;

temp.imag = imag + c2.imag;

return temp;

}

void output()

{

if(imag < 0)

cout << "Output Complex number: "<< real << imag << "i";

else

cout << "Output Complex number: " << real << "+" << imag << "i";

}

};

int main()

{

Complex c1, c2, result;

cout<<"Enter first complex number:\n";

c1.input();

cout<<"Enter second complex number:\n";

c2.input();

result = c1 + c2;

result.output();

return 0;

}

Output:

Enter first complex number:

Enter real and imaginary parts respectively: 2

3

Enter second complex number:

Enter real and imaginary parts respectively: 4

5

Output Complex number: 6+8i

12.Write a program to find mean value of two integers using friend function concept.

#include<iostream>

using namespace std;

class base

{

int val1,val2;

public:

void get()

{

cout<<"\nEnter 1st value :: ";

cin>>val1;

cout<<"\nEnter 2nd value :: ";

cin>>val2;

}

friend float mean(base ob);

};

float mean(base ob)

{

return float(ob.val1+ob.val2)/2;

}

int main()

{

base obj;

obj.get();

cout<<"\nMean value is :: "<<mean(obj)<<"\n";

return 0;

}

Output:

Enter 1st value :: 23

Enter 2nd value :: 45

Mean value is :: 34

13.Write a program to enter the sale value and print the agent's commission using single inheritance.

#include<iostream>

using namespace std;

class Agent

{

protected:

float commissionRate;

public:

Agent(float rate):commissionRate(rate){}

float calculateCommission(float saleValue){

return commissionRate\*saleValue;

}

};

class Salesperson:public Agent {

public:

Salesperson(float rate):Agent(rate){}

void printCommission(float saleValue){

float commission=calculateCommission(saleValue);

cout<<"Agent's commission for sale value $ "<<saleValue<<"is $"<<commission<<endl;

}

};

int main()

{

float commissionRate;

cout<<"Enter the commission rate for the salesperson(in decimal):";

cin>>commissionRate;

Salesperson salesAgent(commissionRate);

float saleValue;

cout<<"Enter the sale value:$";

cin>>saleValue;

salesAgent.printCommission(saleValue);

return 0;

}

Output:

Enter the commission rate for the salesperson(in decimal):12

Enter the sale value:$1000

Agent's commission for sale value $ 1000is $12000

14.Write a program to calculate division of two number with a try block to detect and 15. throw an exception if the condition" divide -by-zero" occurs.

#include<iostream>

using namespace std;

int main()

{

int var1,var2;

float var3;

cout<<"enter the dividend:";

cin>>var1;

cout<<"\n";

cout<<"enter the divisor:";

cin>>var2;

cout<<"\n";

//exception handling begins here

try //try block

{

if(var2!=0) //checking if divisor is zero

{

var3=var1/var2;

cout<<"outcome :"<<var3;

}

else

{

throw(var2); //throwing the exception found

}

}

//catch block

catch(int exc)

{

cout<<"division by zero is not possible. Please try again with different value of variables";

}

}

Output:enter the dividend:12

enter the divisor:0

division by zero is not possible. Please try again with different value of variables

15.Write a CPP program to find the maximum of 2 numbers using the template.

#include <iostream>

using namespace std;

template <typename T>

T myMax(T x, T y)

{

return (x > y) ? x : y;

}

int main()

{

cout << myMax<int>(3, 7) << endl;

cout << myMax<double>(3.0, 7.0) << endl;

cout << myMax<char>('g', 'e') << endl;

return 0;

}