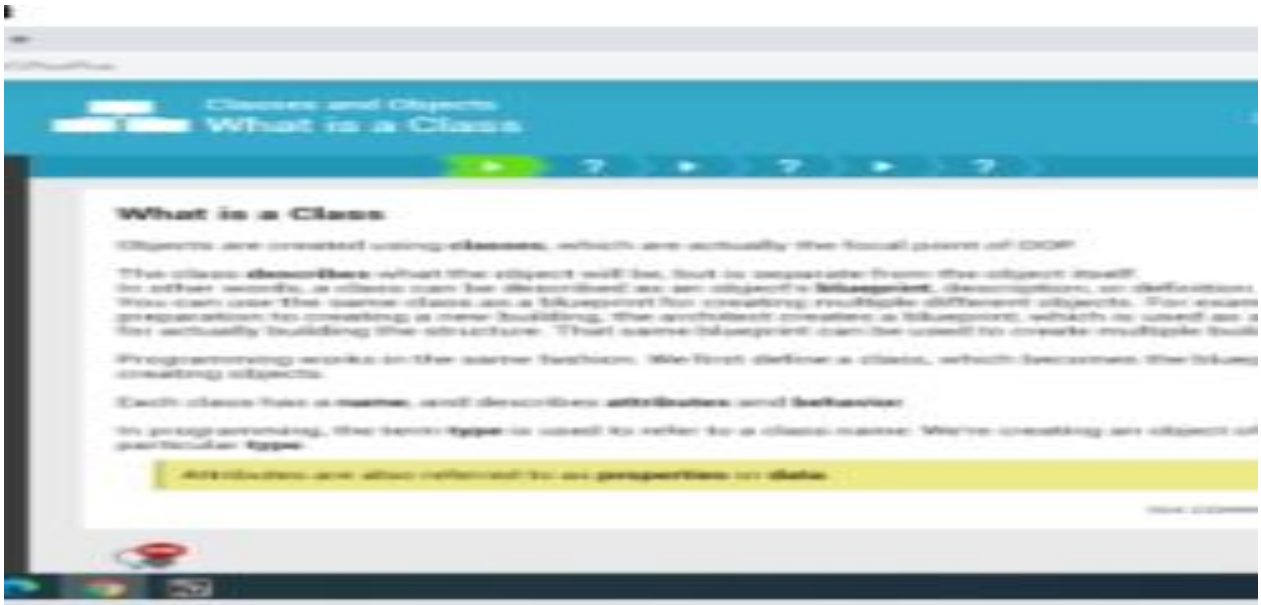
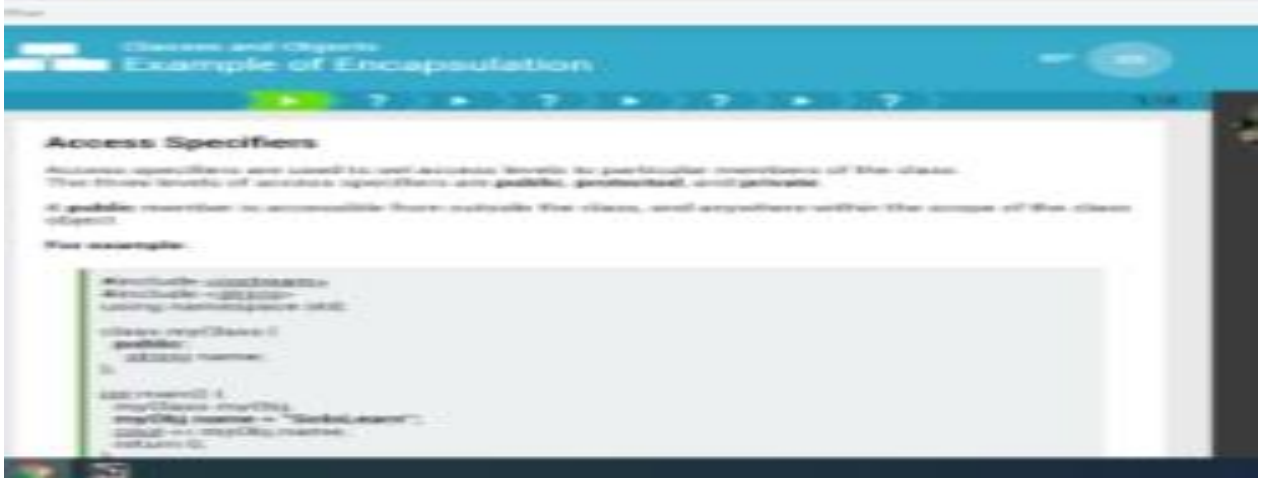
DAILY ASSESSMENT FORMAT

Date:	**24-06-2020**	**Name:**	**Sahana S R**
Course:	**C++ programming**	**USN:**	**4AL17EC083**
Topic:	**Classes and objects,constructor**	**Semester & Section:**	**6th sem 'B' sec**
Github Repository:	**sahanasr-course**		

FORENOON SESSION DETAILS

Image of session





OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. Smalltalk is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

In C++, Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.In other words, object is an entity that has state and behavior. Here, state means data and behavior means functionality.Object is a runtime entity, it is created at runtime.Object is an instance of a class. All the members of the class can be accessed through object.

Let's see an example to create object of student class using s1 as the reference variable.

```
Student s1;  //creating an object of Student
```

Class:

In C++, object is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

Let's see an example of C++ class that has three fields only.

```
class Student
 {
    public:
```

```
    int id;  //field or data member
    float salary; //field or data member
    String name;//field or data member
 }
```

Object and Class Example

Let's see an example of class that has two fields: id and name. It creates instance of the class, initializes the object and prints the object value.

```
#include <iostream>
using namespace std;
class Student {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};
int main() {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Sonoo Jaiswal";
    cout<<s1.id<<endl;
    cout<<s1.name<<endl;
    return 0;
}
```

Constructor:

In C++, constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. The constructor in C++ has the same name as class or structure.

There can be two types of constructors in C++.

- Default constructor
- Parameterized constructor

Default Constructor:

A constructor which has no argument is known as default constructor. It is invoked at the time of creating object.

Let's see the simple example of C++ default Constructor.

```
#include <iostream>
using namespace std;
class Employee
```

```
 {
  public:
      Employee()
      {
         cout<<"Default Constructor Invoked"<<endl;
      }
};
int main(void)
{
   Employee e1; //creating an object of Employee
   Employee e2;
   return 0;
}
```

Parameterized Constructor:

A constructor which has parameters is called parameterized constructor. It is used to provide different values to distinct objects.

Let's see the simple example of C++ Parameterized Constructor.

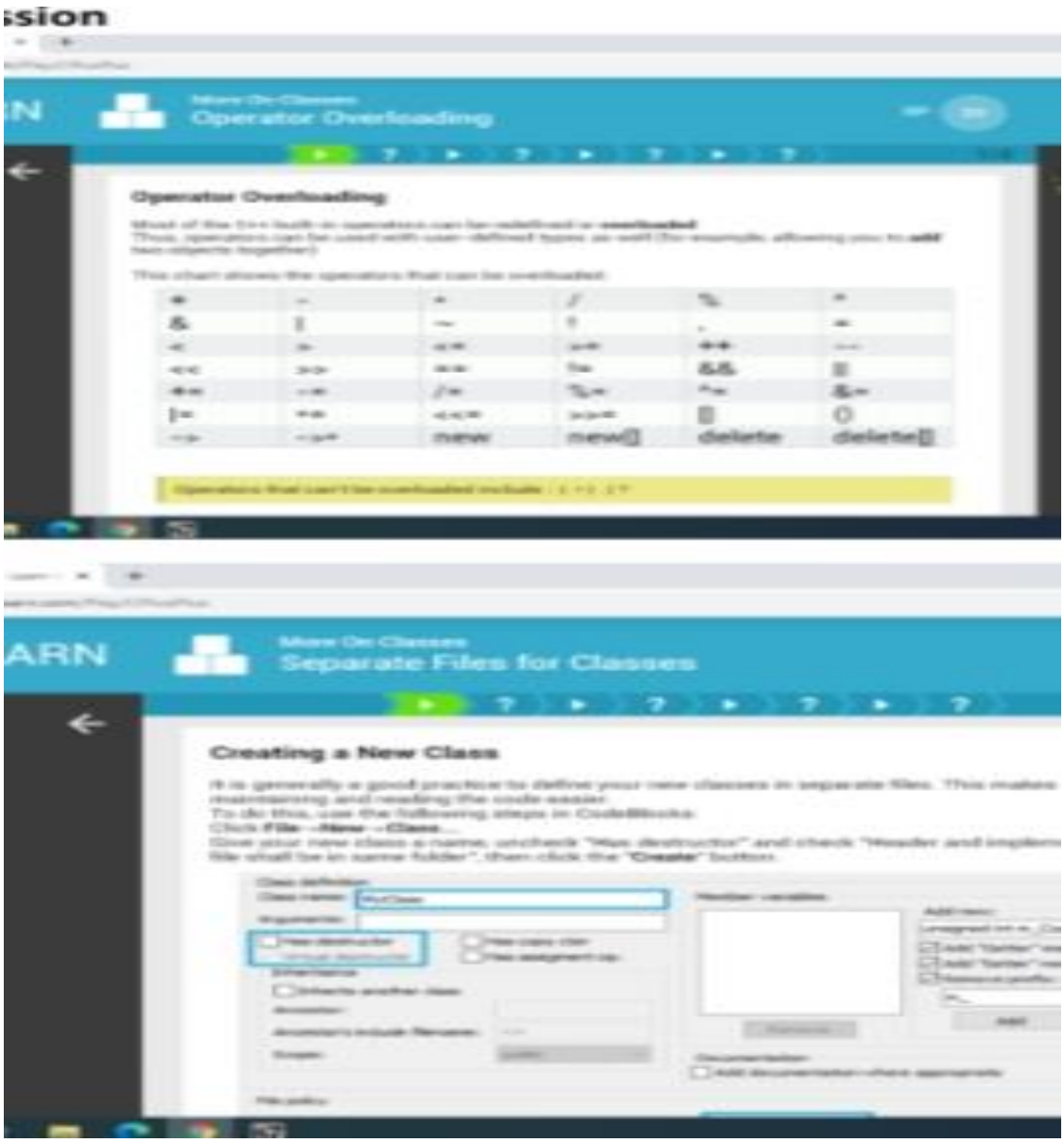
```
#include <iostream>
using namespace std;
class Employee {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     float salary;
     Employee(int i, string n, float s)
      {
         id = i;
         name = n;
         salary = s;
      }
     void display()
      {
         cout<<id<<"  "<<name<<"  "<<salary<<endl;
      }
};
int main(void) {
   Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
   Employee e2=Employee(102, "Nakul", 59000);
   e1.display();
   e2.display();
   return 0;
}
```

DAILY ASSESSMENT FORMAT

Date:	**24-06-2020**	**Name:**	**Sahana S R**
Course:	**C++ programming**	**USN:**	**4AL17EC083**
Topic:	**Destructor ,this pointer, friend function**	**Semester & Section:**	**6th sem 'B' sec**
Github Repository:	**sahanasr-course**		

AFTERNOON SESSION DETAILS

Image of session



Destructor:

A destructor works opposite to constructor; it destructs the objects of classes. It can be defined only once in a class. Like constructors, it is invoked automatically.

A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).

```
#include <iostream>
using namespace std;
class Employee
 {
   public:
       Employee()
       {
          cout<<"Constructor Invoked"<<endl;
       }
       ~Employee()
       {
          cout<<"Destructor Invoked"<<endl;
       }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2; //creating an object of Employee
    return 0;
}
```

this Pointer:

In C++ programming, this is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

- It can be used to pass current object as a parameter to another method**.**
- It can be used to refer current class instance variable.
- It can be used to declare indexers.

```
#include <iostream>
using namespace std;
class Employee {
   public:
      int id; //data member (also instance variable)
      string name; //data member(also instance variable)
      float salary;
      Employee(int id, string name, float salary)
       {
           this->id = id;
```

```
        this->name = name;
        this->salary = salary;
    }
    void display()
    {
        cout<<id<<"  "<<name<<"  "<<salary<<endl;
    }
};
int main(void) {
   Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
   Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee
   e1.display();
   e2.display();
   return 0;
}
```

Friend function:

If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.

By using the keyword friend compiler knows the given function is a friend function.

For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend

Characteristics of a Friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.

```
#include <iostream>
using namespace std;
class Box
{
   private:
```

```
      int length;
   public:
      Box(): length(0) { }
      friend int printLength(Box); //friend function
};
int printLength(Box b)
{
  b.length += 10;
   return b.length;
}
int main()
{
   Box b;
   cout<<"Length of box: "<< printLength(b)<<endl;
   return 0;
}
```