**OBJECT ORIENTED PROGRAMMING**

# PROGRAM 1:

Write a user-defined program to declare a class which stores a complex number. Demonstrate the use of constant objects, constant member function and constant arguments, using this class

# CODE:

//Hassan tariq

//2023-BSAI-004

#include <iostream>

using namespace std;

class hassan{

private:

double real;

double image;

public:

// Constructor

hassan(double r = 0.0, double i = 0.0)

{

real = r;

image = i;

}

// Constant member function to display complex number

void display() const

{

cout << real << " + " << image << "i" << endl;

}

// Constant member function to add two complex numbers

hassan add(const hassan& other) const

{

return ayesha(real + other.real, image + other.image);

}

};

int main() {

// Constant object declaration

const hassan h1(2.0, 3.0);

const hassan h2(1.0, 4.0);

// Display constant objects

cout << "Constant Object c1: ";

h1.display();

cout << "Constant Object c2: ";

h2.display();

// Addition of constant objects

const hassan sum = h1.add(h2);

cout << "Sum of c1 and c2: ";

sum.display();

return 0;

}

# PROGRAM 2:

Write a class that contain the following attribute • The name of car • Direction of car (E, W, N, S) • The position of car (from imaginary zero point) The class has fallowing member function The constructor to be initialize • Turn function use to change the direction of car to one steps right side (e.g. if the direction is E, Should be change to S and so on) • Overload the turn function to change the direction to any side directly. It should accept the direction parameter. • Move function to change the position of car away from zero point. It should accept the distance as parameter.

# CODE:

//HASSAN TARIQ

//2023-bsai-004

#include <iostream>

using namespace std;

class Car {

private:

string name;

char direction;

int pX;

int pY;

public:

Car(string carName, char carDirection, int carPX, int carPY)

{

name = carName;

direction = carDirection;

pX = carPX;

pY = carPY;

}

void turn()

{

switch(direction)

{

case 'N':

direction = 'E';

break;

case 'E':

direction = 'S';

break;

case 'S':

direction = 'W';

break;

case 'W':

direction = 'N';

break;

}

}

void turn(char newDirection)

{

direction = newDirection;

}

void move(int distance)

{

switch(direction)

{

case 'N':

pY += distance;

break;

case 'E':

pX += distance;

break;

case 'S':

pY -= distance;

break;

case 'W':

pX -= distance;

break;

}

}

void display()

{

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

cout << "Direction: " << direction << endl;

cout << "Position: (" << pX << ", " << pY << ")" << endl;

}

};

int main()

{

Car hassan("Toyota", 'N', 0, 0);

hassan.display();

hassan.turn();

hassan.display();

hassan.turn('W');

hassan.display();

hassan.move(5);

hassan.display();

return 0;

}

# PROGRAM 3:

Write a function find(…) that accepts a one-dimensional integer array of size 10 as an argument to the function. Your program then finds the location and value of the largest and second-largest elements in a one-dimensional array. Display answers in main().

# CODE:

//hassan tariq

//2023-bsai-004

#include <iostream>

using namespace std;

void hassan(int arr[], int size, int &maxVal, int &maxIndex, int &secondMaxVal, int &secondMaxIndex)

{

maxVal = arr[0];

maxIndex = 0;

secondMaxVal = arr[1];

secondMaxIndex = 1;

for (int i = 1; i < size; ++i)

{

if (arr[i] > maxVal)

{

secondMaxVal = maxVal;

secondMaxIndex = maxIndex;

maxVal = arr[i];

maxIndex = i;

} else if (arr[i] > secondMaxVal)

{

secondMaxVal = arr[i];

secondMaxIndex = i;

}

}

}

int main()

{

int arr[10];

int maxVal, maxIndex, secondMaxVal, secondMaxIndex;

cout << "Enter 10 integers:" << endl;

for (int i = 0; i < 10; ++i)

{

cout << "Enter element " << i + 1 << ": ";

cin >> arr[i];

}

hassan(arr, 10, maxVal, maxIndex, secondMaxVal, secondMaxIndex);

cout << "Largest element is " << maxVal << " at index " << maxIndex << endl;

cout << "Second largest element is " << secondMaxVal << " at index " << secondMaxIndex << endl;

return 0;

}

# PROGRAM 4:

Write a function arrange(…) that accepts a one-dimensional integer array of size 10 as an argument to the function. The program then shifts negative numbers to the left and positive numbers to the right side of the array. For example, Array is 3 -5 1 2 7 0 -15 6 -4 -8 Output (After Deletion): -5 -15 -4 -8 3 1 2 7 0 6

# CODE:

//hassan tariq

//2023-bsai-004

#include <iostream>

using namespace std;

void hassan(int arr[])

{

int l = 0, r= 9;

while (l< r)

{

while (arr[l] < 0 && l < r)

{

l++;

}

while (arr[r] >= 0 && l < r)

{

r--;

}

if (l < r)

{

int temp = arr[l];

arr[l] = arr[r];

arr[r] = temp;

l++;

r--;

}

}

}

int main()

{

int arr[10];

cout << "Enter 10 integers: ";

for (int i = 0; i < 10; ++i)

{

cin >> arr[i];

}

cout << "Original array: ";

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}

cout << endl;

hassan(arr);

cout << "Arranged array: ";

for (int i = 0; i < 10; i++) {

cout << arr[i] << " ";

}

cout <<endl;

return 0;

}

# PROGRAM 5:

Create a class employee which stores is name, ID and salary of an employee by user input. The ID should be generated upon the creation of object, starting from 1. Include all the constructors and destructor in the class. Create one object using each of the constructors and display it

# CODE:

//HASSAN TARIQ

//023-bsai-004

#include <iostream>

#include <string>

using namespace std;

class hassan {

private:

string name;

int id;

double salary;

public:

// Default constructor

hassan() {

static int nextId = 1;

id = nextId++;

salary = 10000.0;

}

// Parameterized constructor

hassan(string empName, double empSalary) {

static int nextId = 1;

name = empName;

id = nextId++;

salary = empSalary;

}

// Copy constructor

hassan(const hassan& emp) {

name = emp.name;

id = emp.id;

salary = emp.salary;

}

// Destructor

~hassan() {

cout << "Destructor called for Employee " << id << endl;

}

void display() {

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

cout << "ID: " << id << endl;

cout << "Salary: " << salary <<endl;

}

};

int main() {

hassan emp1;

emp1.display();

string name;

double salary;

cout << "Enter name: ";

cin >> name;

cout << "Enter salary: ";

cin >> salary;

hassan emp2(name, salary);

emp2.display();

hassan emp3 = emp2;

emp3.display();

return 0;

}

# PROGRAM 6:

Write a C++ program for the class vehicle and its drive class water transport, road transport and air transport vehicles. Make suitable data variables and member functions. When you create an object must be count and display total no of object created also create every class objects and access member through the member functions

# CODE:

//hassan tariq

//2023-bsai-004

#include <iostream>

using namespace std;

class Vehicle {

private:

static int count;

public:

Vehicle() {

count++;

}

void displayCount() {

cout << "Total number of vehicles: " << count << endl;

}

};

// Initialize static variable count

int Vehicle::count = 0;

class WaterTransport : public Vehicle {

public:

void displayType() {

cout << "Water Transport" << endl;

}

};

class RoadTransport : public Vehicle {

public:

void displayType() {

cout << "Road Transport" << endl;

}

};

class AirTransport : public Vehicle {

public:

void displayType() {

cout << "Air Transport" << endl;

}

};

int main() {

// Creating objects of each class

WaterTransport waterVehicle;

RoadTransport roadVehicle;

AirTransport airVehicle;

// Displaying type and total count of vehicles

waterVehicle.displayType();

waterVehicle.displayCount();

roadVehicle.displayType();

roadVehicle.displayCount();

airVehicle.displayType();

airVehicle.displayCount();

return 0;

}

# PROGRAM 7:

Implement a C++ class named Employee with the following specifications: • The class should have private data members name (string), id (integer), and salary (floatingpoint). • Implement a static data member totalEmployees to keep track of the total number of employees. • Implement a static member function averageSalary() that calculates and returns the average salary of all employees. • Provide member functions to set and get the values of name, id, and salary. • Implement a constructor to initialize the name, id, and salary of an employee. • Implement a destructor to decrement the totalEmployees count when an object is destroyed.

# CODE:

//hassan tariq

//2023-bsai-004

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

string name;

int id;

float salary;

static int totalEmployees;

static float totalSalary;

public:

Employee(std::string \_name, int \_id, float \_salary) {

name = \_name;

id = \_id;

salary = \_salary;

totalEmployees++;

totalSalary += \_salary;

}

~Employee() {

totalEmployees--;

totalSalary -= salary;

}

static float averageSalary() {

if (totalEmployees == 0)

return 0;

return totalSalary / totalEmployees;

}

void setName(std::string \_name) {

name = \_name;

}

std::string getName() const {

return name;

}

void setId(int \_id) {

id = \_id;

}

int getId() const {

return id;

}

void setSalary(float \_salary) {

totalSalary -= salary;

salary = \_salary;

totalSalary += salary;

}

float getSalary() const {

return salary;

}

};

int Employee::totalEmployees = 0;

float Employee::totalSalary = 0;

int main() {

Employee emp1("hassan tariq", 1, 50000);

Employee emp2("masood ", 2, 60000);

cout << "Average Salary: " << Employee::averageSalary() << endl;

return 0;

}

# PROGRAM 8:

(Car Pool Savings Calculator) Research several car-pooling websites. create an application that calculates your daily driving cost, so that you can estimate how much money could be saved by carpooling, which also has other advantages such as reducing carbon emission and reducing traffic congestion. The application should input the following and display the user's cost per day of driving to word: a) Total miles driven per day. b) Cost per gallon of gasoline. c) Average miles per gallon d) Parking fees per day. e) Toll per day.

# CODE:

//hassan tariq

//2023-bsai-004

#include <iostream>

using namespace std;

//mpd=milesperday

//cpg=costpergallon

//mpg=milespergallon

//pf=parkingfee

double calculateDrivingCost(double mpd, double cpg, double mpg, double pf, double toll, int numPeople) {

double gasCost = (mpd / mpg) \* cpg;

double totalCost = gasCost + pf + toll;

double costPerPerson = totalCost / numPeople;

return costPerPerson;

}

int main() {

double mpd, cpg, mpg, pf, toll;

int numPeople;

cout << "\t\tCar Pool Savings Calculator\n\n";

cout << "Enter total miles driven per day: ";

cin >> mpd;

cout << "Enter cost per gallon of gasoline: ";

cin >> cpg;

cout << "Enter average miles per gallon: ";

cin >> mpg;

cout << "Enter parking fees per day: ";

cin >> pf;

cout << "Enter toll per day: ";

cin >> toll;

cout << "Enter number of people in the carpool (including yourself): ";

cin >> numPeople;

double dailyCostPerPerson = calculateDrivingCost(mpd, cpg, mpg, pf, toll, numPeople);

cout << "\nYour daily driving cost per person: $" << dailyCostPerPerson << "\n";

double totalSavings = (mpd / mpg) \* cpg - dailyCostPerPerson;

if (totalSavings > 0) {

cout << "Money saved by carpooling per day: $" << totalSavings << "\n";

} else {

cout << "Carpooling does not lead to savings compared to driving alone.\n";

}

return 0;

}