Unit 3:
Fundamentals of Object Oriented Programming

CLASSES AND OBJECTS

Difference between Procedure oriented and Object Oriented Programming

Procedure Oriented Programming	Object Oriented Programming
❑Emphasis is on doing things (algorithms), so procedures (Functions) are important	❑ Emphasis is on using things (data), so data is important than functions
❑ Large programs are divided into smaller programs known as functions	❑ Programs are divided into objects rather than functions
❑ Most of the functions share global data, may be facing issue of Data Inconsistency	❑Data is hidden and cannot be accessed by unauthorised function (External function), maintain data consistency
❑ Function can communicate with each other through parameters ,so data can flow freely between functions	❑Objects communicate with each other so, data cannot move freely between all the functions
❑ Data security is not possible	❑Data security has utmost importance
❑ Uses top-down approach in the program design	❑Uses bottom –up approach in program design

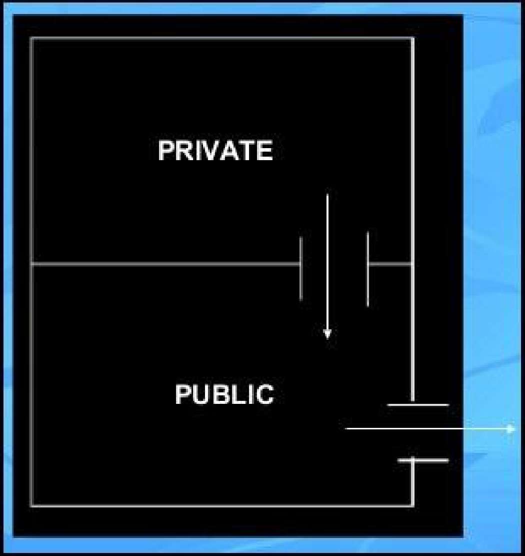
Data Encapsulation

What is Data Encapsulation ?

- ❑ Encapsulation is a process of combing data and function into single unit called class
- ❑ Data is only accessible through the functions present inside the class
- ❑ Hiding the implementation details from external entity, led data abstraction
- ❑ Encapsulation led to the important concept of data hiding
- ❑ Implemented by defining class
- ❑ Only public members are accessible for external entity like main() and non member functions

Example : -

```
// *******Definition of the class********//
class student      ← User defined data type
{
private:    // Access modifier
char name[20];        }
int total_Marks;      }  Data Members
float Percentage;     }
public:    // Access modifier
void readData()       }
calculate_Per()       }  Members Functions
void Printdata()      }
}s1,s2;   //Global declaration of objects
```



Access rights of class members:

Only public members are accessible for external functions

Private members are accessible with the help of public member function

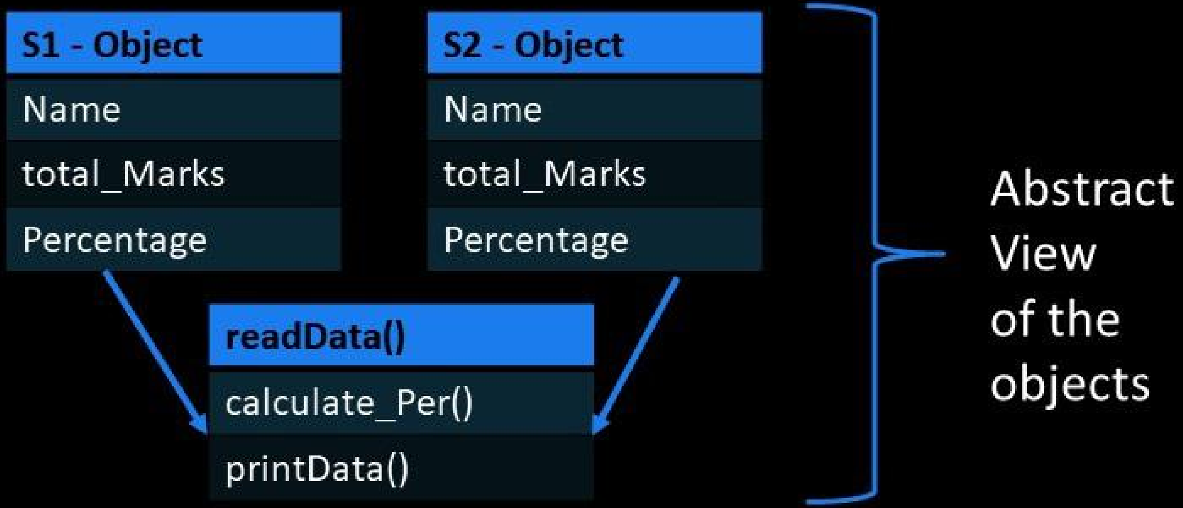
Data Abstraction

What is Data Abstraction ?

- ❑ Abstraction refers to the act of representing essential features without including the background details (class definition).
- ❑ Data abstraction is implemented by creating objects of the class definition
  - ❑ Can be declared global while defining class
  - ❑ Can be declared local within the other user defined function
- ❑ Objects are the basic run time entities in an object-oriented system

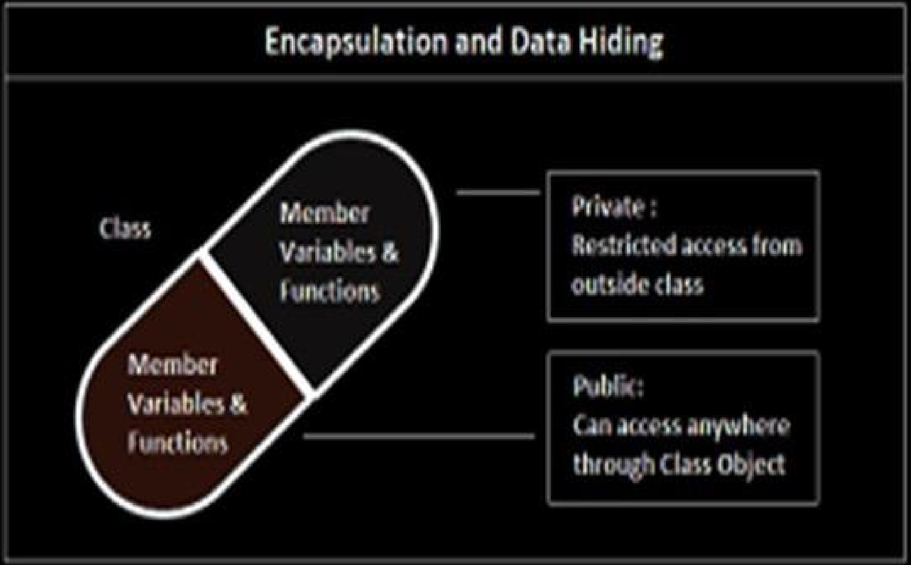
Example : -

```
//*******Declaration  of an object**********//

// Allocation of memory to student class definition//


        int main()
        {
        student s1, s2;       //declaration of local objects.
        }
        int xyz()
        {
        student s1,s2;      //declaration of local objects.
        }
```



Data Hiding

- Data hiding is a process of combining data and functions into a single unit using private access modifier.
- The ideology behind data hiding is to conceal data within a class, to prevent its direct access from outside the class. It helps programmers to create classes with unique data sets and functions, avoiding unnecessary penetration from other program/functions.
- Data hiding only hides class data components, whereas data encapsulation hides class data parts and private methods.
- Data hiding is achieved by private access modifier


Encapsulation and Data Hiding
Class
Member Variables & Functions
Member Variables & Functions
Private : Restricted access from outside class
Public: Can access anywhere through Class Object

Diagrammatic Representation

Access Modifiers

Private members/methods

- ❑Can only be accessed by methods defined as part of the class.
- ❑Data is most often defined as private to prevent direct outside access from other classes.
- ❑Private members can be accessed by members of the class.

Public members/methods

- ❑can be accessed from anywhere in the program.
- ❑Class methods are usually public which is used to manipulate the data present in the class.
- ❑As a general rule, data should not be declared public.
- ❑Public members can be accessed by members and objects of the class.

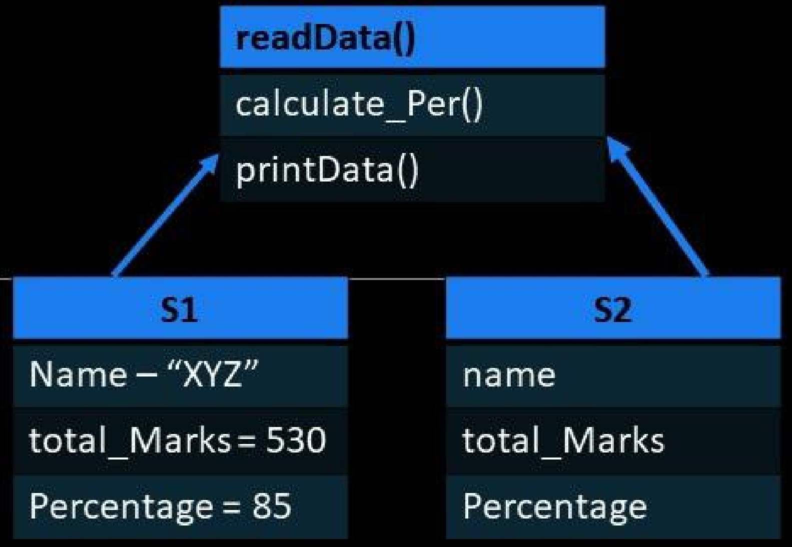
Data Hiding : Example

```
//****provide data security to data***//
class Student
{
char name[20];
int total_Marks;
float Percentage;

public:
void readData() {}
void calculate_Per() {}
void printData() {}
};
```

```
int main()
{
Student s1,s2;
s1.readData();

cin>>s1.name;
```

Not allowed in main function (data hiding)

Data access in main through member functions only



Objects Accessing data using member function

Classes: Defining a Class With a Member Function

Class definition:

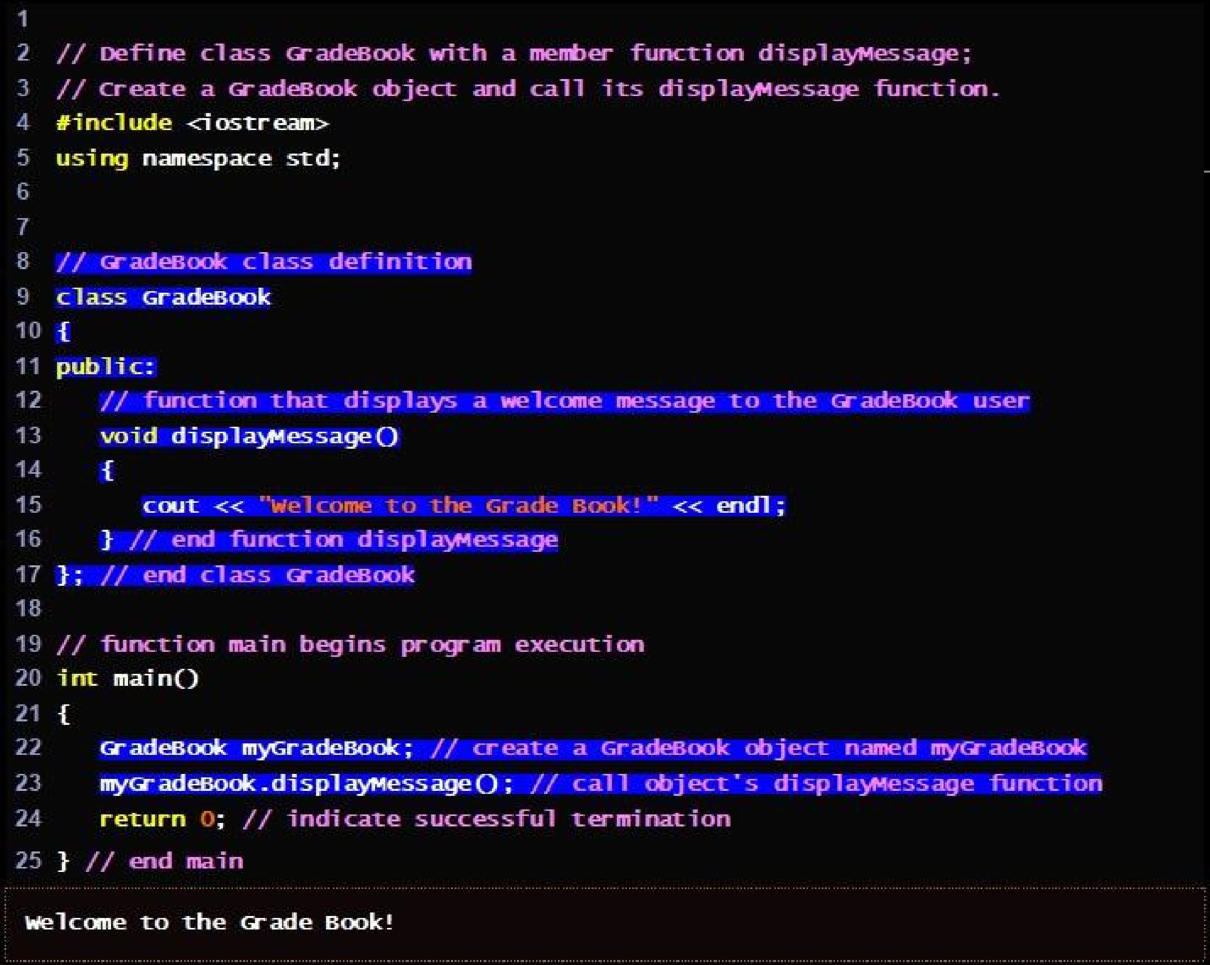
- Tells the compiler what member functions and data members belong to the class.
- Keyword class followed by the class's name.
- Class body is enclosed in braces ({})
  - Specifies data members and member functions
  - Access-specifier **private:**
    - Indicates that a member function or data member is accessible to only member functions of the same class.
  - Access-specifier **public:**
    - Indicates that a member function or data member is accessible to other functions and member functions of other classes.

Example 1: class definition: student

```
// *******Definition  of the class********//
class student
{
private:
char name[20];
int total_Marks;
float Percentage;
public:    // Access modifier (specifiers)
void readData()
calculate_Per()
void Printdata()
}s1,s2;   // Memory of s1 and s2 will be created
```

```
// *******Definition of the class********//
class firstObj
{
private:
 int x;
 int y;
public:    // Access modifier (specifiers)
void readData(int a, int b) { x= a; y=b;}
void PrintAdd() { int c=x+y; cout<<c; }
};
```

Example 2: class definition: Gradebook

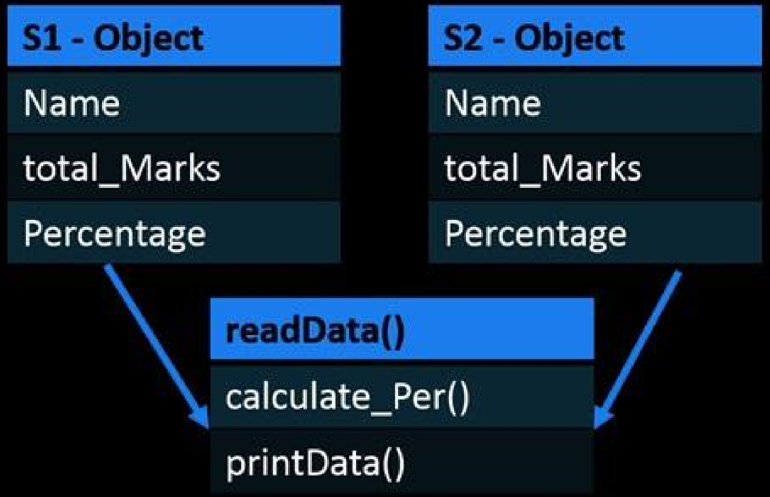


```cpp

// Define class GradeBook with a member function displayMessage;
// Create a GradeBook object and call its displayMessage function.
#include <iostream>
using namespace std;


// GradeBook class definition
class GradeBook
{
public:
   // function that displays a welcome message to the GradeBook user
   void displayMessage()
   {
      cout << "Welcome to the Grade Book!" << endl;
   } // end function displayMessage
}; // end class GradeBook

// function main begins program execution
int main()
{
   GradeBook myGradeBook; // create a GradeBook object named myGradeBook
   myGradeBook.displayMessage(); // call object's displayMessage function
   return 0; // indicate successful termination
} // end main
```

```
Welcome to the Grade Book!
```

Objects:

- ❑ Objects are the basic run time entities in an object-oriented system
- ❑ Member objects constructed in the order they declared
- ❑ Each object has its own copy of data
  - May be objects of any type
- ❑ Member functions are operated on calling object (current)
- ❑ Each object share same copy of functions
- ❑ Initially object have no values while declaring:
  - ❑ They need to initialize by using following
    - ❑ Member Functions
    - ❑ Constructor

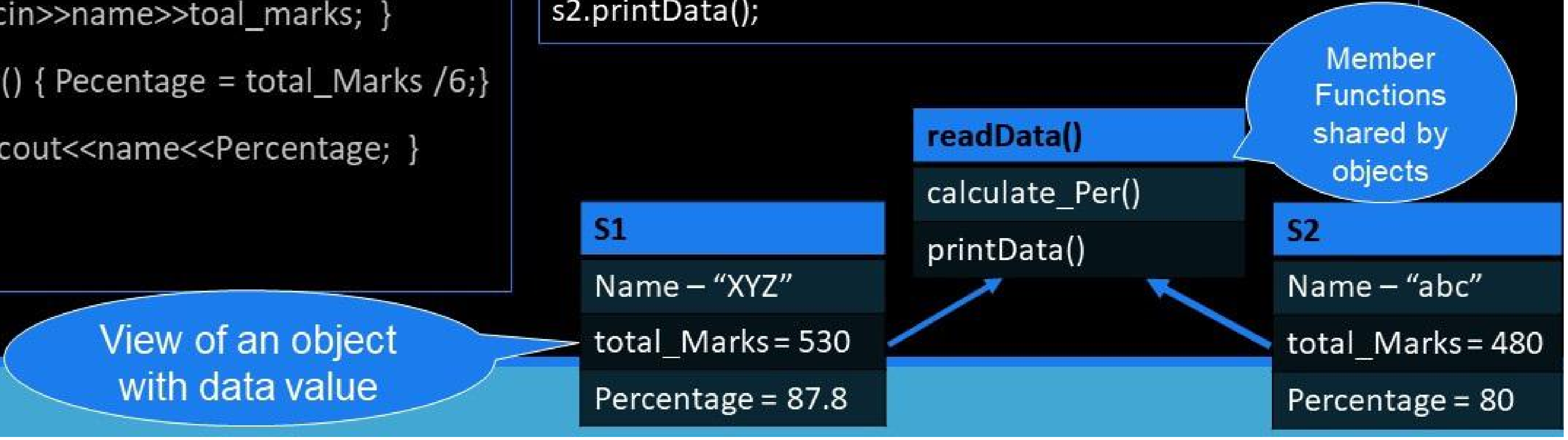


View of an objects of class shown in Example 1

Object Initialization: - using member function

```
//********provide  security to data**********//

class Student

{

char name[20];

int total_Marks;

float Percentage;

public:

void readData() { cin>>name>>toal_marks;  }

void calculate_Per() { Pecentage = total_Marks /6;}

void printData() { cout<<name<<Percentage;  }

};
```

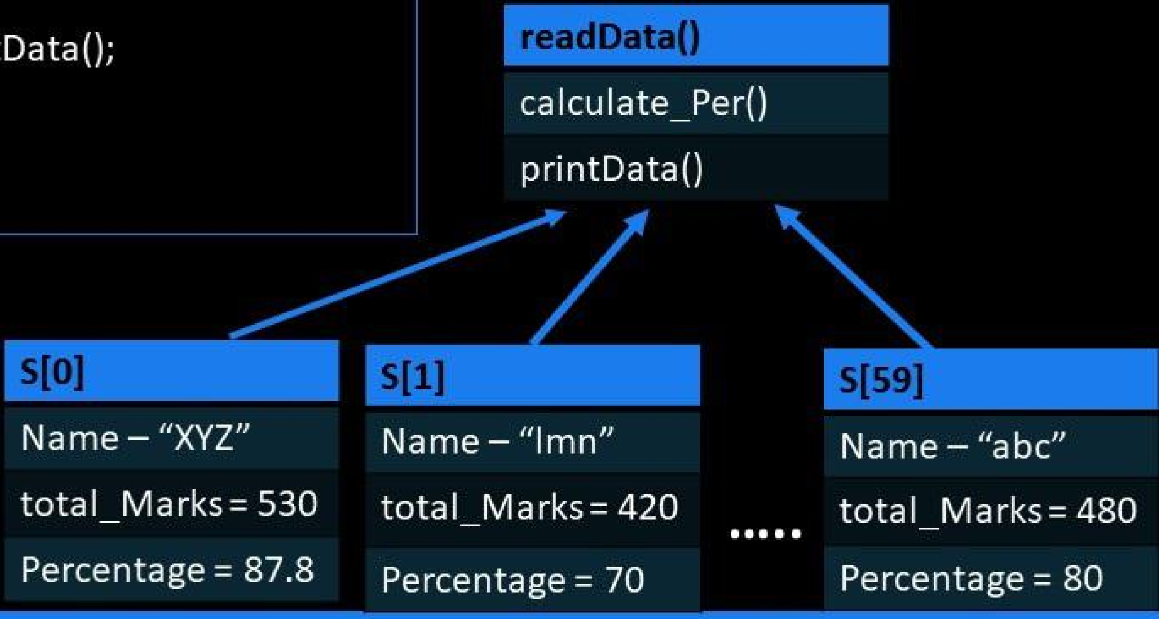
```
int main()
{
Student s1,s2;  // Garbage values in objects
s1.readData();  // Initialize objects with members function
s2.readData();
s1.calculate_per(); // Apply function with an object and
                             calculate percentage

s2.calculate_per();
s1.printData();     // Apply function with object and
                             print data of an object
s2.printData();
```



Array of Objects

```
//********provide  security to
data*****************//

class Student

{

char name[20];

int total_Marks;

float Percentage;


public:

void readData() { cin>>name>>toal_marks  }

void calculate_Per()

{ Pecentage = total_Marks /6;}

void printData() {cout<<name<<percentage;}

};
```

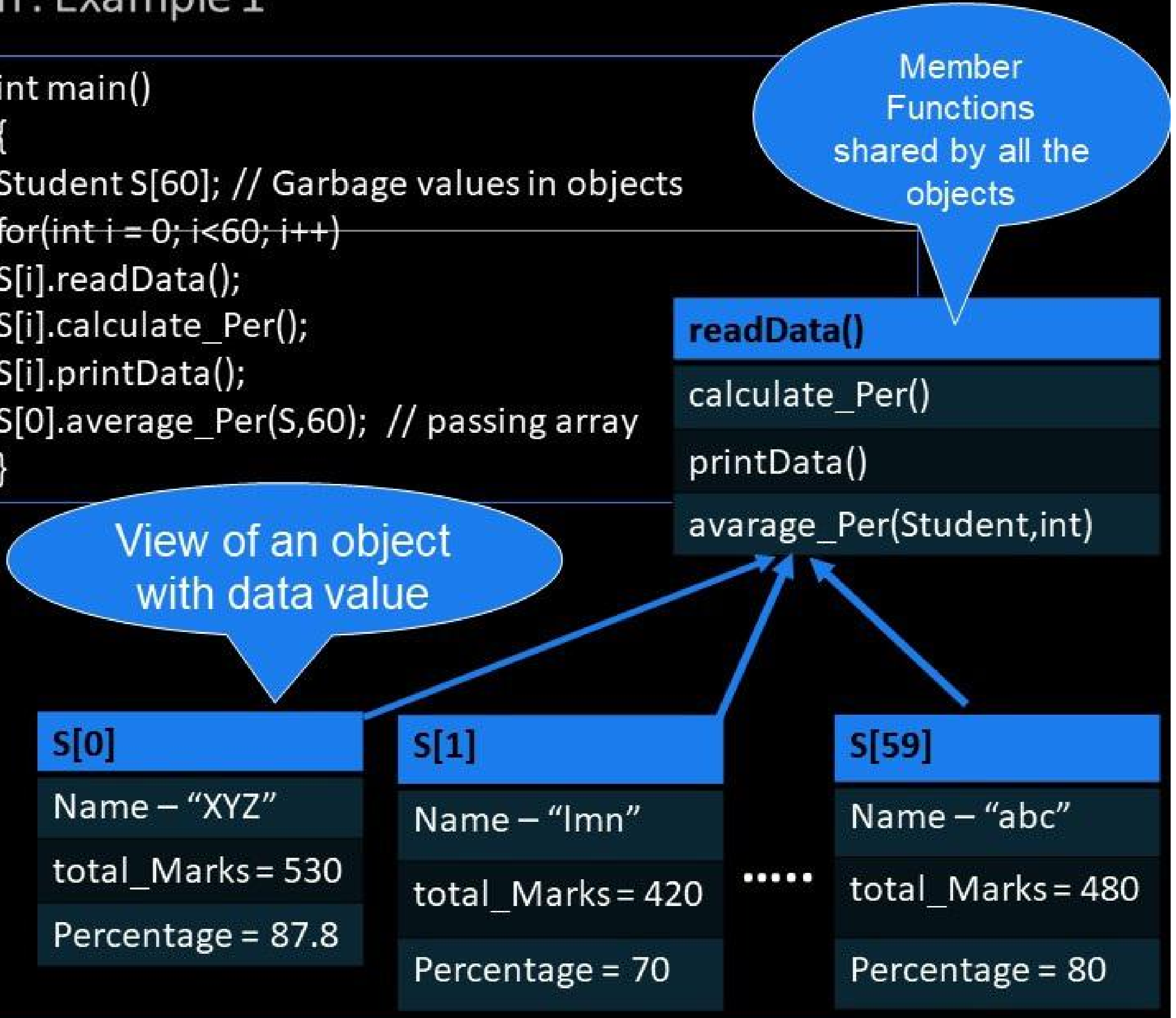
```
int main()
{
Student S[60]; // Random values
for(int i = 0; i<60; i++)
{
S[i].readData();
S[i].calculate_per();
}
for(int i = 0; i<60; i++)
{
S[i].printData();
}
}
```

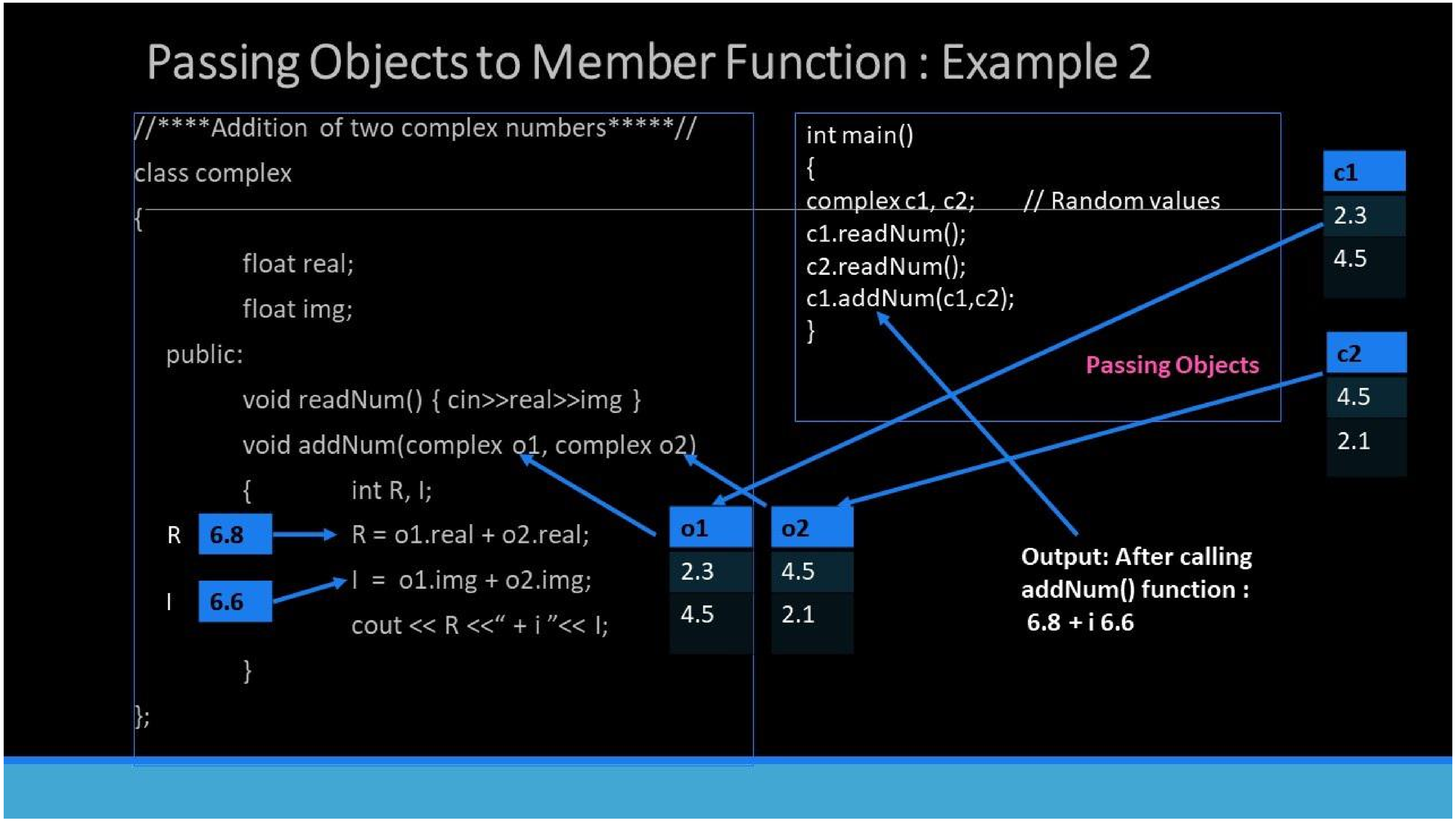


Passing array of Objects to Member Function : Example 1

```
//********provide security to data*****************//
class Student
{
char name[20];
int total_Marks;
float Percentage;
public:
        void readData() { cin>>name>>toal_marks; }
        void calculate_Per() { Pecentage = total_Marks /6;}
        void printData() {cout<<name<<percentage;}
        void average_Per(Student S[], int n)
        {
        float sum=0, Avg_Per;
        for(int i = 0;i<n;i++)
        sum=sum+ S[i[.percentage;
        Avg_Per=sum/n;
        Cout<<Avg_Per;
        }
};
```

```
int main()
{
Student S[60]; // Garbage values in objects
for(int i = 0; i<60; i++)
S[i].readData();
S[i].calculate_Per();
S[i].printData();
S[0].average_Per(S,60);  // passing array
}
```





Passing Objects to Member Function : Example 2

```
//****Addition  of two complex numbers*****//
class complex
{
        float real;
        float img;
  public:
        void readNum() { cin>>real>>img }
        void addNum(complex o1, complex o2)
        {         int R, I;
                  R = o1.real + o2.real;
                  I  =  o1.img + o2.img;
                  cout << R <<“ + i ”<< I;
        }
};
```

```
int main()
{
complex c1, c2;        // Random values
c1.readNum();
c2.readNum();
c1.addNum(c1,c2);
}
```

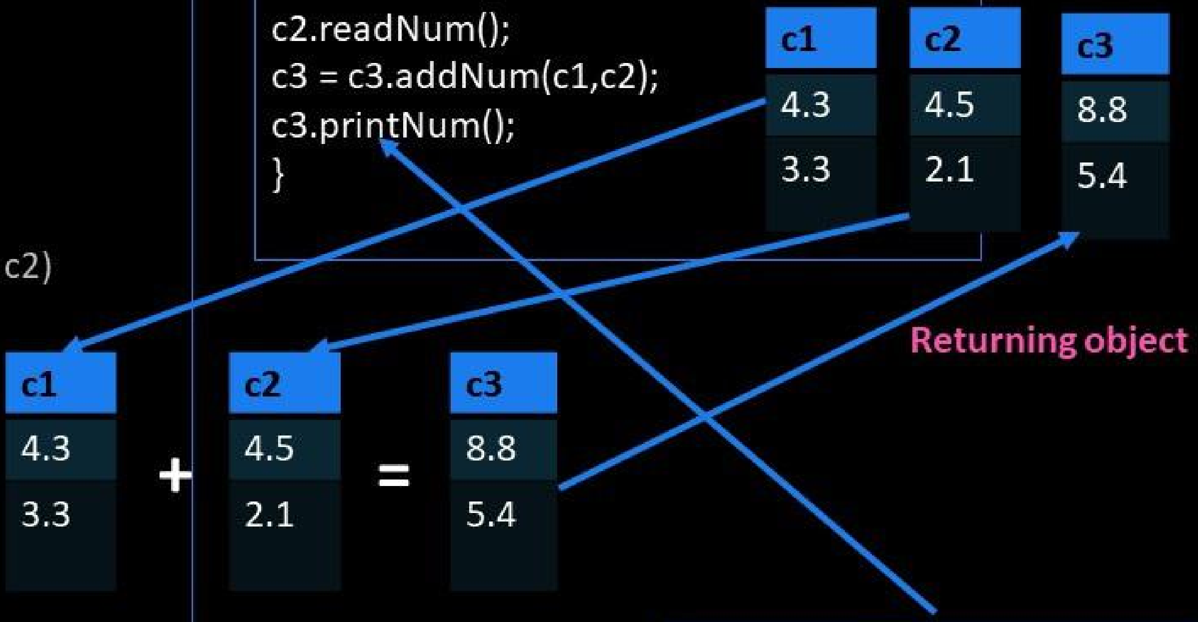
R
6.8
I
6.6
o1
2.3
4.5
o2
4.5
2.1
c1
2.3
4.5
c2
4.5
2.1
Passing Objects

Output: After calling addNum() function :
6.8 + i 6.6

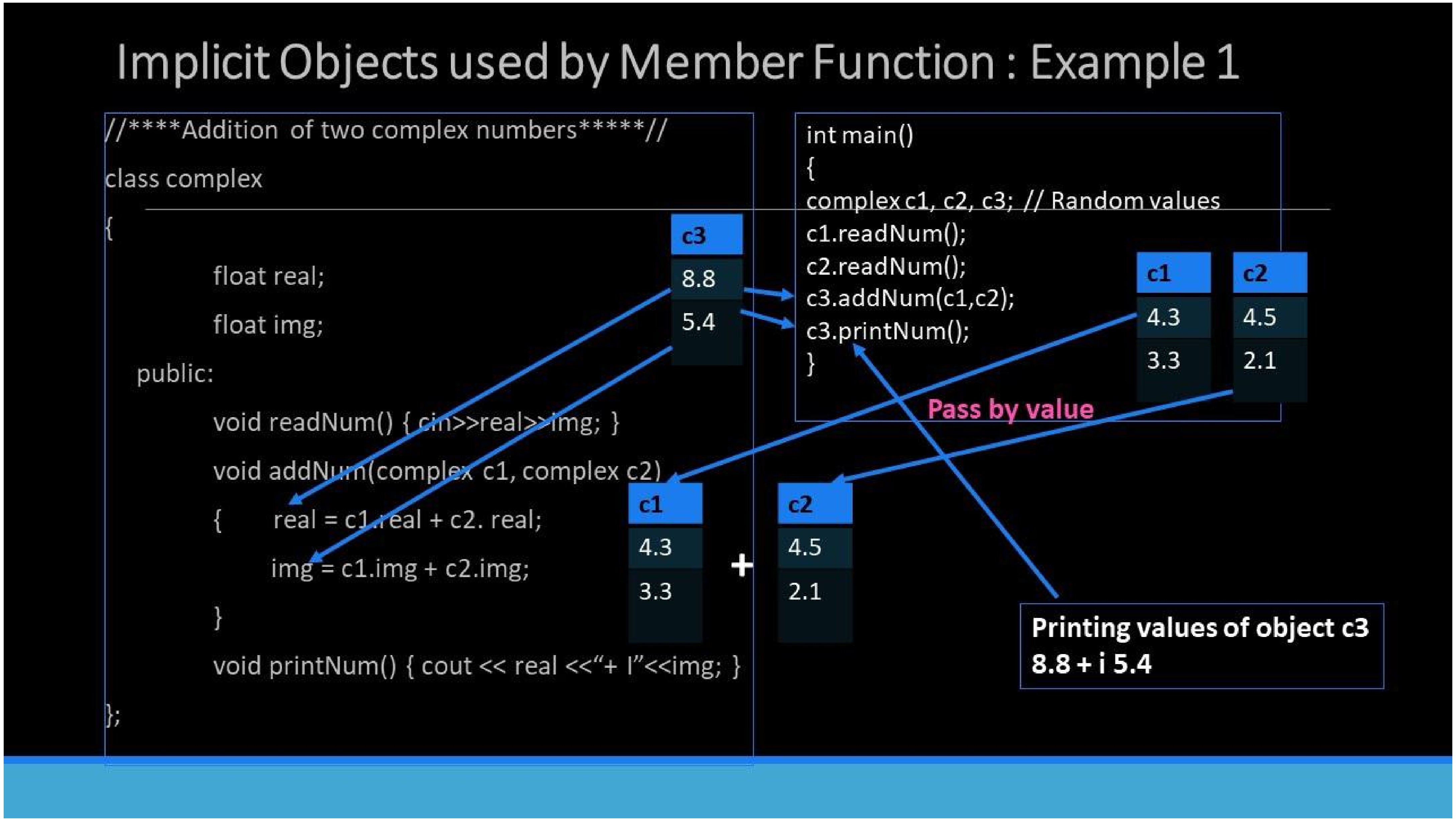
Returning an Objects (Explicit) by Member Function : Example 1

```
//****Addition of two complex numbers*****//
class complex
{
            int real;
            int img;
    public:
            void readNum() { cin>>real>>img; }
            complex addNum(complex c1, complex c2)
            {           complex c3;
                        c3.real = c1.real + c2. real;
                        c3.img = c1.img + c2.img;
                        Return c3;
            }
            void printNum() { cout << real <<"+ i"<<img; }
};
```

```
int main()
{
complex c1, c2, c3;  // Random values
c1.readNum();
c2.readNum();
c3 = c3.addNum(c1,c2);
c3.printNum();
}
```



Printing values of object c3
8.8 + i 5.4



Implicit Objects used by Member Function : Example 1

```
//****Addition  of two complex numbers*****//
class complex
{
          float real;
          float img;
   public:
          void readNum() { cin>>real>>img; }
          void addNum(complex  c1, complex c2)
          {      real = c1.real + c2. real;
                 img = c1.img + c2.img;
          }
          void printNum() { cout << real <<“+ I”<<img; }
};
```

```
int main()
{
complex c1, c2, c3;  // Random values
c1.readNum();
c2.readNum();
c3.addNum(c1,c2);
c3.printNum();
}
```

Printing values of object c3
8.8 + i 5.4

Practice program 1:

Problem Statement: Generate Merit List for admission in MPSTME. Write a program to read the data of students with attributes such as Name, Percentage, meritNo. Use functions to read data, sort student list, generate merit list and print merit list.

Instructions: basic structure of the program is given in the next slide, You have to write your own function for generating merit number and sorting the list of students as per percentage.

Statement - 1

	student [0]	student [1]	student [2]	student [3]	student [999]
name	abc	xyz	sasas	dsds	gsgs
per	40	80.5	50.89	56.8	89.5
meritNo					

```
class admission
{
 char name[20];
  float per;
  int meritNo;

  public:
  void readData(); { }
  void printMerit(); { }
  void sorting(admission s[], int n)
  {
  }
  void assignMerit(admission s[], int n)
  {
  }
  };
```

```
int main()
{
admission student[1000];
for
{
student[i].readData();
}
student[0].sorting(student,1000);
Student[0].assignMerit(student,1000);
for
{
student[i].printMerit();
}
}
```

Statement – 1:

	student [0]	student [1]	student [2]	student [3]		student [999]
name	Aman	Prateek	Sudarshan	xyx		abc
per	89.5	87.6	87.0	86.8	• • •	70.0
meritNo						

```
class admission
{
  char name[20];
  float per;
  int meritNo;

  public:
  void readData(); { }
  void printMerit(); { }
  void sorting(admission s[], int n)
  {
  }
  void assignMerit(admission s[], int n)
  {
  }
  };
```

```
int main()
{
admission student[1000];
for
{
student[i].readData();
}
student[0].sorting(student,1000);
Student[0].assignMerit(student,1000);
for
{
student[i].printMerit();
}
}
```

Array of objects with data values (sorted list)

Practice program 2:

Statement 2: Write a program to maintain record of car service centre and categorise as per service status and print the list separately.

Data members : Make, Model, Car Number, Service status. (Completed, In process, Waiting, Delivered)

Member Function : readData (), assignStatus(), displayStatus().

Instructions: basic structure of the program is given in the next slide, You have to write your own function to change the status of the service and continue after asking choice : y or n.

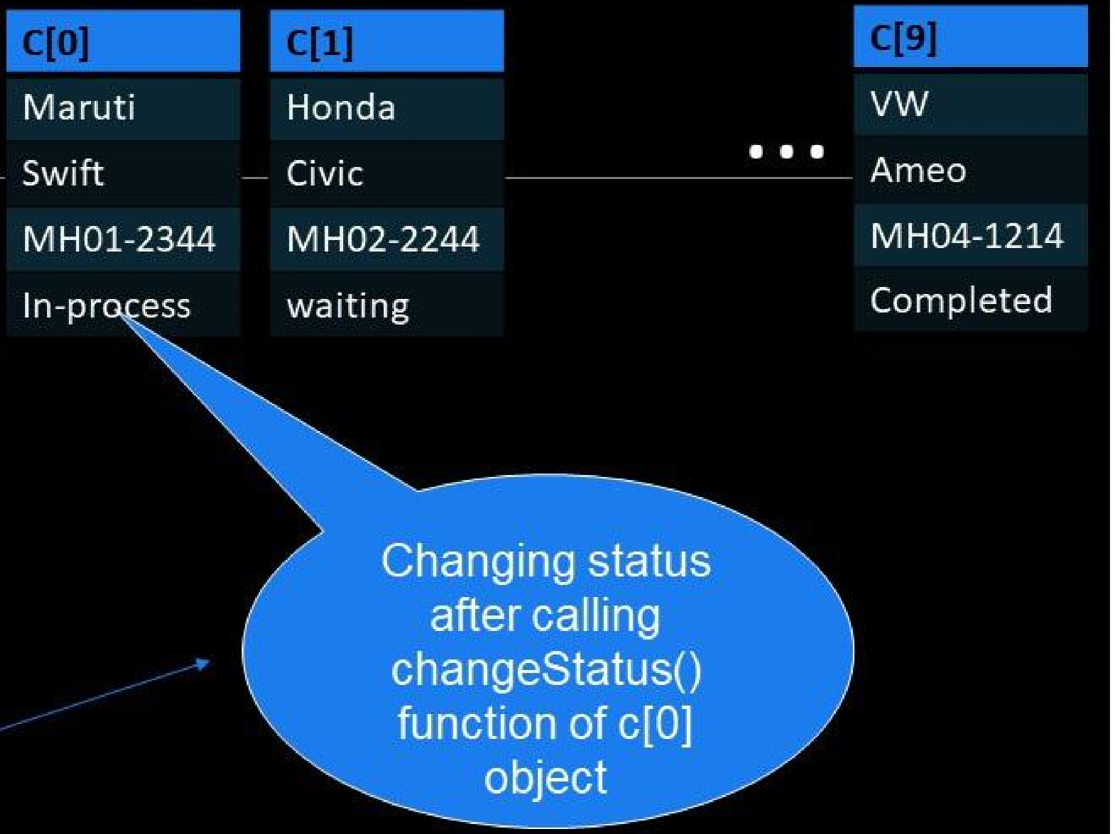
Initialization of objects from main function

```
class Car_Service  {

char make[20];  char model[20];

char no[20];  char status[20];

public:

void getData(char *make1, char *model1, char *no1, char *status1)

{

strcpy(make,make1);      // cin>>make;

strcpy(model,model1);  // cin>>model;

strcpy(no,no1);              //cin>>no;

strcpy(status,status1);   //cin>>status;

}
```

```
void printStatus()

{

cout<<"Make  is :"<<make<<endl;

cout<<"Model  is :"<<model<<endl;

cout<<"no  is :"<<no<<endl;

cout<<"status  is :"<<status<<endl;

}

void changeStatus(char  *no1, char *status1)

{

}

};
```

main function:

```
int main()
{
Car_Service  c[10];
char make1[20], model1[20], no1[10], status1[20], st[20];
char choice;
for (int i=0;i<5;i++)
{
cout<<" make , mpdel , no and status :"<<endl;
cin>>make1>>model1>>no1>>status1;
c[i].getData (make1,model1,no1,status1);
}
C[0].changeStatus(char  *no1, char *st);
for (int i=0;i<5;i++)   {  c[i].printStatus();   }
}
```


C[0]
Maruti
Swift
MH01-2344
In-process
C[1]
Honda
Civic
MH02-2244
waiting
C[9]
VW
Ameo
MH04-1214
Completed
Changing status after calling changeStatus() function of c[0] object