// **Question 9**

```
//    A class called “Shakib Al Hasan” has two
// operator overloading functions.Now implement a code
// for the statement :
// TanzidTamim1 = (LitonKumarDas2 + 2) * TaskinAhmed3
// to compile correctly.


#include<bits/stdc++.h>
using namespace std;
class Shakib_All_Hasan
{
public:
    int a;
    Shakib_All_Hasan() {}
    Shakib_All_Hasan(int x) {
        a = x;
    }
    Shakib_All_Hasan operator + (int n) {  //Operator
overloading(LitonKumarDas2 +2)
        a = a + n;
        return *this;
    }
    Shakib_All_Hasan operator *(Shakib_All_Hasan obj) {//Operator
overloading(LitonKumarDas2 +2)*TaskinAhamed3
        a = a * obj.a;
        return *this;
    }
    void display() {
        cout << a << endl;
    }
};
int main() {
    Shakib_All_Hasan TanzidTamim1, LitonKumarDas2(5), TaskinAhamed3(4);
    TanzidTamim1 = (LitonKumarDas2 + 2) * TaskinAhamed3;
    TanzidTamim1.display();
    return 0;
}
```

}

//Question 8

```
// Design a class named "Eric ten Hag".Eric ten Hag will have
// only one function showDegea() which will output what it saves.
// Create two derived classes, Varane and Maguire.Varane will
// define showDegea() so that it will output “saves goal” and similarly
// Maguire will output “saves nogoals”.In the main() function,
// use the Eric ten Hag class in a way that implements
// the idea of abstraction.

#include <bits/stdc++.h>
using namespace std;

class Eric_ten_Hag{
public:
    virtual void ShowDegea() = 0; // pure virtual function / nothing function
};

class Varane : public Eric_ten_Hag{
public:
    void ShowDegea(){
        cout << "Save goals" << endl;
    }
};

class Maguire : public Eric_ten_Hag{
public:
    void ShowDegea(){
        cout << "Save no goals" << endl;
    }
};

int main(){
    Varane ob1;
    ob1.ShowDegea();
    Maguire ob2;
    ob2.ShowDegea();

    return 0;
}
```

//Question 1

```
//1.    Imagine a publishing company that markets both book and audiocassette
versions
//of its works. Create a class publication that stores the title (a string) and
//price (type float) of a publication. From this class derive two classes: book,
//which adds a page count (type int), and tape, which adds a playing time in
//minutes (type float). Each of these three classes should have a getData()
function
//to get its data from the user at the keyboard, and a putData() function to
display its data.
// Write a main program to test the book and tape classes by creating instances
of them,
 //asking the user to fill in data with getData(), and then displaying the data with
putData().


#include <bits/stdc++.h>

using namespace std;

class publication {
public:
    string title;
    float price;
    virtual void getData() = 0;
    virtual void putData() = 0;
};

class book : public publication {
public:
    int page;
    void getData() {
        cout << "Enter Name of the book ";
        cin >> title;
        cout << "Enter Price of the book ";
        cin >> price;
        cout << "Enter Page count of the book ";
        cin >> page;
    }
    void putData() {
        cout << "Title = " << title << endl;
        cout << "Price = " << price << endl;
        cout << "Total Page = " << page << endl;
    }
};

class tape : public publication {
public:
    float time;
    void getData() {
        cout << "Enter Name of the tape ";
        cin >> title;
        cout << "Enter Price of the tape ";
        cin >> price;
        cout << "Enter Playing time of the tape ";
        cin >> time;
    }
    void putData() {
        cout << "Title = " << title << endl;
        cout << "Price = " << price << endl;
        cout << "Total Playing Time = " << time << endl;
    }
};

int main() {
    book ob1;
    ob1.getData();
    ob1.putData();
    tape ob2;
    ob2.getData();
    ob2.putData();
    return 0;
}
```

//Question 2

//2.   Manchester United is a class with two private integer member variables
//coach and player, and a public void member function getData (). Create an
//object named 'ronaldo' in the main function. Overload the operator '++' in
// this class to perform the increment of both member variables through the
//following instruction ronaldo++ from the main function. After that create
//another instance of the Manchester United class named 'fernandes'. Now,
// you set the values of coach and player for ronaldo to 4 and 5, and for
// fernandes, it is 5 and 6. Overloading only one relational operator,
//compare the result of ronaldo with fernandes before and after
//incrementing ronaldo by one. Which operator will be appropriate for both
//cases? Support your explanation by implementing that operator.

```
#include <bits/stdc++.h>

using namespace std;

class Manchester_United {
private:
    int coach, player;
public:
    void getData(int c, int p) {
        coach = c;
        player = p;
    }
    Manchester_United operator ++ (int) {
        coach = coach + 1;
        player = player + 1;
        return *this;
    }
    bool operator == (Manchester_United ob) {
        if ((coach == ob.coach) and (player == ob.player))
            return true;
        return false;
    }
};

int main() {
    Manchester_United ronaldo, fernandes;
    ronaldo.getData(4, 5);
    fernandes.getData(5, 6);
    if (ronaldo == fernandes) {
        cout << "Ronaldo and Fernandes is equal" << endl;
    }
    else {
        cout << "Ronaldo and Fernandes is not equal" << endl;
    }
    ronaldo++;
    if (ronaldo == fernandes) {
        cout << "Ronaldo and Fernandes is equal" << endl;
    }
    else {
        cout << "Ronaldo and Fernandes is not equal" << endl;9
    }

    return 0;
}
```

//Question 3

//3.   Suppose there are three classes named “Germany”, “Argentina” and “Brasil”.
// Build a diagram to show how these classes are related. What do you need to do
 //tohide the implementation details of the parent class obtained from the diagram?
 //Demonstrate the scenario with the help of an array of pointers.

```
#include <bits/stdc++.h>

using namespace std;

class ParentClass {
protected:
   string name;
public:
   virtual void display() = 0; // Pure virtual function
};

class Germany : public ParentClass {
public:
   Germany() { name = "Germany"; }
   void display() { cout << name << endl; }
};

class Argentina : public ParentClass {
public:
   Argentina() { name = "Argentina"; }
   void display() { cout << name << endl; }
};


class Brazil : public ParentClass {
public:
   Brazil() { name = "Brazil"; }
   void display() { cout << name << endl; }
};
```

```
int main() {
    ParentClass* countries[3];
    Germany g;
    Argentina a;
    Brazil b;
    countries[0] = &g;
    countries[1] = &a;
    countries[2] = &b;

    for (int i = 0; i < 3; i++) {
        countries[i]->display();
    }

    return 0;
}
```