Assignment 0: Revision of CS2255

Index:

Problem 1:

Question Problem 1

```
Problem 1: The code is attached below for class for complex number. Provide
operator overloading functions to facilitate the following operations on any
complex numbers A, B and C:

A == B; overload Boolean operator == to test if A and B are equal

C = A*B; overload multiplication operator *

A = B; overload assignment operator =

cin >> A >> B; overload stream operator >> to take user input for real and
imaginary part of the number (You might want to create a friend function to
facilitate this)

Also, write a static function for the complexNumber class which initializes
numCount variable. Your function takes an integer argument provided by user.
Assign this argument to numCount variable.
```

Solution Problem 1

```cpp
#include <iostream>
#include <format>

class ComplexNum
{
private:
    static int numCount;
    int realPart, imaginaryPart;

public:
    ComplexNum()
    {
        numCount++;
    }

    ComplexNum(int realPart, int imaginaryPart)
    {
        numCount++;
        this->realPart = realPart;
        this->imaginaryPart = imaginaryPart;
    }

    ~ComplexNum()
    {
        numCount--;
    }

    ComplexNum(const ComplexNum &otherComplexNum)
    {
        numCount++;
        realPart = otherComplexNum.realPart;
        imaginaryPart = otherComplexNum.imaginaryPart;
    }

    void printComplex()
    {
        std::cout << std::format(
            "Num of total active complex num: {}\n Complex Num: {} + {}i\n",
            numCount,
            this->realPart,
            this->imaginaryPart);
    }

    ComplexNum operator+(const ComplexNum &b) const
    {
        ComplexNum temp;
        temp.realPart = this->realPart + b.realPart;
        temp.imaginaryPart = this->imaginaryPart + b.imaginaryPart;
        return temp;
    }

    ComplexNum operator*(const ComplexNum &b) const
    {
        ComplexNum temp;
        temp.realPart = (this->realPart * b.realPart) - (this->imaginaryPart *
b.imaginaryPart);
        temp.imaginaryPart = (this->realPart * b.imaginaryPart) + (this-
>imaginaryPart * b.realPart);
        return temp;
    }

    ComplexNum &operator=(const ComplexNum &b)
    {
        this->realPart = b.realPart;
        this->imaginaryPart = b.imaginaryPart;
        return *this;
    }

    friend std::istream &operator>>(std::istream &in, ComplexNum &complexNum)
    {
        std::cout << "Enter Real part: ";
        in >> complexNum.realPart;
        std::cout << "Enter Imaginary part: ";
        in >> complexNum.imaginaryPart;
        return in;
    }

    friend std::ostream &operator<<(std::ostream &out, const ComplexNum
&complexNum)
    {
        out << "Complex Num: "
            << complexNum.realPart
            << " + "
            << complexNum.imaginaryPart
            << "i";
        return out;
    }

    friend bool operator==(const ComplexNum &a, const ComplexNum &b)
    {
        return (a.realPart == b.realPart) && (a.imaginaryPart == b.imaginaryPart);
    }
};

int ComplexNum::numCount = 0;

int main()
{
    ComplexNum A, B, C;

    std::cout << "Enter complex number A\n";
    std::cin >> A;
    std::cout << "Enter complex number B\n";
    std::cin >> B;

    A.printComplex();
    B.printComplex();

    if (A == B)
    {
        std::cout << "A and B are equal." << std::endl;
    }
    else
    {
        std::cout << "A and B are not equal." << std::endl;
    }

    C = A * B;
    std::cout << "A * B = " << C << std::endl;
    A = B;
    std::cout << "A after assignment: " << A << std::endl;

    return 0;
}
```

Output Problem 1

```
===============================================================
Running: build/bin/main.exe
Enter complex number A
Enter Real part: 1
Enter Imaginary part: 2
Enter complex number B
Enter Real part: 1
Enter Imaginary part: 2
Num of total active complex num: 3
 Complex Num: 1 + 2i
Num of total active complex num: 3
 Complex Num: 1 + 2i
A and B are equal.
A * B = Complex Num: -3 + 4i
A after assignment: Complex Num: 1 + 2i
===============================================================
```

Problem 2:

Question Problem 2

```
Problem 2. You are to design an abstract class called Employee whose members are
as given below (make them protected):

Data members:
char *name
long int ID

Two constructors:

A Default constructor // intitialize data memebrs to the default values

and a copy constructor

Methods:
```

```
setPerson (char *n, long int id) //allows user to set information for each person

A function called Print () // should be a virtual function, that prints the data
attributes of the class.

and a destructor

Also define two classes that derived from class Employee, called Manager and
Secretary. Each class should inherit all members from the base class and has its
own data members and member functions as well.

The Manager should have a data member called degree for his/her undergraduate
degree (e.g. diploma, bachelor, master, doctor), the Secretary should have her
contract (can be a Boolean value 1/0 for permanent/temporary).

All member functions of derived class should be override from their base class.
```

Solution Problem 2

```
#include <iostream>
#include <cstring>

class Employee
{
protected:
    char *m_name;
    long int m_ID;

public:
    Employee() : m_name(nullptr), m_ID(0) {}
    Employee(const char *name, long int ID) : m_ID(ID)
    {
        this->m_name = new char[strlen(name) + 1];
        strcpy(this->m_name, name);
    }
    virtual ~Employee()
    {
        delete[] this->m_name;
    }

    virtual void setPerson(const char *name, long int ID)
    {
        delete[] m_name;
        this->m_name = new char[strlen(name) + 1];
        strcpy(this->m_name, name);
        this->m_ID = ID;
    }

    virtual void Print() = 0;
};
```

```
class Manager : public Employee
{
private:
    char *m_degree;

public:
    Manager(const char *name, long int ID, const char *degree) : Employee(name,
ID)
    {
        this->m_degree = new char[strlen(degree) + 1];
        strcpy(this->m_degree, degree);
    }

    ~Manager()
    {
        delete[] this->m_degree;
    }

    void setPerson(const char *name, long int ID, const char *degree)
    {
        Employee::setPerson(name, ID);
        delete[] this->m_degree;
        this->m_degree = new char[strlen(degree) + 1];
        strcpy(this->m_degree, degree);
    }

    void Print() override
    {
        std::cout << "Name: " << this->m_name
                  << ", m_ID: " << this->m_ID
                  << ", Degree: " << this->m_degree
                  << std::endl;
    }
};

class Secretary : public Employee
{
private:
    bool m_isPermanent;

public:
    Secretary() : m_isPermanent(false) {}

    void setPerson(const char *name, long int ID, bool isPermanent)
    {
        Employee::setPerson(name, ID);
        m_isPermanent = isPermanent;
    }

    void Print() override
    {
        std::cout << "Name: " << m_name
                  << ", m_ID: " << m_ID
                  << ", Contract: " << (m_isPermanent ? "Permanent" : "Temporary")
                  << std::endl;
    }
};

int main()
{
    Employee *p = new Manager("Bruce Lee", 234567, "Dr.");
    p->Print();

    Secretary p2;
    p2.setPerson("Wilma Jones", 341256, true);
    p2.Print();

    delete p;

    p = &p2;
    p->Print();

    return 0;
}
```

Output Problem 2

```
===============================================================
Running: build/bin/main.exe
Name: Bruce Lee, m_ID: 234567, Degree: Dr.
Name: Wilma Jones, m_ID: 341256, Contract: Permanent
Name: Wilma Jones, m_ID: 341256, Contract: Permanent
===============================================================
```