Date:

Name:- Muhammad Talha
SapId:- 46287
Subject:- OOP
Sem/Sec:- 2-A
Instructor:- Sir Shahzad
Date :- 27-03-2023

(Q#1) How to create a class?

Ans. To create a class in c++ follow these steps:-

1- Define the class name - This should be meaningful name that describes with the class represent-

2- Define the class variable that will hold data- These variables could be public, Private and protected-

3- Define the class function that will operate on class variables-

4- Define constructor - The constructor is a special function which operates

Date:

when object of class is created.

5- Define destructor- The destructor is also a special function that is called when an object of the class is destroyed.

Example:-

```
class Person
{
    private:
        string name;
        int age;
    public:
        string getName() {
            return name; }
        int getAge(){
            return age;
        }
    Person (string n, string a) {
            name=n;
            age=a; }
        ~Person(){
        cout<<" Destructor called "; }
};
```

Date:

(Q#2) How to add data members & functions?

(Ans)

```
class Myclass
{
    private:
        int myint; // Data Member
        float myfloat; // Data Member
    public:
        void setint(int value)
        {
            myint = value; //Function
        }
};
```

The Myclass has two data members (myint and myfloat) and one function (setint) - Private keyword is used to specify that the data members are only accessible in the class - Public can be accessible outside class.

Date:_____

(Q#3) How and when to use Access Modifiers?

(Ans) Access Modifiers are used to specify the accessibility of class members. There are 3 types of modifiers in C++.

1- Public.

2- Private.

3- Protected.

Public members can be accessed from anywhere in the program, including outside the class.

Private members can only be accessed from within the class, and not from outside the class.

Protected members can be accessed from within the class and from its derived class.

```
class Example
{
    private:
        int privatem;
    public:
        int public_m;
    protected:
        int protected_m;
};
```

(QNO4) Encapsulation:-

Encapsulation is a fundamental concept in oop that refers to the practice of bundling with a class- Encapsulation is implemented by using access Specifiers -

1-public-

2- Private-

3- Protected-

By default membess of a class in C++ are private, so to acess them we ~~de~~ need to define public methods

Date:

called getters and setters - which allows us to read or modify the private data members indirectly. This ensures that the object's internal state is controlled and managed by class.

```
class car{
  private:
      int speed;
      int gear;
  public:
      int getspeed(){
          return speed;
      }
      void setspeed (int a){
          speed = s;
      }
      int getgear(){
          return gear;
      }
      void getgear(int g){
          gear = g;
      }
};
```

(QNo-7) Abstraction:-

Abstraction is another fundamental concept in oop that refers to the practice of representing complex systems. Abstract classes are classes that cannot be instantiated but serve as a base class for other classes. They define a set of pure virtual functions (ie., functions without an implementation), which are intended to be implemented by derived class. Abstract classes provide a way to create a general definition of a concept, which can be further customized and implemented by the derived class.

```
class shape
{
    public:
    virtual double area() const = 0;
    virtual double peri() const = 0;
};
```

Date:

virtual function helps you in ensuring that you are calling the correct function via a reference or pointer.

(Q#6) Inheritance:-

Inheritance is a fundamental concept in oop that allows us to create a new class by inheriting the properties and behaviours of an existing class. Inheritance is implemented using the class keyword, followed by a colon and the access specifier public, private & protected and the name of base class.

The derived class inherits all the member variables and member functions of base class and it can also add new member variables and new member functions or override the existing ones.

```
class Animal
{
    public:
    void eat()
    {
        std :: cout << "I am eating\n";
    }
};

class Dog: public Animal
{
  public:
    void bark()
    {
        std :: cout << " Woof woof!\n";
    }
};
```

In this example, the Dog class is derived from the Animal class using the public access specifier.