Lecture Content:

1. **Procedural programming vs Object Oriented Programming**
2. **Characteristics of OOP**
3. **Class and object**

1) Procedural programming vs Object Oriented Programming

Until now were working in procedural programming domain. There are two common programming methods in practice today: procedural programming and object-oriented programming (or OOP).

- **Procedural programming** is a method of writing a program centered on the procedures or actions that take place in a program.
    - Typically, data is stored in a collection of variables and/or structures, coupled with a set of functions that perform operations on the data.
    - The data and the functions are separate entities.
    - Variables and data structures in a procedural program are passed to the functions that perform the desired operations.
    - The main disadvantage is programs become larger and more complex, the separation of a program s data and the code that operates on the data can lead to problems (especially when format of Data is altered)
- Object-oriented programming is centered on the object. Objects are created from abstract data types that encapsulate related data and functions together. An object is a software entity that contains both data and procedures.
    - The data that are contained in an object are known as the **objects attributes**.
    - The procedures that an object performs are called **member functions**.
    - The object is, conceptually, a self-contained unit consisting of attributes (data) and procedures (functions).

2) Four Basic Characteristics of OOP:

1. Encapsulation (Data Hiding)
2. Abstraction
3. Inheritance
4. Polymorphism

- OOP addresses the problems that can result from the separation of code and data through encapsulation and data hiding. Encapsulation refers to the combining of data and code into a single object. Data hiding refers to an object s ability to hide its data from code that is outside the object. Only the object s member functions may directly access and make changes to the object s data. **An object typically hides its data, but allows outside code to access its member functions.**
    - When an object s internal data are hidden from outside code, and access to that data is restricted to the object s member functions, the data are protected from accidental corruption.
    - Programming code outside the object does not need to know about the format or internal structure of the object's data.

- When a programmer changes the structure of an object s internal data, he or she also modifies the object s member functions so they may properly operate on the data. The way in which outside code interacts with the member functions, however, does not change.
- OOP has also been encouraged by the trend of object reusability through inheritance. An object is not a stand-alone program, but is used by programs that need its service.

Class: In C++, the class is the construct primarily used to create objects. A class is code that specifies the attributes and member functions that a particular type of object may have. Think of a class as a "MAP" that objects may be created from.

Access Specifiers:

Access specifiers specify how class members may be accessed**. These are private, public and protected.**

- Programming statements/codes outside the class cannot access private class members. By default, member of a class are private.
- A class s public members may be accessed by code outside the class.
- Protected data members are only accessed by the derived classes of a class.

Content:

- **How to create a class**
- **Instantiation of an object of a class**
- **Constructors**
  - **Default constructors**
  - **Parameterized constructor**
- **Utility functions**
- **Accessors and Mutator Functions**

1) How to create a class:

A class is similar to a structure. It is a data type de ned by the programmer, consisting of variables and functions. Here is the general format of a class declaration:

```
class ClassName
{
        private:
                //private members are declare here
                .
                .
                .
        public:
                //public members are declare here
                .
                .
                .
};
```

- The declaration statements inside a class declaration are for the variables and functions that are members of that class.
- The **data items** within a class are called ***data members*** or ***data fields*** or ***instance variables.***
- ***Member functions*** are **functions** that are included within a class. Also known as ***instance functions, behaviors or methods***.
- Private class members cannot be accessed by programming statements outside the class. By default, access of a class is private. Member Variables (instance variables) are mostly declared as private.
- Public members may be accessed by code outside the class. To allow access to a class's private member variables, you create public member functions that work with the private member variables.

- Notice that the access specifiers are followed by a colon (:), and then followed by one or more member declarations (As Mentioned in player class below).
- Member functions can be defined inside class scope and are implicitly declared inline.
- Member functions can be defined outside class body/definition tied withnscope resolution (::) operator are still in class scope. It can be done by using following general format:

ReturnType ClassName :: functionName (ParameterList)

- Best practice is define function outside class deceleration. Only small and stable member functions should be defined inside class definition.

2) Instantiation of an object of a class

Class objects must be defined after the class is declared. An object can be created by using syntax:

ClassName objectName; //instantiation of class object

- In the general format, ClassName is the name of a class and objectName is the name we are giving the object. Defining a class object is called the *instantiation* of a class.
- Class objects are not created in memory until they are defined.
- A member function is accessed by using any already existing object's reference as:

objectName.funcName();

- When an object is used to call a member function, the member function has direct access to that object s member variables.
- An object has a unique identity, state, and behaviors.
- The **state** of an **object** consists of ***a set of data fields*** (also known as **properties**) with their **current values**.
- An object's state is simply the data that is stored in the object s attributes at any given moment. Object's state can be changed by calling any member function.
- **Avoid stale data**

3) Constructor:

***A Constructor is a special type of function i*nvoked automatically (implicitly) to construct (initialize) objects from the class. i.e., it instantiate the instance variables of an object.**

- **Constructor has exactly the same name as the defining class. Do not have any return type (Not even a void).**
- **Constructors can be *overloaded* (i.e., multiple constructors with different signatures)**
- **A class may be declared without constructors. In this case, a *no-argument constructor with an empty body* is implicitly declared (by compiler) in the class known as *default constructor.***

4) Utility functions:

These are helper functions in a class used for any calculation or printing purposes.

5) Accessors:

A member function that gets a value from a class s member variable but does not change it is known as an accessor. It is also called getter function. Getter functions are often declared/defined as constant functions because they are not changing value of a member variable.

6) Mutators:

A member function that stores a value in member variable or changes the value of member variable in some other way is known as a mutator. It is also called setter function.

Example Code:

// ..... In this code all class member functions are defined inline inside class Definition

```
class player
{
	int Id;
	int Scores[5];
	float Average;


public:
	player () {}//Default Constructor
	player(int i, int s[], float avg)//Parameterized Constructor
	{
		Id = i;
		for (int i = 0; i < 5; i++)
		{
			Scores[i] = s[i];
		}
		Average = avg;
	}

	// .......  Utility Functions ........
	void print() const
	{
		cout << "\nId of Player is  :" << Id;
		cout << "\nScores of Player are  :";
		for (int i = 0; i < 5; i++)
		{
			cout << Scores[i] << " ";
		}

		cout << "\nAverage is"<<Average;

	}
	float calAverage(void)
	{
		float s = 0.0;
		for (int i = 0; i < 5; i++)
```

```
		{
			s += Scores[i];
		}
		Average = s / 5;

		return Average;
	}

	//......  Setter or Mutator Functions ......
	void setId(int i)
	{
		Id = i;
	}
	void setscore()
	{
		cout << "\nEnter 5 scores for player " << Id;
		for (int i = 0; i < 5; i++)
			cin >> Scores[i];

	}

	// ..... Accessor or Getter functions .......
	int getID() const
	{
		return Id;
	}
	float getAvg(void) const
	{
		return Average;
	}


};

int main()
{
//...... Write code to implement player class

}
```

// ..... In this code all class member functions are defined out of line. i.e., out side the class Defination

```
class player
{
	int Id;
	int Scores[5];
	float Average;


public:
	player();//Default Constructor
	player(int, int [], float);//Parameterized Constructor

	// .......  Utility Functions ........
	void print( void) const;

	float calAverage(void);
```

```
        //......  Setter or Mutator Functions ......
        void setId(int i);
        void setscore(void);

        // ..... Accessor or Getter functions .......
        int getID(void) const;
        float getAvg(void) const;

};

player::player() {}//Default Constructor
player::player(int i, int s[], float avg)//Parameterized Constructor
{
        Id = i;
        for (int i = 0; i < 5; i++)
        {
                Scores[i] = s[i];
        }
        Average = avg;
}

// .......  Utility Functions ........
void player::print() const
{
        cout << "\nId of Player is  :" << Id;
        cout << "\nScores of Player are  :";
        for (int i = 0; i < 5; i++)
        {
                cout << Scores[i] << " ";
        }

        cout << "\nAverage is" << Average;

}
float player::calAverage(void)
{
        float s = 0.0;
        for (int i = 0; i < 5; i++)
        {
                s += Scores[i];
        }
        Average = s / 5;

        return Average;
}

//......  Setter or Mutator Functions ......
void player::setId(int i)
{
        Id = i;
}
void player::setscore()
{
        cout << "\nEnter 5 scores for player " << Id;
        for (int i = 0; i < 5; i++)
                cin >> Scores[i];

}
```

```
// ..... Accessor or Getter functions .......
int player::getID() const
{
	return Id;
}
float player::getAvg(void) const
{
	return Average;
}
int main()
{
	//...... Write code to implement player class

}
```