Unit 2:

C++ OOPs Concepts :

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.

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

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.A Class in C++ is the foundational element that leads to Object-Oriented programming. A class instance must be created in order to access and use the user-defined data type's data members and member functions. An object's class acts as its blueprint.

inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Sub class - Subclass or Derived Class refers to a class that receives properties from another class.

Super class - The term "Base Class" or "Super Class" refers to the class from which a subclass inherits its properties.

Reusability - As a result, when we wish to create a new class, but an existing class already contains some of the code we need, we can generate our new class from the old class thanks to inheritance. This allows us to utilize the fields and methods of the pre-existing class.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. Data abstraction is the process of exposing to the outside world only the information that is absolutely necessary while concealing implementation or background information. For example: phone call, we don't know the internal processing.

In C++, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

C++ 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
```

In this example, Student is the type and s1 is the reference variable that refers to the instance of Student class.

C++ Class

In C++, class 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

 }
```

C++ 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;
}
```

Output:

```
201
Sonoo Jaiswal
```

C++ Class Example: Initialize and Display data through method

Let's see another example of C++ class where we are initializing and displaying object through method.

```
#include <iostream>
using namespace std;
class Student {
   public:
       int id;//data member (also instance variable)
       string name;//data member(also instance variable)
       void insert(int i, string n)
        {
            id = i;
            name = n;
        }
       void display()
        {               cout<<id<<"  "<<name<<endl;
        }
};
int main(void) {
    Student s1; //creating an object of Student
    Student s2; //creating an object of Student
    s1.insert(201, "Sonoo");
    s2.insert(202, "Nakul");
    s1.display();
    s2.display();
    return 0;
}
```

Output:

```
201  Sonoo
```

Access Modifiers in C++

Access modifiers are used to implement an important aspect of Object-Oriented Programming known as **Data Hiding**. Access Modifiers or Access Specifies in a class are used to assign the accessibility to the class members.
There are 3 types of access modifiers available in C++:

1. **Public**
2. **Private**
3. **Protected**

Public : The `public` keyword is an **access specifier.** Access specifiers define how the members (attributes and methods) of a class can be accessed. In the example above, the members are `public` - which means that they can be accessed and modified from outside the code.
ex.

```
#include <iostream>
using namespace std;
class MyClass {   // The class
  public:         // Public access specifier
    int x;        // Public attribute (int variable)
};

int main() {
  MyClass myObj;  // Create an object of MyClass

  // Access attributes and set values
  myObj.x = 15;
  // Print values
  cout << myObj.x;
  return 0;
}
```

example 2:

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

// define a class
class Sample {
    // public elements
   public:
    int age;
    void displayAge() {
        cout << "Age = " << age << endl;
    }
};

int main() {
    // declare a class object
    Sample obj1;

    cout << "Enter your age: ";

    // store input in age of the obj1 object
    cin >> obj1.age;
    // call class function
    obj1.displayAge();

    return 0;
}
```

2. Private: The class members declared as *private* can be accessed only by the member functions inside the class. They are not allowed to be accessed directly by any object or function outside the class. Only the member functions or the friend functions are allowed to access the private data members of the class.
ex.

```
class MyClass {
  public:    // Public access specifier
    int x;   // Public attribute
  private:   // Private access specifier
    int y;   // Private attribute
};

int main() {
  MyClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

3. Protected: The protected access modifier is similar to the private access modifier in the sense that it can’t be accessed outside of its class unless with the help of a friend class. The difference is that the class members declared as Protected can be accessed by any subclass (derived class) of that class as well.

Example

```
#include <iostream>
using namespace std;
// declare parent class
class Sample {
    // protected elements
   protected:
    int age;
};

// declare child class
class SampleChild : public Sample {

   public:
    void displayAge(int a) {
        age = a;
        cout << "Age = " << age << endl;
    }
};

int main() {
    int ageInput;
    // declare object of child class
    SampleChild child;
    cout << "Enter your age: ";
    cin >> ageInput;
    // call child class function
    // pass ageInput as argument
    child.displayAge(ageInput);

    return 0;
}
```