

CSE-1221 Final Autumn 2022 Solution

Name: **Ezaz Ahmed** ID: **C223009** Section: **2AM**

Group-A

1(a)

Ans:

Operator overloading refers to the ability to redefine the behavior of an operator when it is applied to objects of a particular class or data type. In object-oriented programming, operators such as +, -, *, /, and = are predefined with specific meanings for built-in data types like integers, floats, and strings. However, operator overloading allows you to redefine these operators for custom classes, enabling them to perform specific actions or operations on objects of those classes.

Here are some of the reasons why it is necessary to overload an operator:

- To make code more readable and concise.
- To provide additional functionality to user-defined types.
- To make code more efficient.
- To make code more consistent.

1(b)

Ans:

The main difference between operator functions and normal functions is that operator functions are specifically designed to work with operators, allowing user-defined objects to behave like built-in types and enabling the use of operators with those objects. On the other hand, normal functions are general-purpose functions that perform specific tasks or operations but are not directly associated with any particular operator or class.

Example:

```
#include <iostream>
using namespace std;
class A
{
private:
    int a;
public:
    A(int a) : a(a) {}

    // Operator function
    A operator+(const A& b)
```

```

    {
        return A(a + b.a);
    }
    // Normal function
    int getValue()
    {
        return a;
    }
};
int main()
{
    A num1(1);
    A num2(10);

    A sum = num1 + num2;

    cout << "Sum: " << sum.getValue() << endl;

    return 0;
}

```

The code you provided defines a class A with two member functions: operator+() and getValue(). The operator+() function is an operator function that overloads the + operator. This function takes a reference to an A object as its argument and returns an A object. The getValue() function is a normal function that returns the value of the a member variable.

The main() function creates two A objects, num1 and num2. It then calls the operator+() function to add the two objects together and stores the result in the sum variable. Finally, the main() function calls the getValue() function to get the value of the sum variable and prints it to the console.

1(b) or,

Ans:

```

#include <iostream>
using namespace std;
class A
{
private:
    int a;

public:

```

```

A(int a) : a(a) {}

int get()
{
    return a;
}

friend A operator+(int num, const A& obj);
};

A operator+(int num, const A& obj)
{
    return A(num + obj.a);
}

int main()
{
    A Ob1(5);
    A Ob2 = 20 + Ob1;
    cout << "Ob2: " << Ob2.get() << endl;
    return 0;
}

```

1(c)

Ans:

```

#include <iostream>
using namespace std;
class Point
{
private:
    int x;
    int y;

public:
    Point(int x, int y) : x(x), y(y) {}

    int getX() const
    {
        return x;
    }
}

```

```

    }

    int getY() const
    {
        return y;
    }

    friend Point operator-(const Point& p1, const Point& p2);
};

Point operator-(const Point& p1, const Point& p2)
{
    int X = p1.x - p2.x;
    int Y = p1.y - p2.y;

    return Point(X,Y);
}

int main()
{
    Point p1(1, 3);
    Point p2(9, 5);

    Point diff = p1 - p2;

    cout<<"("<<diff.getX() << ", " << diff.getY() << ")";

    return 0;
}

```

1(d)

Ans:

The code will fail to compile. It will produce compilation errors. Therefore, the console will not show any output. The errors are:-

1. The operator+ function is missing a return type. It should be declared as CSE operator+(CSE ob).
2. The operator+ function is using an undefined variable i when calculating the values of temp.a and temp.b. It should be ob.a and ob.b respectively.
3. The main function should have a return type of int instead of void.
4. The variable declarations inside the main function use the class name A instead of CSE.

2(a)

Ans:

```
#include <iostream>
using namespace std;
```

```
class A
{
protected:
    int a;

public:
    A()
    {
        a = 10;
    }

    void printA()
    {
        cout << "Protected Number: " << a << endl;
    }
};
```

```
class B : public A
{
public:
    void accessProtectedMember()
    {
        cout << "Accessing protected member in the derived class." << endl;
        a += 5;
        printA();
    }
};
```

```
int main()
{
    B b;
    b.accessProtectedMember();
    return 0;
}
```

It demonstrates the accessibility of protected members in C++. The **A** class has a protected member **a** and a public method **printA** to print its value. The **B** class publicly inherits from **A**. Inside the **B** class, the method **accessProtectedMember** accesses and modifies the protected member **a** by adding 5 to it and then calls the **printA** method to display the updated value.

2(b)

Ans:

Multiple inheritance is a feature of some object-oriented programming languages that allows a class to inherit from multiple base classes. This can be useful for creating classes that have a complex set of behaviors or properties. However, multiple inheritance can also lead to problems, such as the diamond problem.

```
class A {  
public:  
    int a;  
};
```

```
class B : public A {  
public:  
    int b;  
};
```

```
class C : public A {  
public:  
    int c;  
};
```

```
class D : public B, public C {  
};
```

In this code, the D class inherits from both the B class and the C class, which both inherit from the A class. This means that the D class has two copies of the A class's a member. This can lead to confusion when trying to access the a member.

Virtual inheritance is a feature of some object-oriented programming languages that can be used to solve the diamond problem. Virtual inheritance allows a class to inherit from a base class only once, even if the base class is inherited from by multiple other classes. For example, the following code shows how the D class can be declared using virtual inheritance:

```
class A {  
public:  
    int a;  
};
```

```
class B : public virtual A {  
public:  
    int b;  
};
```

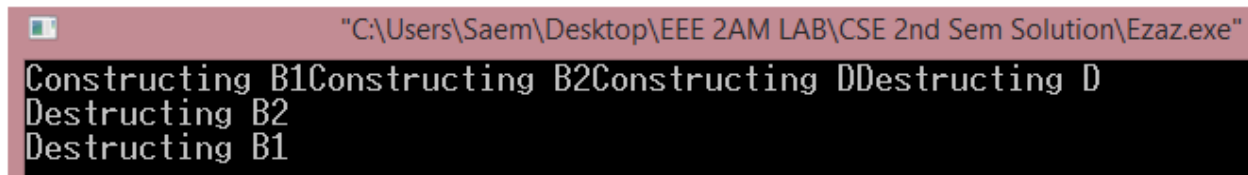
```
class C : public virtual A {  
public:  
    int c;  
};
```

```
class D : public B, public C {  
};
```

In this code, the D class inherits from the B class and the C class, but it only inherits from the A class once. This means that the D class only has one copy of the A class's a member.

2(c)

Ans:

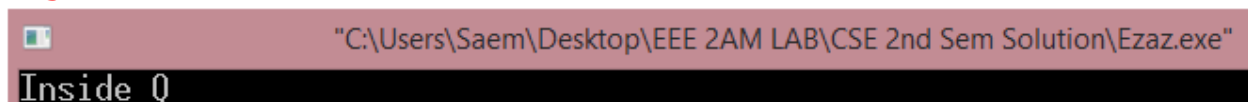


```
"C:\Users\Saem\Desktop\EEE 2AM LAB\CSE 2nd Sem Solution\Ezaz.exe"  
Constructing B1Constructing B2Constructing DDestructing D  
Destructing B2  
Destructing B1
```

The constructors of the base classes are called in the order in which they are declared, and the destructors are called in the reverse order. In this case, the base classes are B1 and B2, and D inherits from both of them. So, the constructor for B1 is called first, followed by the constructor for B2, and finally the constructor for D. When the program exits, the destructors are called in the reverse order, so the destructor for D is called first, followed by the destructor for B2, and finally the destructor for B1.

2(c) or

Ans:



```
"C:\Users\Saem\Desktop\EEE 2AM LAB\CSE 2nd Sem Solution\Ezaz.exe"  
Inside Q
```

The print() function is not defined in class R, so the compiler looks for it in the base classes. The print() function is defined in both P and Q, but Q is derived from P, so the compiler will call the print() function in Q.

2(d)

Ans:

```
#include <iostream>
using namespace std;
class Fruit
{
protected:
    int fruits;
```

```
public:
    Fruit()
    {
        fruits = 0;
    }

    int total()
    {
        return fruits;
    }
};
```

```
class Apples : public Fruit
{
public:
    Apples(int count)
    {
        fruits = count;
    }
};
```

```
class Mangoes : public Fruit
{
public:
    Mangoes(int count)
    {
        fruits = count;
    }
};
```



```

};

int main()
{
    Apples apples(5);
    Mangoes mangoes(3);
    cout << "Number of Apples: " << apples.total() << endl;
    cout << "Number of Mangoes: " << mangoes.total() << endl;
    cout << "Total number of Fruits: " << apples.total() + mangoes.total() << endl;

    return 0;
}

```

The Fruit class is the base class for the Apples and Mangoes classes. The Fruit class has a protected member variable called fruits. The Fruit class also has a public member function called total(). This function returns the value of the fruits member variable. The Apples and Mangoes classes inherit from the Fruit class. This means that they have access to the fruits member variable and the total() function. In the main() function, I have created an Apples object and a Mangoes object. Then calling the total() function on each object to get the number of fruits in each object.

Group-B

3(a)

Ans:

Virtual Function: A virtual function is a function that can be overridden in a derived class. This allows the derived class to provide its own implementation of the function.

Virtual functions are needed to achieve runtime polymorphism, allowing different objects of derived classes to be treated uniformly through a common base class interface

3(b)

Ans:

Here are the reasons to use pure virtual function:

1. **To prevent instantiation of abstract classes:** A pure virtual function is a function that has no implementation. This means that an abstract class, which is a class that contains at least one pure virtual function, cannot be instantiated. This can be useful to prevent users from creating objects of an abstract class, which would not be meaningful.

2. **To force derived classes to implement certain functions:** If a base class has a pure virtual function, then any derived class that inherits from that base class must also implement that function. This can be useful to ensure that all objects of a certain type have certain functionality.
3. **To improve type safety:** When a function is pure virtual, the compiler can ensure that it is only called on objects of the correct type. This can help to prevent errors caused by calling a function on an object of the wrong type.

```
#include <iostream>
using namespace std;
class A
{
public:
    virtual void print() = 0;
};

class B : public A
{
public:
    void print()
    {
        cout << "Hello" << endl;
    }
};

class C : public A
{
public:
    void print()
    {
        cout << "IIUC!" << endl;
    }
};

int main()
{

    B D;
    D.print();
}
```

```
C E;  
E.print();  
return 0;
```

3(c)

Ans:

In object-oriented programming (OOP), polymorphism refers to the ability of objects of different classes to be treated as objects of a common superclass. It allows objects to be processed in a generic way, irrespective of their specific types.

Polymorphism is based on the principle of inheritance, where subclasses inherit properties and behaviors from a superclass. It allows objects to be represented at a higher level of abstraction, enabling code reusability and flexibility.

| Feature | Early Binding | Late Binding |
|----------------------------|---------------|----------------|
| Type of polymorphism | Static | Dynamic |
| Time of type determination | Compile time | Runtime |
| Speed | Faster | Slower |
| Flexibility | Less flexible | More flexible |
| Debugging | Easier | More difficult |

Static polymorphism:

```
#include <iostream>  
using namespace std;  
class Calculator  
{  
public:  
    int add(int a, int b)  
    {  
        return a + b;  
    }  
}
```

```

double add(double a, double b)
{
    return a + b;
}

};

int main()
{
    Calculator calculator;

    int result1 = calculator.add(5, 10);
    cout << "Result 1: " << result1 << endl;

    double result2 = calculator.add(2.5, 3.7);
    cout << "Result 2: " << result2 << endl;

    return 0;
}

```

Dynamic polymorphism:

```

#include <iostream>
using namespace std;
class Calculator
{
public:
    virtual int add(int a, int b) = 0;
    virtual double add(double a, double b) = 0;
};

class IntCalculator : public Calculator
{
public:
    int add(int a, int b) override
    {
        return a + b;
    }

    double add(double a, double b) override
    {

```

```

        return a + b;
    }
};

class DoubleCalculator : public Calculator
{
public:
    int add(int a, int b) override
    {
        return a + b;
    }

    double add(double a, double b) override
    {
        return a + b;
    }
};

int main()
{
    Calculator* calculator = new IntCalculator();

    int result1 = calculator->add(5, 10);
    cout << "Result 1: " << result1 << endl;

    delete calculator;

    calculator = new DoubleCalculator();

    double result2 = calculator->add(2.5, 3.7);
    cout << "Result 2: " << result2 << endl;

    delete calculator;

    return 0;
}

```

4(a)

Ans:

In computer programming, an exception is an event that occurs during the execution of a program that disrupts the normal flow of execution. Exceptions can be caused by a variety of factors, such as invalid input, division by zero, or an attempt to access a memory location that is not accessible.

Here are some advantages of using exception handling mechanism in a program:

- **Error handling:** Exceptions can be used to handle errors in a program. This can help to prevent the program from crashing and can also help to provide a more user-friendly experience.
- **Program flow:** Exceptions can be used to control the flow of a program. This can be useful for things like logging errors or displaying error messages.
- **Debugging:** Exceptions can be used to debug programs. This can help to identify the source of errors and to fix them.

4(a) or,

Ans:

The general form of **try**, **catch**, and **throw** in C++ for exception handling is as follows:

```
try {  
    // Code that may throw an exception  
    // ...  
    throw SomeException(); // Throw an exception explicitly  
}  
catch (ExceptionType1& e1) {  
    // Handle ExceptionType1  
    // ...  
}  
catch (ExceptionType2& e2) {  
    // Handle ExceptionType2  
    // ...  
}  
catch (...) {  
    // Catch all other exceptions  
    // ...  
}
```

The try block is used to enclose code that might throw an exception. If an exception is thrown in the try block, the program will transfer control to the first catch block

that can handle the exception. If there is no catch block that can handle the exception, the program will terminate.

The **catch** block is used to handle an exception. The catch block takes an exception object as a parameter. The exception object contains information about the exception, such as the type of exception and the message that was associated with the exception.

The **throw** keyword is used to throw an exception. The throw keyword takes an exception object as a parameter. The exception object will be thrown to the nearest catch block that can handle the exception.

4(b)

Ans:

STL stands for Standard Template Library. It is a library of C++ templates that provides a set of common data structures and algorithms. STL is used to represent and manipulate data in C++ programs.

A container is a data structure that stores a collection of elements. STL provides a variety of containers, such as vectors, lists, and sets.

An iterator is an object that points to an element in a container. Iterators are used to access and manipulate the elements in a container.

An algorithm is a procedure that performs a specific operation on a collection of data. STL provides a variety of algorithms, such as sorting algorithms, searching algorithms, and mathematical algorithms.

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    vector<int> v = {9, 2, 3, 4, 5}; // Create a vector container to hold integers.
    sort(v.begin(), v.end()); //sort algorithm
    for (int i : v) // Iterate over the sorted vector and print each element
    {
        cout << i << " ";
    }
}
```

```
    return 0;
}
```

4(c)

Ans:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
    vector<int> v;
    cout << "Enter 20 integers:\n";
    for(int i = 0; i < 20; i++)
    {
        int a;
        cin >> a;
        v.push_back(a);
    }
    int N;
    cout << "Enter the value of N: ";
    cin >> N;
    sort(v.begin(), v.begin() + N);
    cout << "Sorted vector of (first " << N << " elements): ";
    for (int i = 0; i < N; i++)
    {
        cout << v[i] << " ";
    }
    cout << endl;
    return 0;
}
```

4(c) or,

Ans:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main()
{
```



```

vector<int> v(10, 6);
cout<<"Before Erasing"<<endl;
for (int a : v)
{
    cout <<a << " ";
}
cout <<endl;
cout<<"After Erasing"<<endl;
v.erase(v.begin()+2,v.begin()+7);
for (int a : v)
{
    cout << a << " ";
}
cout <<endl;
return 0;
}

```

5(a)

Ans:

A stream is an abstraction that represents a sequence of characters that can be read from or written to. Streams provide a convenient way to perform input/output (I/O) operations in a standardized manner, regardless of the actual physical source or destination of the data.

The stream classes hierarchy for console I/O operations in C++ includes the following key classes:

1. **ios:** The base class for all I/O stream classes. It provides basic functionality such as formatting flags, error state, and error handling mechanisms.
2. **istream:** The base class for input streams. It provides member functions for reading data from a stream, such as extraction operator (>>), getline, and others.
3. **ostream:** The base class for output streams. It provides member functions for writing data to a stream, such as insertion operator (<<), write, and others.
4. **iostream:** The base class for both input and output streams. It inherits from both istream and ostream, allowing bidirectional I/O operations.
5. **ifstream:** A class derived from istream that provides input operations specifically for file streams. It is used for reading data from files.
6. **ofstream:** A class derived from ostream that provides output operations specifically for file streams. It is used for writing data to files.
7. **stringstream:** A class derived from iostream that allows input/output operations on strings as if they were streams. It can be used for string manipulation and parsing.

5(a) or,

Ans:

A manipulator is a special function or object that is used to modify the behavior of input/output streams. Manipulators are typically used to perform specific formatting or state manipulation operations on streams.

Here are the differences between manipulators and ios member functions:

| Manipulators | ios Member Functions |
|---|---|
| Manipulators are functions or objects that modify stream state. | ios member functions are member functions of the ios base class that provide various stream operations. |
| Manipulators are applied using the insertion (<<) operator. | ios member functions are called directly on a stream object. |
| Manipulators are typically used for formatting purposes, such as setting precision, width, or manipulating flags. | ios member functions provide a wide range of operations, including error handling, positioning, flushing, and others. |
| Manipulators can be chained together using multiple insertion operators. | ios member functions are called individually on a stream object. |
| Examples of manipulators include setw, setprecision, fixed, hex, endl, etc. | Examples of ios member functions include width, precision, flags, clear, seekg, flush, etc. |

5(b)

Ans:

```
#include <iostream>
using namespace std;
class VECTOR
{
private:
    int x;
    int y;
public:
    VECTOR(int xVal = 0, int yVal = 0) : x(xVal), y(yVal) {}

    void print() const
    {
```

```

        cout << "VECTOR(" << x << ", " << y << ")";
    }
};
int main()
{
    VECTOR vec(3, 5);
    vec.print();
    cout << endl;
    return 0;
}

```

5(b) or,

Ans:

```

#include <iostream>
using namespace std;
class VECTOR
{
private:
    int x;
    int y;

public:
    VECTOR(int xVal = 0, int yVal = 0) : x(xVal), y(yVal) {}

    void print() const
    {
        cout << "VECTOR(" << x << ", " << y << ")";
    }
    friend istream& operator>>(istream& is, VECTOR& v)
    {
        cout << "Enter x and y: ";
        is >> v.x >> v.y;
        return is;
    }
};
int main()
{
    VECTOR vec;
    cin >> vec;
    vec.print();
}

```

```
    cout << endl;
    return 0;
}
```

5(c)

Ans:

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    int high_score;
    ifstream f1("templeRun.txt");

    if (!f1)
    {
        ofstream f2("templeRun.txt");

        cout << "Enter the high score: ";
        cin >> high_score;
        f2 << high_score;
        f2.close();
    }
    else
    {
        f1 >> high_score;
    }

    cout << "Highscore: " << high_score << endl;

    return 0;
}
```

If the file doesn't exist first I have to inset a integer in the file:

```
"C:\Users\Saem\Desktop\EEE 2AM LAB\CSE 2nd Sem Solution\Ezaz.exe"  
Enter the high score: 13  
Highscore: 13
```

If the file does exist only the highscore will be shown:

```
"C:\Users\Saem\Desktop\EEE 2AM LAB\CSE 2nd Sem Solution\Ezaz.exe"  
Highscore: 13
```

5(c) or,

Ans:

এই প্রশ্নে একটু কনফিউশন আছে আমার। এখানে আসলে Data গুলো কি কোডের মধ্যেই আগে থেকে লিখে ফেলতে হবে নাকি Data গুলো Keyboard থেকে নিতে হবে এটা আসলে প্রশ্নে ঠিকভাবে বলা হয়নি। তাই আমি দুইভাবেই Solve করে দিয়েছি।

Solution-01

```
#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main()  
{  
    ofstream f1("WhoAreYou.txt");  
  
    if (f1)  
    {  
        f1 << "Name: abcd" << endl;  
        f1 << "Semester: Autumn 2022" << endl;  
        f1 << "Course Code: CSE-1221" << endl;  
        f1 << "Course Title: Computer Programming 2" << endl;  
  
        cout << "Data written to the file successfully." << endl;  
        f1.close();  
    }  
}
```

```

    }
    else
    {
        cerr << "Error opening the file for writing." << endl;
    }

    return 0;
}

```

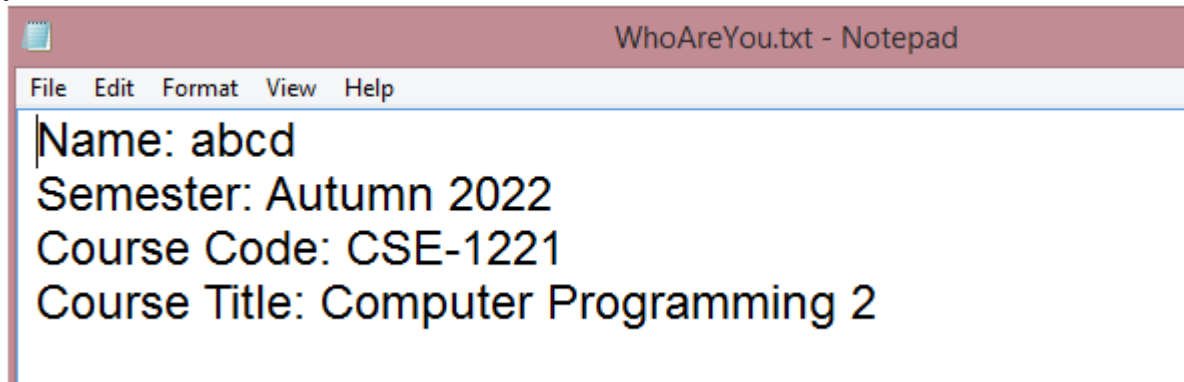
Solution-02

```

#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream f1("WhoAreYou.txt");
    if (f1)
    {
        string name, semester, courseCode, courseTitle;
        cout << "Enter your name: ";
        getline(cin, name);
        cout << "Enter the semester: ";
        getline(cin, semester);
        cout << "Enter the course code: ";
        getline(cin, courseCode);
        cout << "Enter the course title: ";
        getline(cin, courseTitle);
        f1 << "Name: " << name << endl;
        f1 << "Semester: " << semester << endl;
        f1 << "Course Code: " << courseCode << endl;
        f1 << "Course Title: " << courseTitle << endl;
        cout << "Data written to the file successfully." << endl;
        f1.close();
    }
    else
    {
        cerr << "Error opening the file for writing." << endl;
    }
}

```

```
return 0;  
}
```



Thanks Everyone Assalamualikum