Solution

QN	Correct Answer	Marking Scheme	Note
2A	<int> int this->*pmf **Output:** Calling the Function:4	• 01 Marks for filling each blank • 02 Mark for output	The answer of this question is **unique**. Thus either full marks will be awarded or **ZERO**.
2B	p = new T*[CAP] delete[] p; template<typename M> delete obj;	• 02 Marks for filling each blank	
3A	The program must have following components. • A base class is inherited by two intermediate classes • A final derived class inherits from both intermediate classes • This creates ambiguity because two copies of the base class exist. Thus the multiple inheritance must be done through virtual base classes.	• 02 Marks for base class • 04 Marks for intermediate class • 01 Mark for final derived class.	• Multiple solutions are possible, but number of classes and their relationship (diamond structure) will be unique. Thus evaluate for structure of the program. • **Deduct 04 marks**, if virtual base class is not used in multiple inheritance.
3B	(i) A() Caught int in risky() ~A() Unknown (ii) A() ~A() MyError occurred	• 02 marks for the correct output of part(i) • 01 marks for the explanation of part (i) • 02 marks for the correct output of part (ii) • 01 marks for the explanation of part (ii)	• Deduct full marks, if outputs are not correct. • Give full marks in explanation. • If output is correct, but in wrong sequence, then deduct full marks of the output.

Selected AnswerKeys

Q4	Write a C++ program to define a class *FLOAT* with: A. Two private integer variables: *mantissa* and *exponent* and a parameterized constructor with default parameters to initialize both members. Write appropriate getter and setter methods. B. Overload the following operators as member functions i. Arithmetic operators: + and * (to achieve floating point arithmetic) ii. Comparison operator: == iii. Assignment operator: = C. Overload arithmetic operators (+, *) to work between a *FLOAT* object and an integer (int) using a friend function. D. Write a main() function to demonstrate: i. All the arithmetic operations between two *FLOAT* objects ii. Comparison using == iii. Assignment using = iv. Arithmetic between a *FLOAT* object and an *int* **Note:** (i) You must perform all arithmetic and comparison operations directly on the mantissa and exponent integers stored inside the class *FLOAT*. (ii) You must *NOT* convert the *FLOAT* numbers into C++ built-in floating-point types (like float or double) within any of your operator implementations. (iii) A *FLOAT* number N is internally represented as $N=\text{mantissa}\times 10^{\text{exponent}}$. For example, −0.56 is represented as mantissa=−56 and exponent=−2, while 10.234 is represented as mantissa=10234 and exponent=−3. Assume the addition of these numbers will be a *FLOAT* X, then: exponent(X) = min (exponent(-.56) , exponent(10.234)) mantissa (X) = mantissa(-.56) *$10^{|\text{ exponent (-.56) - exponent (10.234) }|}$ + mantissa (10.234) where, |D| represents the modulus of D. *You need to work out the rest of the operator logic yourself based on this understanding.*	(2+4 +3+4)
Ans	Marking Scheme: After summing all partial marks awarded across Sections A-D, The final total must be rounded UP to the nearest 0.5 mark. Sample program can be found at https://github.com/suchat-subho/OOPs_CodeReferences/blob/main/CPP/FLOAT_Operator.cpp 1. #include <iostream> 2. using namespace std; 3. 4.	

```
class FLOAT {
private:
    int mantissa;   // ✔ 0.25 mark: private integer data member
    int exponent;   // ✔ 0.25 mark: private integer data member
// (Total: 0.5 for data members)


    // Manual integer shifting: returns m * 10^n using loop
    int shift10(int m, int n) const {
        for (int i = 0; i < n; i++)
            m *= 10;
        return m;
    }


public:
    // Constructor
    FLOAT(int m = 0, int e = 0) : mantissa(m), exponent(e) {}
// ✔ 0.5 mark: parameterized constructor with default arguments; 0 otherwise


    // Getters
    int getMantissa() const { return mantissa; }   // ✔ 0.25 mark
    int getExponent() const { return exponent; }   // ✔ 0.25 mark
// (Total: 0.5 for getters); 0 void return in getter


    // Setters
    void setMantissa(int m) { mantissa = m; }      // ✔ 0.25 mark
    void setExponent(int e) { exponent = e; }      // ✔ 0.25 mark
// (Total: 0.5 for setters)


    // ========== FLOAT + FLOAT ==========
    FLOAT operator+(const FLOAT &b) const {
        int eX = (exponent < b.exponent) ? exponent : b.exponent;
        // ✔ 0.25 mark: exponent alignment logic

        int shiftA = exponent - eX;
        int shiftB = b.exponent - eX;
       //// Please note: The exponent alignment has to be performed before
       mantissa operation. Mere use of formula may not result in accurate code
       e.g (Fails when A.exponent < B.exponent). Test counter example:
       A=5×10^-3,B=2×10^-1 = 0.205
        int mA = mantissa;
        int mB = b.mantissa;

        if (shiftA > 0) mA = shift10(mA, shiftA);
        if (shiftB > 0) mB = shift10(mB, shiftB);
        // ✔ 0.5 mark: integer-based mantissa shifting (no float conversion)

        int mX = mA + mB;   // ✔ 0.25 mark: correct mantissa addition

        return FLOAT(mX, eX);
    }
    // ✔ Total: 1 mark for operator+; 0.5 if exponent not aligned; 0 otherwise


    // ========== FLOAT * FLOAT ==========
    FLOAT operator*(const FLOAT &b) const {
        int mX = mantissa * b.mantissa;   // ✔ 0.5 mark
        int eX = exponent + b.exponent;   // ✔ 0.5 mark
        return FLOAT(mX, eX);
    }
    // ✔ Total: 1 / 1 mark for operator*


    // ========== Comparison (integer-based) ==========
    bool operator==(const FLOAT &b) const {
       //// Please note: The exponent alignment has to be performed before
       mantissa operation. Mere use of formula may not result in accurate code.
        int eX = (exponent < b.exponent) ? exponent : b.exponent;

        int shiftA = exponent - eX;
        int shiftB = b.exponent - eX;
```

```
74.
75.         int mA = mantissa;
76.         int mB = b.mantissa;
77.
78.         if (shiftA > 0) mA = shift10(mA, shiftA);
79.         if (shiftB > 0) mB = shift10(mB, shiftB);
80.
81.         return (mA == mB);   // ✔ 1 mark: integer-based comparison
82.     }
83.     // ✔ Total: 1 / 1 mark; 0/1 if incorrect
84.
85.
86.     // ========== Assignment ==========
87.     FLOAT & operator=(const FLOAT &b) {
88.         if (this != &b) {   // ✔ 0.5 mark: self-assignment check
89.             mantissa = b.mantissa;
90.             exponent = b.exponent;
91.         }
92.         return *this;       // ✔ 0.5 mark: returns reference
93.     }
94.     // ✔ Total: 1/1 mark, 0.5/1 if Not appropriate return type (e.g. FLOAT &)
95.
96.
97.     // FRIENDS for FLOAT op int
98.     friend FLOAT operator+(const FLOAT &f, int x);   // ✔ 0.5 mark
99.     friend FLOAT operator*(const FLOAT &f, int x);   // ✔ 0.5 mark
100.    // (Total: 1 for Section C); .5 for wrong argument/ret types but correct
    friend syntax.
101.
102.
103.        void print() const {
104.            cout << mantissa << "e" << exponent;
105.        }
106.    };
107.
108.
109.    // FLOAT + int
110.    FLOAT operator+(const FLOAT &f, int x) {
111.        FLOAT temp(x, 0);       // ✔ 0.5 mark: int converted to FLOAT
112.        return f + temp;        // ✔ 0.5 mark: reuse FLOAT + FLOAT logic
113.    }
114.    // ✔ Total: 1 / 1 mark; 0/1 if incorrect
115.
116.
117.    // FLOAT * int
118.    FLOAT operator*(const FLOAT &f, int x) {
119.        return FLOAT(f.mantissa * x, f.exponent);   // ✔ 1 mark
120.    }
121.    // ✔ Total: 1 / 1 mark; 0/1 if incorrect
122.
123.    // ====================== MAIN ======================
124.    int main() {
125.        FLOAT A(10234, -3);   // 10.234
126.        FLOAT B(-56, -2);     // -0.56
127.
128.        cout << "A = "; A.print(); cout << endl;
129.        cout << "B = "; B.print(); cout << endl;
130.
131.        FLOAT C = A + B;      // ✔ 0.5 mark
132.        FLOAT D = A * B;      // ✔ 0.5 mark
133.        // (Total: 1 for FLOAT-FLOAT arithmetic)
134.
135.        cout << "\n\n(A == B)?  " << (A == B ? "YES" : "NO");
136.        // ✔ 1 mark: comparison demonstration
137.
138.        FLOAT E;
139.        E = A;                // ✔ 1 mark: assignment operator demo
140.
141.        FLOAT F = A + 5;      // ✔ 0.5 mark
142.        FLOAT G = B * 3;      // ✔ 0.5 mark
143.        // (Total: 1 for FLOAT-int arithmetic)
144.
145.        cout << endl;
146.        return 0;
147.    }
```