**1.PROGRAM OBJECTIVE:** An electricity board charges the following rates to domestic users to discourage large consumption of energy.

• For the first 100 units: - 60 P per unit

• For the next 200 units: - 80 P per unit

• Beyond 300 units: - 90 P per unit

All users are charged a minimum of Rs 50. If the total amount is more than Rs 300 then an additional surcharge of 15% is added. Implement a C++ program to read the names of users and number of units consumed and display the charges with names.

**PROGRAM CODE:**

#include <iostream>

using namespace std;

int main() {

string name;

int units;

float charges = 0;

cout << "Enter the user's name: ";

getline(cin, name);

cout << "Enter the number of units consumed: ";

cin >> units;

if (units <= 100) {

charges = units \* 0.60;}

else if (units <= 300) {

charges = (100 \* 0.60) + (units - 100) \* 0.80;}

else { charges = (100 \* 0.60) + (200 \* 0.80) + (units - 300) \* 0.90;}

if (charges < 50) {

charges = 50;}

if (charges > 300) {

charges += charges \* 0.15;

}

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

cout << "Total charges: Rs " << charges << endl;

return 0;

}

**OUTPUT:**

Enter the user's name: John

Enter the number of units consumed: 350

Name: John

Total charges: Rs 265

**2.PROGRAM OBJECTIVE:** Construct a C++ program that removes a specific character from a given string and return the updated string.

Typical Input: computer science is the future

Typical Output: compuer science is he fuure

**PROGRAM CODE:**

#include <iostream>

using namespace std;

string removeChar(string str, char ch) {

string result = "";

for (int i = 0; i < str.length(); i++) {

if (str[i] != ch) {

result += str[i];}}

return result;}

int main() {

string input;

char charToRemove;

cout << "Enter a string: ";

getline(cin, input);

cout << "Enter the character to remove: ";

cin >> charToRemove;

string updatedString = removeChar(input, charToRemove);

cout << "Updated string: " << updatedString << endl;

return 0; }

**OUTPUT:**

Enter a string: computer science is the future

Enter the character to remove: t

Updated string: compuer science is he fuure

**3.PROGRAM OBJECTIVE:** Implement a C++ program to find the non-repeating characters in string.

Typical Input: graphic era university

Typical Output: c g h n p s t u v y

**PROGRAM CODE:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string input;

int count[256] = {0};

cout << "Enter a string: ";

getline(cin, input);

for (char ch : input) {

if (ch != ' ') {

count[ch]++; }}

cout << "Non-repeating characters: ";

for (char ch : input) {

if (ch != ' ' && count[ch] == 1) {

cout << ch << " "; }}

cout << endl;

return 0; }

**OUTPUT:**

Enter a string: graphic era university

Non-repeating characters: g h p c n s t u v y

**4.PROGRAM OBJECTIVE:** Implement a C++ program to demonstrate the concept of data abstraction using the concept of Class and Objects.

**PROGRAM CODE:**

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

private:

string name;

int accountNumber;

double balance;

public:

BankAccount(string accName, int accNum, double initialBalance) {

name = accName;

accountNumber = accNum;

balance = initialBalance;

}

void deposit(double amount) {

if (amount > 0) {

balance += amount;

cout << "Deposited: $" << amount << endl;

} else {

cout << "Invalid deposit amount." << endl;

}

}

void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

cout << "Withdrew: $" << amount << endl;

} else {

cout << "Invalid withdrawal amount." << endl;

}

}

void showDetails() {

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

cout << "Account Number: " << accountNumber << endl;

cout << "Balance: $" << balance << endl; }}:

int main() {

BankAccount myAccount("Alice", 123456, 1000.0);

myAccount.showDetails();

myAccount.deposit(500);

myAccount.withdraw(200);

myAccount.withdraw(1500);

myAccount.showDetails();

return 0;

}

**OUTPUT:**

Name: Alice

Account Number: 123456

Balance: $1000

Deposited: $500

Withdrew: $200

Invalid withdrawal amount.

Name: Alice

Account Number: 123456

Balance: $1300

**5.PROGRAM OBJECTIVE:** Define a class Hotel in C++ with the following specifications: **Private members:**

* Rno : Data member to store room number
* Name : Data member to store customer name
* Tariff : Data member to store per day charges
* NOD : Data member to store number of days of stay
* CALC() : Function to calculate and return amount as NOD\*Tariff ,and if the value of days\* Tariff >10000, then total amount is 1.05\* days\*Tariff.

**Public members:**

* Checkin() Function to enter the content Rno, Name, Tariff and NOD
* Checkout() Function to display Rno, Name, Tariff, NOD and Amount (amount to be displayed by calling function) CALC()

**PROGRAM CODE:**

#include <iostream>

#include <string>

using namespace std;

class Hotel {

private:

int roomNumber;

string customerName;

double tariff;

int numberOfDays;

double calculateAmount() {

double amount = numberOfDays \* tariff;

if (amount > 10000) {

amount \*= 1.05; }

return amount; }

public:

void checkIn() {

cout << "Enter Room Number: ";

cin >> roomNumber;

cin.ignore();

cout << "Enter Customer Name: ";

getline(cin, customerName);

cout << "Enter Tariff per day: ";

cin >> tariff;

cout << "Enter Number of Days of Stay: ";

cin >> numberOfDays; }

void checkOut() {

double amount = calculateAmount();

cout << "\nCheckout Details:\n";

cout << "Room Number: " << roomNumber << endl;

cout << "Customer Name: " << customerName << endl;

cout << "Tariff per day: $" << tariff << endl;

cout << "Number of Days of Stay: " << numberOfDays << endl;

cout << "Total Amount: $" << amount << endl; }};

int main() {

Hotel hotel;

hotel.checkIn();

hotel.checkOut();

return 0; }

**OUTPUT:** Enter Room Number: 101

Enter Customer Name: Alice

Enter Tariff per day: 500

Enter Number of Days of Stay: 22

Checkout Details:

Room Number: 101

Customer Name: Alice

Tariff per day: $500

Number of Days of Stay: 22

Total Amount: $11000

**6.PROGRAM OBJECTIVE:** You are given an array of elements. Now you need to choose the best index of this array. An index of the array is called best if the special sum of this index is maximum across the special sum of all the other indices. To calculate the special sum for any index you pick the first element that is and add it to your sum. Now you pick next two elements i.e., and and add both of them to your sum. Now you will pick the next elements, and this continues till the index for which it is possible to pick the elements. Find the best index and in the output print its corresponding special sum. Note that there may be more than one best index, but you need to only print the maximum special sum. Input First line contains an integer as input. Next line contains space separated integers denoting the elements of the array

Output In the output you have to print an integer that denotes the maximum special sum

Input/Output Format

Typical Input Expected Output

5

1 3 1 2 5 8

10

2 1 3 9 2 4 -10 -9 1 3 9

**PROGRAM CODE:**

#include <iostream>

using namespace std;

int maxSpecialSum(int arr[], int n) {

int maxSum = -100000;

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

int currentSum = 0;

int count = 1;

for (int idx = i; idx < n; ) {

for (int j = 0; j < count && idx < n; ++j) {

currentSum += arr[idx++]; }

count++; }

if (currentSum > maxSum) {

maxSum = currentSum; }}

return maxSum; }

int main() {

int n;

cout << "Enter the size of the array: ";

cin >> n;

int arr[100];

cout << "Enter the elements of the array: ";

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

cin >> arr[i]; }

cout << "The maximum special sum is: " << maxSpecialSum(arr, n) << endl;

return 0; }

**OUTPUT:**

Enter the size of the array: 5

Enter the elements of the array: 1 3 1 2 5

The maximum special sum is: 10

**7.PROGRAM OBJECTIVE:** Implement a Program in C++ by defining a class to represent a bank account. Include the following:

**Data Member:** Name of the depositor, Account number, Type of account (Saving, Current etc.), Balance amount in the account.

**Member Functions:** To assign initial values, To deposit an amount, To withdraw an amount after checking the balance, To display name and balance.

**PROGRAM CODE:**

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

private:

string depositorName;

int accountNumber;

string accountType;

double balance;

public:

void initializeAccount(string name, int accNum, string accType, double initialBalance) {

depositorName = name;

accountNumber = accNum;

accountType = accType;

balance = initialBalance;

}

void deposit(double amount) {

if (amount > 0) {

balance += amount;

cout << "Amount deposited successfully. New balance: $" << balance << endl;

} else {

cout << "Invalid deposit amount!" << endl;

}

}

void withdraw(double amount) {

if (amount > 0 && amount <= balance) {

balance -= amount;

cout << "Amount withdrawn successfully. New balance: $" << balance << endl;

} else if (amount > balance) {

cout << "Insufficient balance!" << endl;

} else {

cout << "Invalid withdrawal amount!" << endl;

}

}

void displayDetails() const {

cout << "Depositor Name: " << depositorName << endl;

cout << "Account Balance: $" << balance << endl;

}

};

int main() {

BankAccount account;

account.initializeAccount("John Doe", 123456, "Saving", 500.0);

account.displayDetails();

account.deposit(200.0);

account.withdraw(100.0);

account.displayDetails();

return 0;

}

**OUTPUT:**

Depositor Name: John Doe

Account Balance: $500

Amount deposited successfully. New balance: $700

Amount withdrawn successfully. New balance: $600

Depositor Name: John Doe

Account Balance: $600

**8.PROGRAM OBJECTIVE:**

Anna is a contender for valedictorian of her high school. She wants to know how many students (if any) have scored higher than her in the exams given during this semester. Create a class named Student with the following specifications:

* An instance variable named scores holds a student's 5 exam scores.
* A void input () function reads 5 integers and saves them to scores.
* An int calculateTotalScore() function that returns the sum of the student's scores.

**PROGRAM CODE:**

#include <iostream>

#include <vector>

using namespace std;

class Student {

private:

vector<int> scores;

public:

void input() {

scores.resize(5); // Ensure there is space for 5 scores

cout << "Enter 5 exam scores: ";

for (int i = 0; i < 5; i++) {

cin >> scores[i]; }}

int calculateTotalScore() const {

int total = 0;

for (int score : scores) {

total += score; }

return total; }};

int main() {

int n;

cout << "Enter the number of students: ";

cin >> n;

vector<Student> students(n);

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

cout << "Student " << i + 1 << ":\n";

students[i].input(); }

int annaScore = students[0].calculateTotalScore();

int higherScores = 0;

for (int i = 1; i < n; i++) {

if (students[i].calculateTotalScore() > annaScore) {

higherScores++; }}

cout << "Number of students who scored higher than Anna: " << higherScores << endl;

return 0; }

**OUTPUT:**

Enter number of students : 3

Enter Marks in different Subjects :

Student 1 : 30 40 45 10 10

Student 2 : 40 40 40 10 10

Student 3 : 50 20 30 10 10

Number of students having better grades than Anna : 1

**9.PROGRAM OBJECTIVE:**

Construct a Program in C++ to show the working of function overloading(compile time polymorphism) by using a function named calculate Area () to calculate area of square, rectangle and triangle using different signatures as required.

**PROGRAM CODE:**

#include <iostream>

using namespace std;

class AreaCalculator {

public:

double calculateArea(double side) {

return side \* side;

}

double calculateArea(double length, double width) {

return length \* width;

}

double calculateArea(double base, double height, char triangle) {

return 0.5 \* base \* height; }};

int main() {

AreaCalculator calculator;

double squareSide = 5.0;

cout << "Area of square: " << calculator.calculateArea(squareSide) << endl;

double rectLength = 7.0, rectWidth = 4.0;

cout << "Area of rectangle: " << calculator.calculateArea(rectLength, rectWidth) << endl;

double triBase = 6.0, triHeight = 8.0;

cout << "Area of triangle: " << calculator.calculateArea(triBase, triHeight, 'T') << endl;

return 0;

}

**10.PROGRAM OBJECTIVE**: Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance.

**Data Members:**

• partNumber (type String)

• partDescription (type String)

• quantity of the item being purchased (type int)

• price\_per\_item (type double)

Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount() that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named invoiceTest that demonstrates class Invoice’s capabilities.

**PROGRAM CODE:**

#include <iostream>

#include <string>

using namespace std;

class Invoice {

private:

string partNumber;

string partDescription;

int quantity;

double price\_per\_item;

public:

Invoice(string partNum, string partDesc, int qty, double price) {

partNumber = partNum;

partDescription = partDesc;

quantity = (qty > 0) ? qty : 0;

price\_per\_item = (price > 0.0) ? price : 0.0;

}

double getInvoiceAmount() const {

return quantity \* price\_per\_item;

}

void display() const {

cout << "Part Number: " << partNumber << "\n";

cout << "Part Description: " << partDescription << "\n";

cout << "Quantity: " << quantity << "\n";

cout << "Price per Item: $" << price\_per\_item << "\n";

cout << "Total Amount: $" << getInvoiceAmount() << "\n\n";

}

};

int main() {

Invoice item1("A101", "Hammer", 10, 15.50);

Invoice item2("B202", "Screwdriver", -5, 7.25);

Invoice item3("C303", "Wrench", 8, -3.75);

cout << "Invoice Details:\n";

item1.display();

item2.display();

item3.display();

return 0;

}

**11.PROGRAM OBJECTIVE**: Imagine a tollbooth with a class called TollBooth. The two data items are of type unsigned int and double to hold the total number of cars and total amount of money collected. A constructor initializes both of these data members to 0. A member function called payingCar( )increments the car total and adds 0.5 to the cash total. Another function called nonPayCar( ) increments the car total but adds nothing to the cash total. Finally a member function called display( )shows the two totals. Include a program to test this class. This program should allow the user to push one key to count a paying car and another to count a non paying car. Pushing the ESC key should cause the program to print out the total number of cars and total cash and then exit.

**PROGRAM CODE:**

#include <iostream>

#include <conio.h>

using namespace std;

class TollBooth {

private:

unsigned int totalCars;

double totalCash;

public:

TollBooth() {

totalCars = 0;

totalCash = 0.0;

}

void payingCar() {

totalCars++;

totalCash += 0.5;

}

void nonPayCar() {

totalCars++;

}

void display() const {

cout << "Total number of cars: " << totalCars << endl;

cout << "Total cash collected: $" << totalCash << endl;

}

};

int main() {

TollBooth tollbooth;

char key;

cout << "Press 'P' for paying car, 'N' for non-paying car, and ESC to exit and display totals." << endl;

while (true) {

key = \_getch();

if (key == 27) {

tollbooth.display();

break;

}

else if (key == 'P' || key == 'p') {

tollbooth.payingCar();

}

else if (key == 'N' || key == 'n') {

tollbooth.nonPayCar();

}

}

return 0;

}

**12.PROGRAM OBJECTIVE**: Create a class called Time that has separate int member data for hours, minutes and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. A member function should display it in 11:59:59 format. A member function named add() should add two objects of type time passed as arguments. A main ( ) program should create two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable.

**PROGRAM CODE:**

#include <iostream>

using namespace std;

class Time {

private: int hours; int minutes; int seconds;

public:

Time() : hours(0), minutes(0), seconds(0) {}

Time(int h, int m, int s) : hours(h), minutes(m), seconds(s) {}

void display() { cout << hours << ":" << minutes << ":" << seconds << endl; }

void add(const Time& t1, const Time& t2) {

seconds = t1.seconds + t2.seconds;

minutes = t1.minutes + t2.minutes + seconds / 60;

hours = t1.hours + t2.hours + minutes / 60;

seconds %= 60;

minutes %= 60;

hours %= 24; }};

int main() {

Time t1(10, 30, 45);

Time t2(5, 50, 20);

Time t3;

t3.add(t1, t2);

t3.display();

return 0; }

**13.PROGRAM OBJECTIVE**: Create class SavingsAccount. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest() to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12.This interest should be added tosavingsBalance. Provide a static method modifyInterestRate() that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of Rs2000.00 and Rs3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month’s interest and print the new balances for both savers.

**PROGRAM CODE:**

#include <iostream>

using namespace std;

class SavingsAccount {

private:

static double annualInterestRate;

double savingsBalance;

public:

SavingsAccount(double balance) {

savingsBalance = balance;

}

void calculateMonthlyInterest() {

savingsBalance += (savingsBalance \* annualInterestRate / 12);

}

static void modifyInterestRate(double newRate) {

annualInterestRate = newRate;

}

void displayBalance() const {

cout << "Balance: Rs " << savingsBalance << endl;

}

};

double SavingsAccount::annualInterestRate = 0.04;

int main() {

SavingsAccount saver1(2000.00);

SavingsAccount saver2(3000.00);

SavingsAccount::modifyInterestRate(0.04);

saver1.calculateMonthlyInterest();

saver2.calculateMonthlyInterest();

saver1.displayBalance();

saver2.displayBalance();

SavingsAccount::modifyInterestRate(0.05);

saver1.calculateMonthlyInterest();

saver2.calculateMonthlyInterest();

saver1.displayBalance();

saver2.displayBalance();

return 0;

}

**14.PROGRAM OBJECTIVE**: Create a class Complex having two int type variable named real & img denoting real and imaginary part respectively of a complex number. Overload +, - , == operator to add, to subtract and to compare two complex numbers being denoted by the two complex type objects.

**PROGRAM CODE:**

#include <iostream>

using namespace std;

class Complex {

private:

int real; int img;

public:

Complex(int r = 0, int i = 0) : real(r), img(i) {}

Complex operator+(const Complex& other) {

return Complex(real + other.real, img + other.img); }

Complex operator-(const Complex& other) {

return Complex(real - other.real, img - other.img); }

bool operator==(const Complex& other) {

return (real == other.real && img == other.img); }

void display() const { cout << real << " + " << img << "i" << endl; }};

int main() { Complex c1(3, 4), c2(1, 2), c3(3, 4), result;result = c1 + c2;

cout << "Addition: ";

result.display();

result = c1 - c2;

cout << "Subtraction: ";result.display();

if (c1 == c2)

cout << "c1 and c2 are equal." << endl;

else

cout << "c1 and c2 are not equal." << endl;

if (c1 == c3)

cout << "c1 and c3 are equal." << endl;

else

cout << "c1 and c3 are not equal." << endl;

return 0;

}

**OUTPUT:**

Addition: 4 + 6i

Subtraction: 2 + 2i

c1 and c2 are not equal.

c1 and c3 are equal.

**15.PROGRAM OBJECTIVE**: Using the concept of operator overloading. Implement a program to overload the following:

**a.** Unary –

**b.** Unary ++ preincrement, postincrement

**c.** Unary -- predecrement, postdecrement

**PROGRAM CODE:**

#include <iostream>

using namespace std;

class Counter {

private:

int value;

public:

Counter(int v = 0) : value(v) {}

Counter operator-() {

return Counter(-value); }

Counter& operator++() {

++value;

return \*this; }

Counter operator++(int) {

Counter temp = \*this;

++value;

return temp; }

Counter& operator--() {

--value;

return \*this; }

Counter operator--(int) {

Counter temp = \*this;

--value;

return temp; }

void display() const {

cout << "Value: " << value << endl; }};

int main() {

Counter c1(5), c2(10);

cout << "Original value of c1: ";

c1.display();

cout << "Original value of c2: ";

c2.display();

cout << "\nUnary - (negation) on c1: ";

(-c1).display();

cout << "\nPre-increment on c1: ";

(++c1).display();

cout << "\nPost-increment on c2: ";

(c2++).display();

c2.display();

cout << "\nPre-decrement on c1: ";

(--c1).display();

cout << "\nPost-decrement on c2: ";

(c2--).display();

c2.display();

return 0;

}

**OUTPUT:**

Original value of c1: Value: 5

Original value of c2: Value: 10

Unary - (negation) on c1: Value: -5

Pre-increment on c1: Value: 6

Post-increment on c2: Value: 10

Value: 11

Pre-decrement on c1: Value: 5

Post-decrement on c2: Value: 11

Value: 10