National University of Computer & Emerging Sciences, Karachi
Spring-2023 School of Computing (BSCS, BSCS-R, BSSE, BSCY, BSAI)
Midterm 2
3rd Apr 2023, 10:00 am – 11:00 am

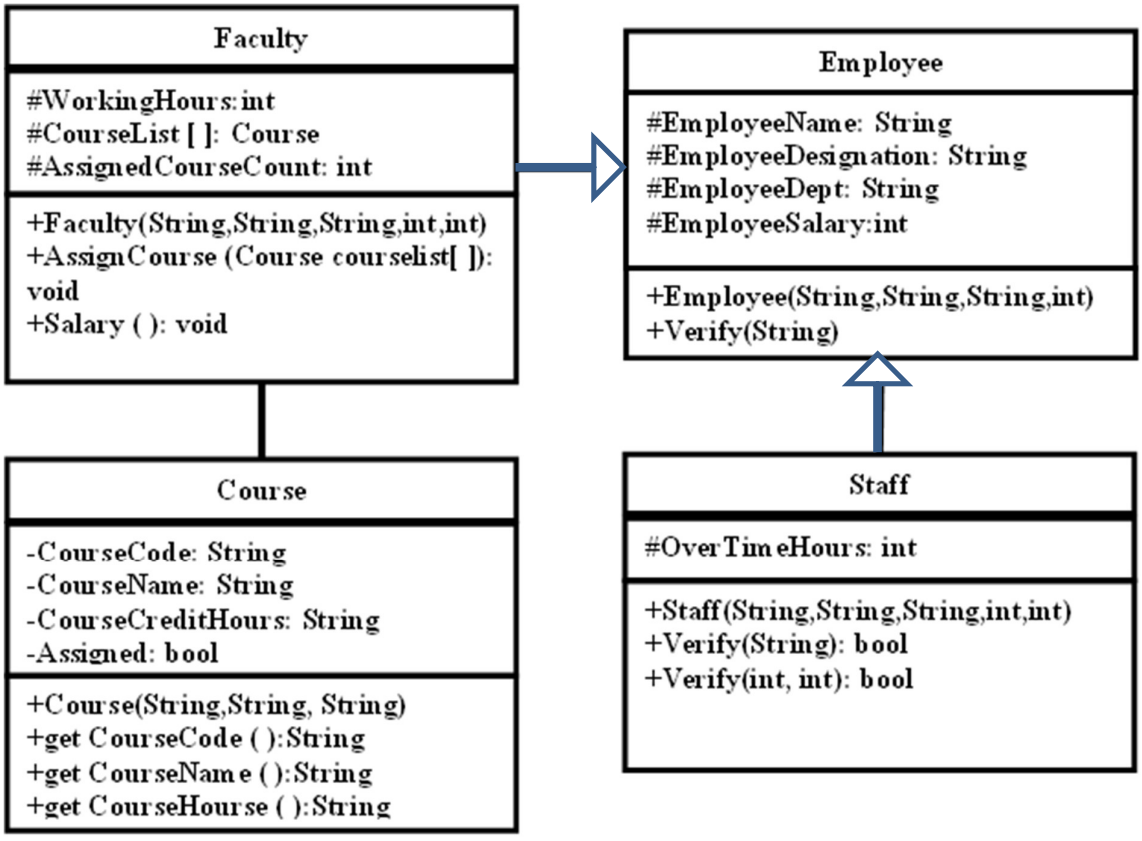
Course Code: CS1004	Course Name: Object Oriented Programming
Instructors Name: Dr. Farooque Hassan Kumbhar, Dr. Abdul Aziz, Mr. Zain-ul-Hassan, Ms. Abeer Gauher, Mr. Basit Ali, Ms. Sobia Iftikhar, Ms. Aqsa Zahid, Ms. Sumaiyah, Ms. Abeeha Sattar, Ms Javeria Farooq, Mr. Shahroz Bakht, Ms. Eman Shahid	
Student Roll No:	Section No:

Instructions:

- Return the question paper and make sure to keep it inside your answer sheet.
- Read questions completely before answering. There are **4 questions, 2 sides on 1 page**.
- In case of any ambiguity, you may make assumptions. But your assumption should not contradict any statement in the question paper.
- You are **not allowed to write** anything on the question paper (except your ID and section).

Time: 60 minutes. **Max Marks**: 60 Marks

A university has decided to update its employee portal. The class diagram given below shows an updated view of the employee portal. Implement the class diagram according to each question's requirement.



Recommended Rubrics:

Q1. Employee class (5), Inherited Faculty class (5), and Course class (5). All with parameterized constructors.

Q2. Assign Course Function (10), Salary Function (5)

Q3. Operator overloaded in three different classes. Postfix ++ operator in Employee class (5), Postfix ++ operator in Faculty class (5), Binary + operator in Staff class (5)

Q4. Staff Class (5), Verify function with string (5), Verify function with two numeric (5)

Q 1- **[15 min, 15 Marks, CLO 2]** Create an "Employee" class that has attributes name, designation, department, and salary as attributes. The class also has a parameterized constructor that sets these attributes. Derive a sub-class "Faculty" that has additional attributes working hours and course list. Create a parameterized constructor that sets these attributes and also invoke the base class's constructor. Create a separate "Course" class for Has-A relation that has name, course code and credit hours as attributes. Create a parameterized constructor that sets these attributes. Create getter methods for all the attributes.

```
#include<iostream>
#include<string>
using namespace std;
class Employee {
protected:
   string EmployeeName, EmployeeDesignation, EmployeeDept;
   double EmployeeSalary;

public:
   Employee(string name, string desig, string dept, double salary) {
      EmployeeName = name;
      EmployeeDesignation = desig;
      EmployeeDept = dept;
      EmployeeSalary = salary;
   }
   string getEmployeeName() { return EmployeeName; }
   string getDesignation() { return EmployeeDesignation; }
   string getDepartment() { return EmployeeDept; }
   double getSalary() { return EmployeeSalary; }
   double operator++(int);
};

class Course {
private:
   string CourseName, CourseCode;
   int CourseCreditHours;
   bool Assigned;
public:
   Course(string name, string code, int hours, bool assigned) {
      CourseName = name;
      CourseCode = code;
      CourseCreditHours = hours;
      Assigned = assigned;
   }
   string getCourseName() { return CourseName; }
   string getCourseCode() { return CourseCode; }
   int getCreditHours() { return CourseCreditHours; }
   bool isAssigned() { return Assigned; }
   void setAssigned(bool f) {Assigned = f;}
};
```

```
class Faculty : public Employee {
private:
   int workingHours;
   Course* courseList;
   int AssignedCourseCount;

public:
   Faculty(string name, string desig, string dept, double salary, int hours, Course* courses) :
Employee(name, desig, dept, salary) {
      workingHours = hours;
      courseList = courses;
      AssignedCourseCount = 0;
   }
   Faculty(string name, string desig, string dept, double salary, int hours): Employee(name, desig,
dept, salary), workingHours(hours){}

   int getWorkingHours() { return workingHours; }
   Course* getCourses() { return courseList; }

   void AssignCourse(Course courseList[]);
   void Salary();
   Faculty operator++(int);
};
```

Q 2- **[15 min, 15 Marks, CLO 3]** Create a function **AssignCourse** that takes all courses and assigns one course per call to the current faculty object based on the following criteria:

a. If the calling object is from "Computer Science" department, then assign the available course with course code starting with "C".
b. If the calling object is from "Management Science" department, then assign the available course with course code starting with "M".
c. If the calling object is from "Electrical Engineering" department, then assign the available course with course code starting with "E".
d. While assigning courses to the faculty, do invoke a warning message if the total assigned credit hours exceed maximum 12 credit hours.

Also create the **Salary** function calculates and prints the salary on the following criteria: If the faculty's working hours are equal to 36 display the current salary. If the faculty's working hours are more than 36 then add 1000 Rs for each extra hour and display the updated salary.

```
void Faculty::Salary() {
   int baseSalary = EmployeeSalary;
   int extraHours = workingHours - 36;
   if (extraHours > 0) {
      baseSalary += extraHours * 1000;
   }
   std::cout << "Current Salary: " << baseSalary << " Rs" << std::endl;
}
void Faculty::AssignCourse(Course courseList[]) {
        int totalCreditHours = AssignedCourseCount * 3;
        int allCourses = 20;
        if (totalCreditHours > 12) {
                cout << "Warning: Assigned credit hours (" << totalCreditHours << ") exceed
maximum limit of 12." << endl;
        for (int i = 0; i < allCourses; i++) {
                if (!courseList[i].isAssigned()) {
                        if (EmployeeDept == "Computer Science" &&
courseList[i].getCourseCode().substr(0, 1) == "C") {
                                courseList[i].setAssigned(true);
                                totalCreditHours += courseList[i].getCreditHours();
                                cout << "Course " << courseList[i].getCourseName() << " assigned to
" << getEmployeeName() << endl;
                                i = allCourses;
                        }
                        else if (EmployeeDept == "Management Science" &&
courseList[i].getCourseCode().substr(0, 1) == "M") {
                                courseList[i].setAssigned(true);
                                totalCreditHours += courseList[i].getCreditHours();
                                cout << "Course " << courseList[i].getCourseName() << " assigned to
" << getEmployeeName() << endl;
                                i = allCourses;
                        }
                        else if (EmployeeDept == "Electrical Engineering" &&
courseList[i].getCourseCode().substr(0, 1) == "E") {
                                courseList[i].setAssigned(true);
                                totalCreditHours += courseList[i].getCreditHours();
                                cout << "Course " << courseList[i].getCourseName() << " assigned to
" << getEmployeeName() << endl;
                                i = allCourses;
                        }
                }
        }
        }
}
```

Q 3- **[15 min, 15 Marks, CLO 4]** You are required to implement functionalities that enable the following operations to work in the main function.

- *Employee ob1("Ali", "Lecturer", "Electrical Engineering", 150000);*
  int increased10percentage = (ob1++)
- *Faculty ob2("Jawed", "Instructor", "Computer Science", 100000, 0);*
  ob2 = ob2++; /* The updated object has 20% increased salary and updated designation to lecturer from instructor /to assistant professor from lecturer. */
- *Faculty ob3("Naveed", "Instructor", "Management Science", 100000, 0);*
  Staff ob4("Majeed", "Instructor", "Computer Science", 100000, 0);
  int sumOfSalaries = ob4 + ob3;

```
// Overloaded ++ operator for Employee class
double Employee::operator++(int) {
   double increase = EmployeeSalary * 0.1;
   EmployeeSalary += increase;
   return EmployeeSalary;
}

// Overloaded ++ operator for Faculty class
Faculty Faculty::operator++(int) {
   double increase = EmployeeSalary * 0.2;
   EmployeeSalary += increase;
   if (EmployeeDesignation == "Instructor") {
      EmployeeDesignation = "Lecturer";
   } else if (EmployeeDesignation == "Lecturer") {
      EmployeeDesignation = "Assistant Professor";
   }
   return *this;
}
// Overloaded + operator for Staff class
int Staff::operator + (Faculty & ob)
   {
      return EmployeeSalary + ob.getSalary();
   }
```

Q 4- **[15 min, 15 Marks, CLO 3]** Derived class "Staff" has additional attributes of overtime hours per week and parameterized constructors that initialize inherited and specialized members. The Verify function have two renditions in the staff class as given below:

- Create a function named "Verify" that takes department name as input and returns a Boolean value (true or false) if the name of department matches with the department of this staff member.
- Overload the function "Verify" such that it takes two numeric values as parameters and returns a Boolean value to indicate if the salary of this staff member lies between the range of these values.

```
class Staff : public Employee {
private:
   int overtimeHours;

public:
   Staff(string name, string desig, string dept, float salary, int overtime)
      : Employee(name, desig, dept, salary), overtimeHours(overtime)
   {}

   // Verify function with department name parameter
   bool Verify(string deptName) const {
      return (EmployeeDept == deptName);
   }

   // overloaded Verify function with salary range parameters
   bool Verify(float minSalary, float maxSalary) const {
      return (EmployeeSalary >= minSalary && EmployeeSalary <= maxSalary);
   }

   int operator + (Faculty & ob)
   {
      return EmployeeSalary + ob.getSalary();
   }
};
```