

```

class Student
{
    int roll;
    char grade;
    float per;
public:
    Student(int,char,float);
    void get();
    void show();
};
Student::Student(int r,char g,float p)
{
    roll=r;
    grade=g;
    per=p;
}
void Student::get()
{
    cout<<"Enter roll,grade and per:";
    cin>>roll>>grade>>per;
}

void Student::show()
{
    cout<<roll<<" "<<grade<<" "<<per;
}

int main()
{
    Student S(10,'A',65.3);
    Student P;
    P.get();
    S.show();
    P.show();
    return 0;
}

```

enter! →

Solution 1: Use Constructor Overloading

```

class Student
{
    int roll;
    char grade;
    float per;
public:
    Student(int,char,float);
    void get();
    void show();
    Student();
};
Student::Student()
{
}
Student::Student(int r,char g,float p)
{
    roll=r;
    grade=g;
    per=p;
}

void Student::get()
{
    cout<<"Enter roll,grade and per:";
    cin>>roll>>grade>>per;
}

void Student::show()
{
    cout<<roll<<" "<<grade<<" "<<per<<endl;
}

int main()
{
    Student S(10,'A',65.3);
    Student P;
    P.get();
    S.show();
    P.show();
    return 0;
}

```

Solution 2: Use Default Parameterized Constructor

```
class Student
{
    int roll;
    char grade;
    float per;
public:
    Student(int=0,char=' ',float=0.0);
    void get();
    void show();
};

Student::Student(int r,char g,float p)
{
    roll=r;
    grade=g;
    per=p;
}

void Student::get()
{
    cout<<"Enter roll,grade and per:";
    cin>>roll>>grade>>per;
}
```

```
void Student::show()
{
    cout<<roll<<" "<<grade<<" "<<per<<endl;
}

int main()
{
    Student S(10,'A',65.3);
    Student P;
    P.get();
    S.show();
    P.show();
    return 0;
}
```

Destructor

```
int main()
{
    int a = 10;

    ① {
        int b = 20;

        cout<<b; → 20

        ② {
            cout<<a; → 10
        }

        return 0;
    }
```

```
int main()
{
    int a = 10;

    {
        int b = 20; Box B?

        cout<<a;

        ③ {
            cout<<b;
        }

        return 0;
    }
```

Destructor

=====

1. In C++ just like we have constructor, similarly we also have the concept of Destructor .
2. A destructor is another special member function of the class having same name as that of the class but prefixed with the symbol of Tiled(~).
3. This means that if the class name is Box, the name of the constructor will be **Box()** and the name of the destructor will be **~Box()**.
4. Now, whenever the C++ compiler will decide to destroy any object of the class then just before removing it from memory it will automatically call the destructor function present in the class.

5. So we can say that in C++ every object in its entire life time always automatically calls at least two member functions called

1. Constructor and
 2. Destructor. The constructor function is called immediately after the object gets created and the destructor function gets called just before the object is to be destroyed.
6. Moreover both these functions are implicitly called by the C++ compiler.
7. Also if in the class we don't declare any destructor ourselves then the C++ compiler automatically inserts a destructor in our class called as... default destructor having an empty body.

```

class Student
{
    int roll;
    char grade;
    float per;
public:
    Student();
    void show();
    ~Student();

};

Student::Student()
{
    cout<<"Constructor called..."<<endl;
    cout<<"Enter roll,grade and per:";
    cin>>roll>>grade>>per;
}

```

```

void Student::show()
{
    cout<<roll<<","<<grade<<","<<per<<endl;
}

Student::~~Student()
{
    cout<<"Destructor called..."<<endl;
}

int main()
{
    Student S;
    Student P;
    S.show();
    P.show();
    return 0;
}

```