3. Operaters in C++

Q.1 Answere the following Questions:

1. Define operator overloading.

→ Operator overloading is a feature in C++ that allow developer to redefine the behaviour of operators (such as +, -, *, etc.) for user define data types. This operator enables the use of these operators with objects in a way that is natural and intuitive, making code more readable and expressive.

2. Which operators cannot be overloaded?

→ Certain operators cannot be overloaded. This include:

i) Scope resolution operators (::)
ii) Member selection operator (.)
iii) Member selection through a pointer to a member. (.*).
iv) Conditional (ternary) operator (?:).

3. Which operator cannot be used with Friend function?

→ The assignment operator (=), function call operator (()), subscript operator ([]), and arrow operator (->) must be overloaded as member functions, not as friend function. This is because they need direct access to the objects internals.

which member function Provide.

4. Define type Conversion.

→ Type Conversion is the process of Converting a value from one data type to another. In C++, type Conversion can be implicit (automatic) or explicit (forced by programmer), allowing Compatibility and flexibility between different data types during Computations or assignments.

Q.2. Answere the following Questions?

(I.) Describe member Dereferencing operators.

→ Member dereferencing operators are used to access members (data or function) of an object in C++. They allow interaction with member of an object or class, especially when dealing with Pointers. The primary members member dereferencing operators are:

- Dot Operator (.) :-

Used to access members of an object directly.

ex:- "object.member" is used to access the member of object.

- Arrow operator (→) :-

. Used to access member of an object when you have a Pointer to the object.

ex:- if "ptr" is a Pointer to an object "ptr → member" is used to access member.

- Deference operator (*) :-

Used to access the object that a Pointer Points to.

ex:- if Vehical *ptr = &car then "*ptr" gives the actual car object, and (*ptr) Speed access speed.

PAGE NO.

2. Explain type Cast operator.

→ The type Cast operator in C++ is used to Convert a value from one data type to another.

There, four main types of casting in C++ :-

(i) Static_Cast :-

It is used for standard Conversation.

like : f = Static_Cast<float>(10);

(ii) Dynamic_Cast :

used in inheritance for safe casting. mainly with Pointers or references.

ex:- Derived *d = dynamic_Cast<Derived*>(baseptr);

(iii) Const_Cast :

used to add or remove Const from a variable.

ex: int *p = Const_Cast<int*>(ConstPtr);

(iv) reinterpret_Cast :

Used for low level Casting, like Converting one Pointer to another.

ex:- int* p = reinterpret_Cast<int*>(ptr);

These operaters help in Conserting types safely and efficiently in OOP.

PAGE NO.

(3.) Explain operator overloading rules.

→

Here, Same rules for operator overloading :-

(i) Operator Perecedence Can not be Changed :- you Can not Changed the defult Precedence or associativity of operators.

(ii) Certain Condition Can not be overloaded :- operators like :: , . , ?: , and Sizeof Can not be overloaded.

(iii) At list one operand must be a user define type :

The operator should involve at least one object of a Custom (user-defined type).

(iv) Number of operands Cannots be Changed : you Cannot after the number of operand the operator takes.

ex: "+" will always take two operands.

(v.) Define Clearly as member or Non-member function :-

Overloaded operator must be defined as either a member function or a non member function.

Q.3 Answare the following Questions:-

(1) Explain "operator overloading with example.

→ Operator overloading allow you to define Custom behaviour for operators (like +, -, *, etc.) when they are used to with objects of user-defined classes.

- Example:

overloading the "+" operator for a Complex number class.

Let's Create a "Complex" class that represents Complex numbers, and overloading the "+" operator to add two Complex numbers.

- Steps:

(i) Define the class:

Define a class that hold real and imaginary parts as its data members.

(ii) Overload the "+" operator:

Define a member function or non-member function to overload the "+" operator.

This will allow to add two Complex numbers using the "+" operator.

→ Code:

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

Class Rectangle {
    public:
    int length, width;

    Rectangle (int l, int w) : length(l), width(w) {}

    int operator + (const Rectangle & other) {
    return (length * width) + (other.length * other.width);
    }
};
int main()
{
    Rectangle rect1 (4,5);
    Rectangle rect2 (3,6);

    int totalArea = rect1 + rect2;

    Cout << " Total Area : " << totalArea << endl;
    return 0;
}
```

2. 3 + 4 * 5
6 + 20

Output:

Total area : 38

PAGE NO.

(2.) Explain type Conversion with an example.

→ Type Conversion in C++ refers to changing a variable from one data type to another.

In OOP, type Conversion can happen between built-in types (like int to float) or between user-defined type.

There are two main types of type Conversion in C++:

(i) Implicit type Conversion (Autometic):

This type of Conversion is automatically done by Compile when you mix different data types in expression.

ex: if you add an int and float, the int will automatically be Converted to float.

→ Code:

```
int a = 5;
float b = 10.5;
float result = a+b;
Cout << "Result: " << result << endl;
```

→ output:

Result: 15.5

(ii) Explicit type Conversion (user-defined):

In Some case you need to define how object of a class can be Converted to another type.

This is called explicit type Conversion and is done by defining a type Conversion operator or a Constructor

→ Code:

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

Class Centimeter {
    Public:
    int cm;
    Centimeter (int c) : cm(c) {}
    Operator float ()
    {
        return cm / 100.00;
    }
};
int main()
{
    Centimeter dist (250);
    float meters = dist;
    Cout << "Distance in meters: " << meter << "meters"
    return 0;
}
```

→ Output:

Distance in meters : 2.5 meters.

Harsha