Content:

- **Constructor with default parameters**
- **Initializer list with constructor**
- **Static data member and static functions**

1) Constructor with default parameters

- For any class we can create a constructor, which has default arguments for all its parameters.
- Constructor with default parameters can be called with all, limited or no explicit arguments. (Recall default arguments are passed to parameters automatically if no argument is provided in the function call. The default value is listed in the parameter list of the function s declaration or the function header)
- Remember it then becomes the default constructor. Think about this.

See in the code…1 below.

See its implementation carefully. Especially implementation of Default parameterized constructor and driver function.

```
//////////////////////////////.....Code...1.....//////////////////////////
class player
{
	int Id;
	int Scores[5];
	float Average;

	public:
	//.....Both default and Parameterized constructors are commented .....//
	//player();//Default Constructor
	//player(int, int [], float);//Parameterized Constructor

	///... Constructor with default parameters

	player(int = 1, int[] = NULL, float = 0.0);

	// .......  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;


};
```

```
///... Constructor with default parameters

player::player(int i, int a[], float avg)
{

	cout << "\nIn Default Parameterized Constructor\n";
	Id = i;
	if (a == NULL)
	{
		for (int i = 0; i < 5; i++)
		{
			Scores[i] = 1;
		}

	}
	else {

		for (int i = 0; i < 5; i++)
		{
			Scores[i] = a[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;
}


//////////////////////////////Driver Function Implementation/////////////////////
int main()
{
	int a[] = { 1,2,3,4,5 };
	player p1;
	p1.print();
	player p2(2);
	p2.print();
	player p3(3, a);
	p3.print();
	player p4(4,a,0);
	p4.print();

}
```

2) Initializer list with constructor

- All data members can be initialized using member initializer syntax with constructors.
- constants and references must be initialized using member initializer syntax

See in the code..2 below.

2) Static data member and static functions

- It is possible to create a member variable or member function that does not belong to any instance/object of a class. Such members are known as a static member variables and static member functions.
- When a value is stored in a static member variable, it is not stored in an instance of the class.
- Static member variables and static member functions belong to the class instead of to an instance/object of the class.
- Remember Static data member:
  - is declared with static Keyword.
  - Only one copy is created in memory.
  - Shared between all instances/objects of the class. [As given in code below it is use to count the number of objects created]
  - Initialized outside of the class declaration at file scope (**but have class scope**). [It is required and creates static variable in memory]
  - The lifetime is the lifetime of the program.
  - Exists before any instances of the class are created.
  - only accessible to objects of same class
  - Can be **public**, **private** or **protected**
- A static member function can be declare by placing the static keyword in the function's prototype.
- Remember Static member function:
  - cannot access any non-static member data of a class.
  - can be called before any instances of the class are created.

See in the code..2 below.

```
////////////////////////////.....Code...2.....//////////////////////
//////////////////////.....Player class with Static data member and Static Function.....///

class player
{
	int Id;			///////
	int Scores[5];		////////---------> Non Static Data
	float Average;		//////

	static int count; // Static Data member of class


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;

	//..... Static Member Functions ..........
	static void printcount(void);
};

int player::count = 0; //Initialization of Static variable o  Initialized outside of the
                           class declaration at file scope (but have class scope). [It is
                           required and creates static variable in memory]

player::player() //Default Constructor
{
	cout << "\nIn Default Parameter less Constructor\n";
	count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s[], float avg):Id(i),Average(avg)
{
	cout << "\nIn Parameterized Constructor\n";

	for (int i = 0; i < 5; i++)
	{
		Scores[i] = s[i];
	}
	count++;
}

// .......  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;
	cout << count;

}
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;
}
void player::printcount(void)
{
	cout << "\nNo. of Objects Created are   ::  " << count;
}
```

```
//////////////////////////Driver Function Implementation/////////////////////
int main()
{

	player::printcount();
	player p1;

	player::printcount();

	int a[] = { 1,2,3,4,5 };
	player p2(2, a, 0);
	p2.calAverage();
	p2.print();

	player::printcount();
	p2.printcount();

}
```

- **Destructors**
- **Pointer data member of a class**
- **Constant objects of a class**
- **Constant data member of a class**

1) Destructors:

- Destructors is a function in every class, which is called when the object goes out of scope. i.e. when,
  - a function ends
  - the program ends
  - a block containing local objects/variables ends
  - a delete operator is called
- Like constructor, destructor is automatically/implicitly called when an object is destroyed (The main purpose of destructor is to remove all dynamic memories)
- Cleanup is as important as initialization and is guaranteed through the use of destructors.
- A destructor:
  - Have same name as class name proceeded by ~ (tilde).
  - Have no return type.
  - Does not have any arguments/parameters because it never needs any options.
  - Cannot be overloaded
- If we do not write our own destructor in a class, compiler creates a **default destructor** for us. The default destructor works fine unless we have dynamically allocated memory or pointer in class. When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed.

See in the code...3 below:

```
////////////////////////////.....Code...3.....//////////////////////////
//////////////////////.....Player class with Destructor implementation.....////////////////

#include <iostream>

using namespace std;

class player
{
	int Id;			///////
	int Scores[5];		////////---------> Non Static Data
	float Average;		//////

	static int count; // Static Data member of class


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

	~player();//Destructor Declaration

	// .......  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;

	//..... Static Member Functions ..........
	static void printcount(void);
};

int player::count = 0;

player::player() //Default Constructor
{
	cout << "\nIn Default Parameter less Constructor\n";
	count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s[], float avg) :Id(i), Average(avg)
{
	cout << "\nIn Parameterized Constructor\n";

	for (int i = 0; i < 5; i++)
	{
		Scores[i] = s[i];
	}
	count++;
}
//Destructor Implementation
```

```
player::~player()
{
	cout << "\nIn Destructor\n";
	count--;
	cout << count;
}

// .......  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;
	cout << count;

}
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;
}
void player::printcount(void)
{
	cout << "\nNo. of Objects Created are   ::  " << count;
```

```
}

////////////////////////////Driver Function Implementation/////////////////////
int main()
{

	player::printcount();
	player p1;

	player::printcount();

	int a[] = { 1,2,3,4,5 };
	player p2(2, a, 0);
	p2.calAverage();
	p2.print();

	player::printcount();
	p2.printcount();


}
```

2) Pointer data member of a class

- A class can contain a pointer data member.
- It can be used for dynamic memory allocation.
- **Remember:** When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed.

See Code 4 below and understand it well:

```
////////////////////////////.....Code...4.....////////////////////////////
///////.....Player class with Destructor implementation for dynamic Memory Allocation .....////

#include <iostream>

using namespace std;

class player
{
	int Id;              ///
	int *Scores;          ///
	float Average;          /// ----------> Non Static Data
	int size;               ////

	static int count; // Static Data member of class


public:
	player();//Default Constructor
	player(int, int, float = 0);//Parameterized Constructor last parameter is Default

	~player();
```

```
        // .......  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;

        //..... Static Member Functions ..........
        static void printcount(void);
};

int player::count = 0;

player::player() //Default Constructor
{
        cout << "\nIn Default Parameter less Constructor\n";
        count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s, float avg) :Id(i), Average(avg)
{
        cout << "\nIn Parameterized Constructor\n";
        size = s;
        Scores = new int[size];
        cout << "\nEnter " << size << " Values for scores";
        for (int i = 0; i < size; i++)
        {
                cout << "\nEnter " << i + 1 << " Value : ";
                cin >> Scores[i];
        }
        count++;
}
//Destructor Implementation
player::~player()
{
        cout << "\n: : : Destructor is called for ID  "<<Id<<"  :::"<<endl;
        delete [] Scores;
        count--;
        cout << "\nRemianing objects are :: "<<count;
}

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

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

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

	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 < size; i++)
	{
		cout << "\n Enter " << i << " Value";
		cin >> Scores[i];
	}

}

// ..... Accessor or Getter functions .......
int player::getID() const
{
	return Id;
}
float player::getAvg(void) const
{
	return Average;
}
void player::printcount(void)
{
	cout << "\nNo. of Objects Created are   ::  " << count;
}

////////////////////////////Driver Function Implementation/////////////////////
int main()
{

	player p1(1, 5);
	p1.calAverage();
	p1.print();
	player *p;
	p = new player(3, 3);
	p->calAverage();
	p->print();

	delete p;
```

```
        player p2(2, 4);
        p1.calAverage();
        p1.print();
}
```

//////////////////////////////////////Content of next Lecture//////////////////////////////

3) Constant objects of a class

- Principle of least privilege
  - Only give objects permissions they need, no more
- Keyword const
  - Specify that an object is not modifiable
  - Any attempt to modify the object is a syntax error
- const (Constant) Objects and const Member Functions
  - **const** objects require **const** functions
  - Member functions declared **const** cannot modify their object
  - Constant functions returns the value of a data member but doesn't modify anything so is declared **const**
  - **const** must be specified in function prototype and definition

- Prototype:
  ReturnType FunctionName(param1,param2…) const;
- Definition:
  ReturnType FunctionName(param1,param2…) const { …}

- Example: in above code getID(), getAvgs() and print functions are constant functions. See and understand their prototype and defination.
- **Remember**: Constructors / Destructors cannot be constant as they need to initialize variables, therefore modifying them.

Use following driver function with player class implementation given in code 4 and see what happens.

```
int main()
{

	const player p1(2, 2);

	p1.print();

	cout << p1.getID();

	p1.setId(10);

	p1.calAverage();

}
```