**EXP -1 Classes and Objects**

/\*

Problem statement:

    Airline details format and its attributes

    1)Name of the pasenger

    2) Age

    3)Flight no

    4)Departure time

    5)location

    write a c++ code to accept and display the values of the passenger for the airline

Author: Shreerang P Mhatre

Date: 23/08/2023

Input:  Enter the name of the Passenger: Shree

        Enter the age of the Passenger: 20

        Enter the Flight Number: 5674

        Enter the Departure Time: 5

        Enter the current source: pune

Output: Airline Details:

        Name: Shree

        Age: 20

        Flight No: 5674

        Departure Time: 5

        Source: pune

\*/

#include<iostream>

using namespace std;

// Airline class to accept and display data

class airline{

private:

    string name;

    int age;

    int flight;

    int time;

    string source;

public:

    void readDeatails(); //Function to read data

    void displayDetails(); //Function to display data

};

// Accept the input values from the passenger

void airline::readDeatails() {

    cout<<"Enter the name of the Passenger: ";

    cin >> name;

    cout<<"Enter the age of the Passenger: ";

    cin >> age;

    cout<<"Enter the Flight Number: ";

    cin >> flight;

    cout<<"Enter the Departure Time: ";

    cin >> time;

    cout<<"Enter the current source: ";

    cin >> source;

}

//Display the accepted values in organized format

void airline::displayDetails() {

    cout << "Name: " << name << endl;

    cout << "Age: " << age << endl;

    cout << "Flight No: " << flight << endl;

    cout << "Departure Time: " << time << endl;

    cout << "Source: " << source << endl;

}

// Main function

int main() {

    airline air;

    air.readDeatails();

    cout << "Airline Details: "<< endl;

    air.displayDetails();

    return 0;

}

**EXP-2 Database of Airline Company**

/\*

Employee Data Base

Develop an object oriented program in C++ to create a database of employee into system containing following info:

Employee name, Employee No, Qualification ,address ,contact , Salary (basic, DA, Ta , Net Salary)

Construct the Database with suitable inline member function for initializing and destroying the data via

Constructor, default Constructor, Copy Constructor , destructor.

Use dynamic memory allocation concept while creating and destroying the object of class.

Use static data member concept whenever needed display the Employee info.

\*/

#include <iostream>

#include <string>

class Employee {

private:

    static int empCounter;

    std::string empName;

    int empNumber;

    std::string qualification;

    std::string address;

    std::string contactNumber;

    struct Salary {

        double basic;

        double DA;

        double TA;

        double netSalary;

    } salary;

public:

    // Parameterized constructor

    Employee(const std::string& name, int number, const std::string& qual, const std::string& addr,

             const std::string& contact, double basicSalary, double DA, double TA) {

        empName = name;

        empNumber = number;

        qualification = qual;

        address = addr;

        contactNumber = contact;

        salary.basic = basicSalary;

        salary.DA = DA;

        salary.TA = TA;

        salary.netSalary = calculateNetSalary();

        empCounter++;

    }

    // Default constructor

    Employee() : Employee("", 0, "", "", "", 0.0, 0.0, 0.0) {}

    // Copy constructor

    Employee(const Employee& other) {

        empName = other.empName;

        empNumber = other.empNumber;

        qualification = other.qualification;

        address = other.address;

        contactNumber = other.contactNumber;

        salary.basic = other.salary.basic;

        salary.DA = other.salary.DA;

        salary.TA = other.salary.TA;

        salary.netSalary = other.salary.netSalary;

        empCounter++;

    }

    // Destructor

    ~Employee() {

        empCounter--;

    }

    // Calculate the net salary based on basic, DA, and TA

    double calculateNetSalary() const {

        return salary.basic + salary.DA + salary.TA;

    }

    // Display employee information

    void displayInfo() const {

        std::cout << "Employee Name: " << empName << std::endl;

        std::cout << "Employee Number: " << empNumber << std::endl;

        std::cout << "Qualification: " << qualification << std::endl;

        std::cout << "Address: " << address << std::endl;

        std::cout << "Contact Number: " << contactNumber << std::endl;

        std::cout << "Basic Salary: " << salary.basic << std::endl;

        std::cout << "DA: " << salary.DA << std::endl;

        std::cout << "TA: " << salary.TA << std::endl;

        std::cout << "Net Salary: " << salary.netSalary << std::endl;

        std::cout << "--------------------------------" << std::endl;

    }

    // Static member function to get the total number of employees

    static int getTotalEmployees() {

        return empCounter;

    }

};

// Initialize the static member

int Employee::empCounter = 0;

int main() {

    // Create employee objects using dynamic memory allocation

    Employee\* emp1 = new Employee("Shreerang Mhatre", 302, "B.Tech Engineering", "MIT-WPU", "+91 123456789 ", 5000.0, 1000.0, 500.0);

    Employee\* emp2 = new Employee("Aman Singh", 401, "M.Tech Engineering", "BVP PUNE", "+91 325689741", 6000.0, 1200.0, 600.0);

    // Display employee information

    emp1->displayInfo();

    emp2->displayInfo();

    // Get the total number of employees

    std::cout << "Total Employees: " << Employee::getTotalEmployees() << std::endl;

    // Clean up memory and release resources

    delete emp1;

    delete emp2;

    return 0;

}

**EXP-3 Employee Payroll System**

/\*

Problem statement:

   Create an Employee Payroll System in C++ using object-oriented programming.

   Define a base class Employee with attributes and a pay slip calculator.

   Implement derived classes for different employee categories (Programmer, Team Lead,

   Assistant Project Manager, and Project Manager) inheriting from Employee.

   In the main function, instantiate employees of each category, input their details

   and basic pay, and display their pay slips, demonstrating inheritance,

   constructors, and polymorphism.

Author: Shreerang P Mhatre

Date: 13/09/2023

\*/

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

    string emp\_name;

    string emp\_id;

    string address;

    string mail\_id;

    string mobile\_no;

public:

    Employee(string name, string id, string addr, string mail, string mobile)

        : emp\_name(name), emp\_id(id), address(addr), mail\_id(mail), mobile\_no(mobile) {}

    void display\_pay\_slip(double basic\_pay) {

        cout << "Employee: " << emp\_name << endl;

        cout << "Employee ID: " << emp\_id << endl;

        cout << "Address: " << address << endl;

        cout << "Mail ID: " << mail\_id << endl;

        cout << "Mobile Number: " << mobile\_no << endl;

        double da = 0.97 \* basic\_pay;

        double hra = 0.10 \* basic\_pay;

        double pf = 0.12 \* basic\_pay;

        double staff\_club\_fund = 0.001 \* basic\_pay;

        double gross\_salary = basic\_pay + da + hra;

        double net\_salary = gross\_salary - pf - staff\_club\_fund;

        cout << "Basic Pay: " << basic\_pay << endl;

        cout << "Dearness Allowance (DA): " << da << endl;

        cout << "House Rent Allowance (HRA): " << hra << endl;

        cout << "Provident Fund (PF): " << pf << endl;

        cout << "Staff Club Fund: " << staff\_club\_fund << endl;

        cout << "Gross Salary: " << gross\_salary << endl;

        cout << "Net Salary: " << net\_salary << endl;

        cout << "----------------------------------" << endl;

    }

};

class Programmer : public Employee {

public:

    Programmer(string name, string id, string addr, string mail, string mobile, double basic\_pay)

        : Employee(name, id, addr, mail, mobile) {

        display\_pay\_slip(basic\_pay);

    }

};

class TeamLead : public Employee {

public:

    TeamLead(string name, string id, string addr, string mail, string mobile, double basic\_pay)

        : Employee(name, id, addr, mail, mobile) {

        display\_pay\_slip(basic\_pay);

    }

};

class AssistantProjectManager : public Employee {

public:

    AssistantProjectManager(string name, string id, string addr, string mail, string mobile, double basic\_pay)

        : Employee(name, id, addr, mail, mobile) {

        display\_pay\_slip(basic\_pay);

    }

};

class ProjectManager : public Employee {

public:

    ProjectManager(string name, string id, string addr, string mail, string mobile, double basic\_pay)

        : Employee(name, id, addr, mail, mobile) {

        display\_pay\_slip(basic\_pay);

    }

};

int main() {

    Programmer programmer("Viraj Parmar", "EMP123", "123 Main St.", "john@example.com", "555-1234", 50000);

    TeamLead team\_lead("Sagar Acharya", "EMP456", "456 Elm St.", "jane@example.com", "555-5678", 60000);

    AssistantProjectManager assistant\_pm("Sandeep Patil", "EMP789", "789 Oak St.", "michael@example.com", "555-9876", 70000);

    ProjectManager project\_manager("Sourabh Shah", "EMP101", "101 Pine St.", "sarah@example.com", "555-1111", 80000);

    return 0;

}

**EXP-4 Friend Function in C++**

/\*

Shreerang Mhatre

Rollno -52

Batch - A3

Exp -4

Friend Function in C++

In C++ define a class Box consisting of the following

data members length, breadth and height member functions

1 one default constructor

2 Two overloaded operator member function '<<' and '>>'

to display and read box dimensions

3 One member function '+' to add two box objects and one friend function to compute the volume

and the box using operator overloading

\*/

#include <iostream>

class Box {

private:

    double length;

    double breadth;

    double height;

public:

    // Default constructor

    Box() : length(0.0), breadth(0.0), height(0.0) {}

    // Overloaded '<<' operator to display box dimensions

    friend std::ostream& operator<<(std::ostream& os, const Box& box) {

        os << "Length: " << box.length << " Breadth: " << box.breadth << " Height: " << box.height;

        return os;

    }

    // Overloaded '>>' operator to read box dimensions

    friend std::istream& operator>>(std::istream& is, Box& box) {

        std::cout << "Enter length: ";

        is >> box.length;

        std::cout << "Enter breadth: ";

        is >> box.breadth;

        std::cout << "Enter height: ";

        is >> box.height;

        return is;

    }

    // Member function to add two Box objects

    Box operator+(const Box& other) {

        Box result;

        result.length = this->length + other.length;

        result.breadth = this->breadth + other.breadth;

        result.height = this->height + other.height;

        return result;

    }

    // Friend function to compute the volume of the Box

    friend double computeVolume(const Box& box) {

        return box.length \* box.breadth \* box.height;

    }

};

int main() {

    Box box1, box2, result;

    std::cout << "Enter dimensions for Box 1:" << std::endl;

    std::cin >> box1;

    std::cout << "Enter dimensions for Box 2:" << std::endl;

    std::cin >> box2;

    std::cout << "Box 1: " << box1 << std::endl;

    std::cout << "Box 2: " << box2 << std::endl;

    result = box1 + box2;

    std::cout << "Sum of Box 1 and Box 2: " << result << std::endl;

    double volume = computeVolume(result);

    std::cout << "Volume of the sum of Box 1 and Box 2: " << volume << std::endl;

    return 0;

}

**EXP-5 Virtual Function**

/\*

Shreerang Mhatre

Rollno - 52

Batch - A3

Expno - 5

Write a C++ program with base classEmployee and derive classes Class1\_Employee,

Class2\_Employee and Class3\_Employee.

Salary of an employee is calculated as per his/her designation.

Declare calculate salary () as a pure virtual function in the base class and

define it in respective derive classes to calculate salary of an employee.

\*/

#include <iostream>

// Base class

class Employee {

public:

    virtual double calculateSalary() const = 0; // virtual function

    virtual void displayType() const {

        std::cout << "Base Employee" << std::endl;

    }

};

// Derived class 1

class Class1\_Employee : public Employee {

public:

    double calculateSalary() const override {

        // Implement salary calculation logic for Class1\_Employee

        return 50000.0;

    }

    void displayType() const override {

        std::cout << "Class1\_Employee" << std::endl;

    }

};

// Derived class 2

class Class2\_Employee : public Employee {

public:

    double calculateSalary() const override {

        // Implement salary calculation logic for Class2\_Employee

        return 60000.0;

    }

    void displayType() const override {

        std::cout << "Class2\_Employee" << std::endl;

    }

};

// Derived class 3

class Class3\_Employee : public Employee {

public:

    double calculateSalary() const override {

        // Implement salary calculation logic for Class3\_Employee

        return 70000.0;

    }

    void displayType() const override {

        std::cout << "Class3\_Employee" << std::endl;

    }

};

int main() {

    Class1\_Employee employee1;

    Class2\_Employee employee2;

    Class3\_Employee employee3;

    // Displaying employee types and their salaries

    employee1.displayType();

    std::cout << "Salary: $" << employee1.calculateSalary() << std::endl;

    employee2.displayType();

    std::cout << "Salary: $" << employee2.calculateSalary() << std::endl;

    employee3.displayType();

    std::cout << "Salary: $" << employee3.calculateSalary() << std::endl;

    return 0;

}

**EXP-6 File and exception Handling**

/\*

Name: Shreerang Mhatre

Rollno: 52

Batch: A3

Exp: 6 - file and exception handling

Problrm statement:

A School maintains the mark sheets of all standard students in the following format:

PRN

Student Name

Maths

Physics

Chemistry

Total %

Grade

A techer put marks for the student by his/her PRN and the system checks whether marks

for different subjects are negative or not. If it is negative, the  system displays appropriate

messagr otherwise updates the files by storing the marks across the subjects. The system calculates

the total percentage after putting marks for all three subjectsand accordinglyfinds the grade.

Whenever an  administrartor wants to search a studenets record , he/she inputs student PRN and

the system searches the file and displays wheather theit is available or not, otherwise  an

appropriate message is displayed. An administrator can also delete/modify a record of a student.

Design such system using c++ Program with file and exception handling.

\*/

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class Student {

public:

    int prn;

    string name;

    float maths;

    float physics;

    float chemistry;

    float totalPercentage;

    char grade;

    // Member functions

    void calculatePercentageAndGrade() {

        totalPercentage = (maths + physics + chemistry) / 3.0;

        if (totalPercentage >= 90) {

            grade = 'A';

        } else if (totalPercentage >= 80) {

            grade = 'B';

        } else if (totalPercentage >= 70) {

            grade = 'C';

        } else if (totalPercentage >= 60) {

            grade = 'D';

        } else {

            grade = 'F';

        }

    }

};

void addStudentRecord() {

    ofstream outfile("students.txt", ios::app);

    if (!outfile.is\_open()) {

        cerr << "Error opening file for writing!" << endl;

        return;

    }

    Student student;

    cout << "Enter PRN: ";

    cin >> student.prn;

    // Check if PRN already exists

    ifstream infile("students.txt");

    Student tempStudent;

    bool prnExists = false;

    while (infile >> tempStudent.prn >> tempStudent.name >> tempStudent.maths >> tempStudent.physics

    >> tempStudent.chemistry >> tempStudent.totalPercentage >> tempStudent.grade) {

        if (tempStudent.prn == student.prn) {

            prnExists = true;

            break;

        }

    }

    infile.close();

    if (prnExists) {

        cout << "PRN already exists. Please use modify option to update the record." << endl;

        return;

    }

    cout << "Enter Student Name: ";

    cin.ignore();

    getline(cin, student.name);

    cout << "Enter marks for Maths: ";

    cin >> student.maths;

    if (student.maths < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        return;

    }

    cout << "Enter marks for Physics: ";

    cin >> student.physics;

    if (student.physics < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        return;

    }

    cout << "Enter marks for Chemistry: ";

    cin >> student.chemistry;

    if (student.chemistry < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        return;

    }

    // Calculate total percentage and grade

    student.calculatePercentageAndGrade();

    // Write to file

    outfile << student.prn << " " << student.name << " " << student.maths << " " << student.physics

    << " " << student.chemistry << " " << student.totalPercentage << " " << student.grade << endl;

    outfile.close();

    cout << "Record added successfully!" << endl;

}

void searchStudentRecord() {

    ifstream infile("students.txt");

    if (!infile.is\_open()) {

        cerr << "Error opening file for reading!" << endl;

        return;

    }

    int searchPRN;

    cout << "Enter PRN to search: ";

    cin >> searchPRN;

    Student student;

    bool found = false;

    while (infile >> student.prn >> student.name >> student.maths >> student.physics >>

    student.chemistry >> student.totalPercentage >> student.grade) {

        if (student.prn == searchPRN) {

            found = true;

            break;

        }

    }

    infile.close();

    if (found) {

        cout << "Record found:" << endl;

        cout << "PRN: " << student.prn << endl;

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

        cout << "Maths: " << student.maths << endl;

        cout << "Physics: " << student.physics << endl;

        cout << "Chemistry: " << student.chemistry << endl;

        cout << "Total Percentage: " << student.totalPercentage << "%" << endl;

        cout << "Grade: " << student.grade << endl;

    } else {

        cout << "Record not found." << endl;

    }

}

void modifyStudentRecord() {

    ifstream infile("students.txt");

    ofstream outfile("temp.txt");

    if (!infile.is\_open() || !outfile.is\_open()) {

        cerr << "Error opening file for reading or writing!" << endl;

        return;

    }

    int modifyPRN;

    cout << "Enter PRN to modify: ";

    cin >> modifyPRN;

    Student student;

    bool found = false;

    while (infile >> student.prn >> student.name >> student.maths >> student.physics >>

    student.chemistry >> student.totalPercentage >> student.grade) {

        if (student.prn == modifyPRN) {

            found = true;

            break;

        }

        outfile << student.prn << " " << student.name << " " << student.maths << " "

        << student.physics << " " << student.chemistry << " " << student.totalPercentage

        << " " << student.grade << endl;

    }

    if (!found) {

        cout << "Record not found." << endl;

        infile.close();

        outfile.close();

        return;

    }

    cout << "Enter new marks for Maths: ";

    cin >> student.maths;

    if (student.maths < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        infile.close();

        outfile.close();

        return;

    }

    cout << "Enter new marks for Physics: ";

    cin >> student.physics;

    if (student.physics < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        infile.close();

        outfile.close();

        return;

    }

    cout << "Enter new marks for Chemistry: ";

    cin >> student.chemistry;

    if (student.chemistry < 0) {

        cerr << "Error: Marks cannot be negative!" << endl;

        infile.close();

        outfile.close();

        return;

    }

    // Calculate total percentage and grade

    student.calculatePercentageAndGrade();

    // Write modified record to file

    outfile << student.prn << " " << student.name << " " << student.maths << " " << student.physics

    << " " << student.chemistry << " " << student.totalPercentage << " " << student.grade << endl;

    // Copy the rest of the records

    while (infile >> student.prn >> student.name >> student.maths >> student.physics >>

    student.chemistry >> student.totalPercentage >> student.grade) {

        outfile << student.prn << " " << student.name << " " << student.maths << " "

        << student.physics << " " << student.chemistry << " " << student.totalPercentage << " "

        << student.grade << endl;

    }

    infile.close();

    outfile.close();

    // Rename temp file to original file

    remove("students.txt");

    rename("temp.txt", "students.txt");

    cout << "Record modified successfully!" << endl;

}

int main() {

    int choice;

    do {

        cout << "\n\*\*\*\*\* Student Record System \*\*\*\*\*" << endl;

        cout << "1. Add Student Record" << endl;

        cout << "2. Search Student Record" << endl;

        cout << "3. Modify Student Record" << endl;

        cout << "4. Quit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                addStudentRecord();

                break;

            case 2:

                searchStudentRecord();

                break;

            case 3:

                modifyStudentRecord();

                break;

            case 4:

                cout << "Exiting program. Goodbye!" << endl;

                break;

            default:

                cout << "Invalid choice. Please enter a valid option." << endl;

        }

    } while (choice != 4);

    return 0;

**EXP-7 Bubble sort Algorithm**

/\*

Problem statement:

   Perform bubble sort operation using the template for integer and floating data types

Author: Shreerang P Mhatre

Date: 27/09/2023

\*/

#include <iostream>

#include <vector>

template <typename T>

void bubbleSort(std::vector<T> &arr) {

    int n = arr.size();

    bool swapped;

    do {

        swapped = false;

        for (int i = 0; i < n - 1; ++i) {

            if (arr[i] > arr[i + 1]) {

                std::swap(arr[i], arr[i + 1]);

                swapped = true;

            }

        }

    } while (swapped);

}

int main() {

    // Sorting integers

    std::vector<int> intArr = {7,10,888,2,3};

    std::cout << "Original integer array: ";

    for (const int &num : intArr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    bubbleSort(intArr);

    std::cout << "Sorted integer array: ";

    for (const int &num : intArr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    // Sorting floats

    std::vector<float> floatArr = {3.14, 1.23, 2.71, 0.99, 4.56};

    std::cout << "Original float array: ";

    for (const float &num : floatArr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    bubbleSort(floatArr);

    std::cout << "Sorted float array: ";

    for (const float &num : floatArr) {

        std::cout << num << " ";

    }

    std::cout << std::endl;

    return 0;

}

**EXP-8 List and Arrays**

/\*

Shreerang Mhatre

Rollno - 52

Batch - A3

Exp - 8

Write a program in C++  to manage a shopping list. Each shopping list item is

represented by a string stored in a container. Your design requires

a print function that prints out the contents of the shopping list.

Create an empty list.

Append the items, "eggs," "milk," "sugar","chocolate," and

"flour" to the list. Print the list.

Remove the first element from the list. Print the list.

Insert the item, "coffee" at the beginning of the list. Print the list.

Find the item, "sugar" and replace it with "honey." Print the list.

Insert the item, "baking powder" before "milk" in the list. Print the

list.

Sort and Search the item in the list.

\*/

#include <iostream>

#include <vector>

#include <algorithm>

int main() {

    std::vector<std::string> shoppingList;

    // Append items to the list

    shoppingList.push\_back("eggs");

    shoppingList.push\_back("milk");

    shoppingList.push\_back("sugar");

    shoppingList.push\_back("chocolate");

    shoppingList.push\_back("flour");

    // Print the list

    std::cout << "Shopping List:" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Remove the first element

    shoppingList.erase(shoppingList.begin());

    // Print the modified list

    std::cout << "\nAfter removing the first item:" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Insert "coffee" at the beginning

    shoppingList.insert(shoppingList.begin(), "coffee");

    // Print the modified list

    std::cout << "\nAfter inserting 'coffee' at the beginning:" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Find and replace "sugar" with "honey"

    for (std::string& item : shoppingList) {

        if (item == "sugar") {

            item = "honey";

        }

    }

    // Print the modified list

    std::cout << "\nAfter replacing 'sugar' with 'honey':" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Insert "baking powder" before "milk"

    auto it = std::find(shoppingList.begin(), shoppingList.end(), "milk");

    if (it != shoppingList.end()) {

        shoppingList.insert(it, "baking powder");

    }

    // Print the modified list

    std::cout << "\nAfter inserting 'baking powder' before 'milk':" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Sort the list

    std::sort(shoppingList.begin(), shoppingList.end());

    // Print the sorted list

    std::cout << "\nSorted Shopping List:" << std::endl;

    for (const std::string& item : shoppingList) {

        std::cout << item << std::endl;

    }

    // Search for an item in the list

    std::string searchItem = "chocolate";

    auto searchResult = std::find(shoppingList.begin(), shoppingList.end(), searchItem);

    if (searchResult != shoppingList.end()) {

        std::cout << "\n'" << searchItem << "' found in the list." << std::endl;

    } else {

        std::cout << "\n'" << searchItem << "' not found in the list." << std::endl;

    }

    return 0;

}