Content:

- **Copy Constructor**
- **Arrays and Dynamic allocation of Class objects**
- **Operator Overloading**

1) Copy Constructor

```
//////////////////////////////.....Code...8.....//////////////////////////
////////.....Player class with this Copy Constructor 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
	////Copy constructor
	player(player&);

	~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++;
}


player::player(player & p):Gender(p.Gender)
{
	cout << "\nIn Copy Constructor\n";
	this->Id = p.Id;
	this->size = p.size;

	this->Scores = new int[this->size];
	cout << "\nEnter " << size << " Values for scores";
	for (int i = 0; i < this->size; i++)
	{
		cout << "\nEnter " << i + 1 << " Value : ";
		cin >> this->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();

	player p2 = p1;

	p2.print();

	p2.setscore();

	p1.print();


}
```

- **Operator Overloading**

```
////////////////////////////.....Code...9.....//////////////////////////
////////.....Player class with =operator overloading implementation .....////


#include <iostream>

using namespace std;


class player
{
private:
	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
	////Copy constructor
	player(const player&);

	~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 &);

	//....overloaded operators
	const player& operator=(const 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++;
}


player::player(const player & p):Gender(p.Gender)
{
	cout << "\nIn Copy Constructor\n";
	this->Id = p.Id;
	this->size = p.size;

	this->Scores = new int[this->size];
	cout << "\nEnter " << size << " Values for scores";
	for (int i = 0; i < this->size; i++)
	{
		cout << "\nEnter " << i + 1 << " Value : ";
		cin >> this->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;

}

///...operator overloading code;
const player
player&::operator=(const player &p)
{
        cout << "\nIn Overloaded operator= \n";
        this->Id = p.Id;
        this->size = p.size;

        delete[] this->Scores;
        this->Scores = new int[this->size];
        for (int i = 0; i < this->size; i++)
        {
                this->Scores[i] = p.Scores[i];
        }

        return *this;

}

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

        player p2(2, 2, 'f');

        player p3(1, 2, 'm');

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

        p2.print();

        p2 = p1;

        p2.print();


}
```