Lecture # 8 outline and homework (3-7-2023)

- **Player class**
- **Parameterized Default Constructor**
- **Constant member functions**
- **Initializer list**
- **Static data members and functions**

Code-1:

```
/////////. . . . . code1.cpp . . . . . . . . . ////////

class player
{
	int Id;
	char name;
	int Scores[5];//scores for 5 matches
	float Average;//Use to calculate average of scores

public:
	player(int = 0, char = 'a', int [] = NULL, float = 0.0);//Q. Default constructor
with default parameters

	// .......  Utility Functions ........
	float calAverage(void);
	void print() const;//Q. a new utility function named as print() that prints name,
ID, scores and average of a player

	//......  Setter or Mutator Functions ......
	void setId(int);
	void setName(char);
	void setScores(int[]);//Q. setScores() function

	// ..... Accessor or Getter functions .......
	int getID(void);
	char getName(void);
	float getAverage(void);//Q. getAverage() function
	~player(); //Destructor


};
////////////////////// . . . define class functions out of line/scope . . . ////////

player::player(int i, char n, int s[], float a)//Q. Default Constructor with default
parameters
{

	cout << "\nInside parameterized Constructor";
	Id = i;
	name = n;
	if (s == NULL)
	{
		cout << "\nEnter 5 score values";
		for (int i = 0; i < 5; i++)
```

```
		{
			cin >> Scores[i];
		}
		Average = 0;
	}
	else
	{

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

}

// .......  Utility Functions ........

float player::calAverage(void)
{
	cout << "\nInside CalculateAverage() Function\n";
	int s = 0;
	for (int i = 0; i < 5; i++)
	{
		s += Scores[i];
	}
	Average = s / 5.0;

	return Average;
}
void player::print() const
{
	cout << "\nInside print() function";
	cout << "\n Name of player is:   " << name;
	cout << "\n ID of player is:   " << Id;
	cout << "\n Scores of player is:   ";
	for (int i = 0; i < 5; i++)
	{
		cout << Scores[i] << " ";
	}
	cout << endl;
	/////.........intresting fact here ....../
	cout << "\n Average of player is:   "<<Average;

}

//......  Setter or Mutator Functions ......
void player::setId(int i)
{
	cout << "\nInside setId() function";
	Id = i;
}

void player::setName(char c)
{
	cout << "\nInside setName() function";
	name = c;
```

```
}
void player::setScores(int s[])
{
	cout << "\nInside setScores() function";
	for (int i = 0; i < 5; i++)
	{
		Scores[i] = s[i];
	}
}

// ..... Accessor or Getter functions .......
int player::getID()
{
	cout << "\nInside getId() function\n";
	return Id;
}
char player::getName(void)
{
	cout << "\nInside getName() function\n";
	return name;
}
float player::getAverage()
{
	cout << "\nInside getAverage() function\n";
	calAverage();
	return Average;

}

//Definition of Destructor
player::~player() //Destructor
{
	cout << "\nInside Destructor that does nothing\n";
}



int main()
{
	player p1;
	p1.print();
	p1.calAverage();
	p1.print();

}
```

code 2: Using dynamic array

```
/////////. . . . . code2.cpp . . . . . . . . . ////////
class player
{
	int Id;
	char name;
	int size;//determine size
	int *Scores;//pointer data member for storing array of scores which should be
dynamically allocated
	float Average;//Use to calculate average of scores

public:
	player(int = 0, char = 'a', int s=2, int * = NULL);//Q. Default constructor with
default parameters

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

	//......  Setter or Mutator Functions ......
	void setId(int);
	void setName(char);
	void setsize(int);
	void setScores(int *);//interesting

	// ..... Accessor or Getter functions .......
	int getID(void) const;
	char getName(void) const;
	float getAverage(void);
	int getsize(void) const;
	//How to write getscores function ?????

	~player(); //Destructor


};
////////////////////////// . . . define class functions out of line/scope . . . ////////

player::player(int i, char n, int s, int * arr)//Q. Default Constructor with default
parameters
{

	cout << "\nInside parameterized Constructor";
	Id = i;
	name = n;
	size = s;
	if (arr == NULL)
	{

		Scores = new int[size];
		cout<<"Enter values of 2 students";
```

```
		for (int i = 0; i < size; i++)
		{
			cin >> Scores[i];
		}
	}
	else
	{
		Scores = new int[size];
		cout << "Enter values of   "<<size<< " students";
		for (int i = 0; i < size; i++)
		{
			cin >> Scores[i];
		}

	}
	player::calAverage();//calculating average in constructor

}

// .......  Utility Functions ........

float player::calAverage(void)
{
	cout << "\nInside CalculateAverage() Function\n";
	int s = 0;
	for (int i = 0; i < size; i++)
	{
		s += Scores[i];
	}
	Average = float(s) / size;

	return Average;
}
void player::print() const
{
	cout << "\n.............................";
	cout << "\nInside print() function";
	cout << "\nName of player is :   " << name;
	cout << "\nID of player is :   " << Id;
	cout << "\nTotal matches played are : " << size;
	cout << "\nScores of player is:   ";
	for (int i = 0; i < size; i++)
	{
		cout << Scores[i] << " ";
	}
	cout << endl;

	cout << "\nAverage of player is:   " << Average;
	cout << "\n.............................\n";
}

//......  Setter or Mutator Functions ......
void player::setId(int i)
{
	cout << "\nInside setId() function";
	Id = i;
}
```

```
void player::setName(char c)
{
	cout << "\nInside setName() function";
	name = c;

}

void player::setsize(int s)
{
	cout << "\nInside setsize() function";
	size = s;
}
void player::setScores(int *arr)
{
	cout << "\nInside setScores() function";
	delete[] Scores;
	Scores = new int[size];

	for (int i = 0; i < size; i++)
	{
		Scores[i] = arr[i];
	}


}
// ..... Accessor or Getter functions .......
int player::getID(void) const
{
	cout << "\nInside getId() function\n";
	return Id;
}
char player::getName(void) const
{
	cout << "\nInside getName() function\n";
	return name;
}
float player::getAverage() ///an interesting fact inside function
{
	cout << "\nInside getAverage() function\n";
	calAverage();
	return Average;

}
int player::getsize(void) const
{
	return size;
}
//Definition of Destructor
player::~player() //Destructor
{
	cout << "\nInside Destructor that does memory delete\n";
	delete[] Scores;
}


int main()
{
	player p1;
```

```
    p1.print();
}
```

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. (I told you this in last week's lecture) see **code-1.cpp and code-2.cpp.**

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-3.cpp 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.

- Remember a non-static member function can access a static data.

```
/////////. . . . . code3.cpp . . . . . . . . . ////////
class player
{
	int Id;//		> .
	char name;//		> .
	int size;//		>  non static data members
	int *Scores;//		> .
	float Average;//	> .

	static int count;  //static data members

public:
	player(int = 0, char = 'a', int s=2, int * = NULL);//Q. Default constructor with
default parameters

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

	//......  Setter or Mutator Functions ......
	void setId(int);
	void setName(char);
	void setsize(int);
	void setScores(int *);//interesting

	// ..... Accessor or Getter functions .......
	int getID(void) const;
	char getName(void) const;
	float getAverage(void);
	int getsize(void) const;
	//How to write getscores function ?????

	static void showcount() // static function
	{
		//cout << name;
		cout << "\nValue of count" << count;
	}

	~player(); //Destructor


};
////////////////////// . . . define class functions out of line/scope . . . ////////

int player::count = 0;//assigning value to static data member of class

player::player(int i, char n, int s, int * arr)//Q. Default Constructor with default
parameters
{

	cout << "\nInside parameterized Constructor";
	Id = i;
	name = n;
```

```
	size = s;
	if (arr == NULL)
	{

		Scores = new int[size];
		cout<<"Enter values of 2 player";

		for (int i = 0; i < size; i++)
		{
			cin >> Scores[i];
		}

	}
	else
	{
		Scores = new int[size];
		cout << "Enter values of   "<<size<< " students";
		for (int i = 0; i < size; i++)
		{
			cin >> Scores[i];
		}

	}
	player::calAverage();//calculating average in constructor
	count++;
}

// .......  Utility Functions ........

float player::calAverage(void)
{
	cout << count;
	cout << "\nInside CalculateAverage() Function\n";
	int s = 0;
	for (int i = 0; i < size; i++)
	{
		s += Scores[i];
	}
	Average = float(s) / size;

	return Average;
}
void player::print() const
{
	cout << "\n.............................";
	cout << "\nInside print() function";
	cout << "\nName of player is :   " << name;
	cout << "\nID of player is :   " << Id;
	cout << "\nTotal matches played are : " << size;
	cout << "\nScores of player is:   ";
	for (int i = 0; i < size; i++)
	{
		cout << Scores[i] << " ";
	}
	cout << endl;

	cout << "\nAverage of player is:   " << Average;
	cout << "\n.............................\n";
```

```
}

//......  Setter or Mutator Functions ......
void player::setId(int i)
{
	cout << "\nInside setId() function";
	Id = i;
}

void player::setName(char c)
{
	cout << "\nInside setName() function";
	name = c;

}

void player::setsize(int s)
{

	cout << "\nInside setsize() function";
	size = s;
}
void player::setScores(int *arr)
{
	cout << "\nInside setScores() function";
	delete[] Scores;
	Scores = new int[size];

	for (int i = 0; i < size; i++)
	{
		Scores[i] = arr[i];
	}


}
// ..... Accessor or Getter functions .......
int player::getID(void) const
{
	cout << "\nInside getId() function\n";
	return Id;
}
char player::getName(void) const
{
	cout << "\nInside getName() function\n";
	return name;
}
float player::getAverage()///an interesting fact inside function
{
	cout << "\nInside getAverage() function\n";
	player::calAverage();
	return Average;

}
int player::getsize(void) const
{
	return size;
}
```

```
//Definition of Destructor
player::~player() //Destructor
{
	cout << "\nInside Destructor that does nothing\n";
	delete[] Scores;
	count--;
}



int main()
{
	// write your implementstion code here
}
```

3) 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...4.....//////////////////////////
///////////.....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();



}
```