**Content:**

- **Constant data member of a class**
- **Friend function and classes**
- **This pointer**

1) Constant data members of a class

- As with constant member functions, data members can also be constant
- Member initializer syntax is used to initialize **const**ant data members

```
//////////////////////////////.....Code...5.....//////////////////////////
//////////.....Player class with constant data member implementation .....////////

#include <iostream>

using namespace std;

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

	const char Gender; // ----> constant data member

	static int count; // ----> Static Data member of class


public:
	player();//Default Constructor
	player(int, int, char, float = 0);//Parameterized Constructor

	~player(); //Destructor

	// .......  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():Gender('M') //Default Constructor + initializer for constant data
{
	cout << "\nIn Default Parameter less Constructor\n";
	count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s, char g, float avg) : Gender(g) //initializer for constant
data
{
	Id = i;
	size = s;
	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 << "\nGender of Player is: " << Gender;
	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(3, 3, 'f');

	p1.calAverage();
	p1.print();



}
```

2) Friend function and classes

- A friend is a function or class that is not a member of a class, but has access to the private members of the class.
- Classes keep a list of their friends, and only the external functions or classes whose names appear in the list are granted access.
- A function is declared a friend by placing the key word friend in front of a prototype of the function.
- Here is the general format:

  friend *ReturnType FunctionName* (*ParameterTypeList*);
- How a class can be declared a friend of another class? Think about it and practice it.
- Remember:

  - Friendship is granted, not taken.
  - Not symmetric (if **B** a **friend** of **A**, **A** not necessarily a **friend** of **B**)
  - Not transitive (if **A** a **friend** of **B**, **B** a **friend** of **C**, **A** not necessarily a **friend** of **C**)

```
////////////////////////////.....Code...6.....//////////////////////////
//////////.....Player class with Friend Function implementation .....////////

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

	const char Gender; // ----> constant data member

	static int count; // ----> Static Data member of class


public:
	player();//Default Constructor
	player(int, int, char, float = 0);//Parameterized Constructor

	~player(); //Destructor

	// .......  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);

	//......Decleration of Friend Functions......
	friend void Printall(player &);
};


int player::count = 0;

player::player():Gender('M') //Default Constructor + initializer for constant data
{
	cout << "\nIn Default Parameter less Constructor\n";
	count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s, char g, float avg) : Gender(g) //initializer for constant
data
{
	Id = i;
	size = s;
	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 << "\nGender of Player is: " << Gender;
	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;
}

///....Defination of Printall global function Friend of Class player ..... ///
void Printall(player &p)
{
	cout << "\n\"In PrintAll Friend Function\" ";
	cout << "\nId of Player is  : " << p.Id;
	cout << "\nGender of Player is: " << p.Gender;
	cout << "\nScores of Player are  : ";
	for (int i = 0; i < p.size; i++)
	{
		cout << p.Scores[i] << " ";
	}
	cout << "\nAverage is : " << p.Average;

}
s
////////////////////////////Driver Function Implementation/////////////////////
int main()
{
	player p1(3, 3, 'f');

	Printall(p1);
```

}

3) **This pointer**

**To understand 'this' pointer, it is important to know how objects look at functions and data members of a class.**

- **Each object gets its own copy of the data member.**
- **All-access the same function definition as present in the code segment.**

**Meaning each object gets its own copy of data members and all objects share a single copy of member functions. Then now question is that if only one copy of each member function exists and is used by multiple objects, how are the proper data members are accessed and updated? The compiler supplies an implicit pointer along with the names of the functions as 'this'. The 'this' pointer is passed as a hidden argument to all non static member function calls and is available as a local variable within the body of all non static functions. 'this' pointer is not available in static member functions as static member functions can be called without any object (with class name).**

Reference: https://www.geeksforgeeks.org/this-pointer-in-c/

- **this** pointer
  - Allows objects to access their own address
  - Not part of the object itself
  - Implicitly reference member data and functions
- The type of the **this** pointer depends upon the type of the object and whether the member function using **this.**
- Examples using **this**
  - For a member function print data member **x**, either

    **this->x**

    or

    **( *this ).x**
- Cascaded member function calls
  - Function returns a reference pointer to the same object
  - **{ return *this; }**
  - Other functions can operate on that pointer
  - Functions that do not return references must be called last

•

```
////////////////////////////.....Code...7.....//////////////////////////
///////////.....Player class with this pointer implementation .....//////////


using namespace std;


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

	const char Gender; // ----> constant data member

	static int count; // ----> Static Data member of class


public:
	player();//Default Constructor
	player(int, int, char, float = 0);//Parameterized Constructor

	~player(); //Destructor

	// .......  Utility Functions ........
	player &print(void) ;

	player &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);

	//......Decleration of Friend Functions......
	friend void Printall(player &);
};


int player::count = 0;

player::player():Gender('M') //Default Constructor + initializer for constant data
{
	cout << "\nIn Default Parameter less Constructor\n";
	count++;
}
////....Initializer List with Parameterized Constructor
player::player(int i, int s, char g, float avg) : Gender(g) //initializer for constant
data
```

```
{
	Id = i;
	size = s;
	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 ........
player& player::print()
{
	cout << "\nId of Player is  : " << Id;
	cout << "\nGender of Player is: " << Gender;
	cout << "\nScores of Player are  : ";
	for (int i = 0; i < size ; i++)
	{
		cout << Scores[i] << " ";
	}

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

	return *this;

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

	return *this;
}

//......  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;
}

///....Defination of Printall global function Friend of Class player ..... ///
void Printall(player &p)
{
	cout << "\n\"In PrintAll Friend Function\" ";
	cout << "\nId of Player is  : " << p.Id;
	cout << "\nGender of Player is: " << p.Gender;
	cout << "\nScores of Player are  : ";
	for (int i = 0; i < p.size; i++)
	{
		cout << p.Scores[i] << " ";
	}

	cout << "\nAverage is : " << p.Average;

}
////////////////////////////Driver Function Implementation/////////////////////
int main()
{
	player p1(3, 3, 'f');


	p1.calAverage().print();



}
```