Assignment:-2

```
// 1. Class with public and private members
#include <iostream>
using namespace std;
class Sample {
private:
    int privateVar;
public:
    int publicVar;
    Sample() {
        privateVar = 10;
        publicVar = 20;
    }
    void printValues() {
        cout << "Private: " << privateVar << ", Public: " << publicVar << endl;
    }
};
int main() {
    Sample obj;
    obj.printValues();
    return 0;
}

// 2. Getter and Setter
#include <iostream>
using namespace std;
class MyClass {
private:
    int value;
public:
    void setValue(int v) { value = v; }
    int getValue() { return value; }
};
int main() {
    MyClass obj;
    obj.setValue(42);
    cout << "Value: " << obj.getValue() << endl;
    return 0;
}

// 3. Enum for weekdays
#include <iostream>
using namespace std;
enum Weekday { Monday, Tuesday, Wednesday, Thursday, Friday };
int main() {
    Weekday day = Wednesday;
    cout << "Wednesday value: " << day << endl;
    return 0;
}

// 4. Constructor initialization
#include <iostream>
using namespace std;
class Init {
public:
    Init(int val) {
        cout << "Initialized with value: " << val << endl;
    }
};
int main() {
    Init obj(100);
    return 0;
}
```

```
// 5. Destructor message
#include <iostream>
using namespace std;
class Demo {
public:
    ~Demo() {
        cout << "Destructor called!" << endl;
    }
};
int main() {
    Demo d;
    return 0;
}

// 6. Access modifiers and inheritance
#include <iostream>
using namespace std;
class Base {
public:
    int pub = 1;
protected:
    int prot = 2;
private:
    int priv = 3;
};
class Derived : public Base {
public:
    void show() {
        cout << "Public: " << pub << ", Protected: " << prot << endl;
    }
};
int main() {
    Derived d;
    d.show();
    return 0;
}

// 7. Data hiding, abstraction, encapsulation - ATM example
#include <iostream>
using namespace std;
class ATM {
private:
    int balance;
public:
    ATM(int bal) { balance = bal; }
    void withdraw(int amount) {
        if (amount <= balance) {
            balance -= amount;
            cout << "Withdrawn: " << amount << ", Remaining: " << balance <<
endl;
        } else {
            cout << "Insufficient balance." << endl;
        }
    }
};
int main() {
    ATM atm(1000);
    atm.withdraw(300);
    return 0;
}

// 8. Enum with color selection
#include <iostream>
```

```
using namespace std;
enum Color { Red, Green, Blue };
int main() {
    Color c = Green;
    cout << "Selected color value: " << c << endl;
    return 0;
}

// 9. Single-level inheritance
#include <iostream>
using namespace std;
class Animal {
public:
    void eat() { cout << "Eating..." << endl; }
};
class Dog : public Animal {
public:
    void bark() { cout << "Barking..." << endl; }
};
int main() {
    Dog d;
    d.eat();
    d.bark();
    return 0;
}

// 10. Multi-level inheritance
#include <iostream>
using namespace std;
class Grandparent {
public:
    void showG() { cout << "Grandparent" << endl; }
};
class Parent : public Grandparent {
public:
    void showP() { cout << "Parent" << endl; }
};
class Child : public Parent {
public:
    void showC() { cout << "Child" << endl; }
};
int main() {
    Child c;
    c.showG();
    c.showP();
    c.showC();
    return 0;
}

// 11. Multiple inheritance
#include <iostream>
using namespace std;
class Person {
public:
    string name = "John";
};
class Marks {
public:
    int score = 95;
};
class Student : public Person, public Marks {
public:
    void show() {
        cout << "Name: " << name << ", Score: " << score << endl;
```

```
    }
};
int main() {
    Student s;
    s.show();
    return 0;
}

// 12. Constructor overloading
#include <iostream>
using namespace std;
class Overload {
public:
    Overload() { cout << "Default constructor" << endl; }
    Overload(int x) { cout << "Parameterized: " << x << endl; }
};
int main() {
    Overload o1;
    Overload o2(10);
    return 0;
}

// 13. Copy constructor
#include <iostream>
using namespace std;
class CopyDemo {
public:
    int val;
    CopyDemo(int v) { val = v; }
    CopyDemo(const CopyDemo &obj) { val = obj.val; }
    void show() { cout << "Value: " << val << endl; }
};
int main() {
    CopyDemo a(5);
    CopyDemo b = a;
    b.show();
    return 0;
}

// 14. Destructors with multiple objects
#include <iostream>
using namespace std;
class Multi {
public:
    Multi() { cout << "Object created" << endl; }
    ~Multi() { cout << "Object destroyed" << endl; }
};
int main() {
    Multi a, b;
    return 0;
}
```