Lecture # 10 and 11 outline and homework

Today's Content:

- **Operator Overloading**

- **The variables of native data types can perform a number of different operations (functions) using operators (+, - , / , *)**
  - **Example: a + b * c**
  - **Example: if (a < b)**
- **However, with user defined (classes) objects we can not use operators**
  - **Example:**

    player obj1, obj2;

    if (ob1 < obj2)//obj1.player::operator<(obj2)

- **With the help of operator overloading we can add operator functionality in the class's objects**
- **However, before using any kind of operator we need to implement its functionality in the class**
- **In order to add operator functionality in the class**
  - **First create a function for the class**
  - **Set the name of the function with the operator name**
  - **operator+ for the addition operator '+'**
  - **operator> for the comparison operator '>'**
- **As shown in the player class below:**
- **Although the syntax of defining prototype of operator overloading is following in most of cases**
- **datatype operator+ (datatype)**
- **However, for some operators, there is little bit change is above syntax**
  - **++, -- operators**
  - **>> , << operators**
  - **& and [] operators**

- With operator overloading we cannot change
    - How operators act on built-in data types
        - i.e., cannot change integer addition
    - Precedence of operator (order of evaluation)
        - Use parentheses to force order-of-operations
    - Associativity (left-to-right or right-to-left)
    - Arity (Number of operands)
        - & is unitary, only acts on one operand
- Cannot create new operators
- Operators must be overloaded explicitly
    - Overloading + does not overload +=

Operators that can be overloaded							
+	-	*	/	%	^	&	|
~	!	=	<	>	+=	-=	*=
/=	%=	^=	&=	|=	<<	>>	>>=
<<=	==	!=	<=	>=	&&	||	++
--	->*	,	->	[]	()	new	delete
new[]	delete[]						

Operators that cannot be overloaded				
.	.*	::	?:	sizeof

////complete operator overloading code.

```
	void printall(void);///A gloabal function which we make friend of player class

class player
{
	int Id;//               //> .
	char name;//               //> .
	int size;//                   //>  non-static and  non-constant data members
	int *Scores;//             //> .
	float Average;//       //> .

	static int count;  //static data members

	const char gender;//Constant data member

public:
	player(int = 0, char = 'a', int s = 2, char = 'M', int * = NULL);//Default
parameterized constructor

	//player(int);

	player(double a):gender('M')///conversion function using constructor
	{
		Average = a;
	}



	//player();//default constructor discuss during lecture

	////Copy constructor//discuss during lecture
	player(const player&);

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

	//......  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

	///operator overloading
	void operator=(const player &);

	///implement here other arithmetic operators like operator-, operator*, operator/,
operator%, operator--
	player& operator+(player &);
	player& operator++(void);//prefix ++
	player& operator++(int);//postfix ++

	void operator|(const player &p)
	{
		cout << this->size<<endl;
		cout << p.size<<endl;

	}

	int& operator[](int i)
	{
		return Scores[i];
	}

	//Implement here other comparison and logical operators
	bool operator<(player &);

	//conversion function

	operator int()
	{
		return Id;
	}


	/////friend functions
	friend ostream& operator<<(ostream& out, player &p);
	friend void printall();//Granting printall() function as friend of class player

};
////////////////////////// . . . 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, char g, int *arr) : Id(i), name(n), size(s),
gender(g)//Constant data member must need intilizer with constructor
{
	cout << "\nInside parameterized Constructor : \n";

	if (arr == NULL)
	{

		Scores = new int[size];
		cout << "Enter values of " << size << "  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++)
			{
				cout << "\nEnter " << i + 1 << " Value : ";
				cin >> Scores[i];
			}

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


/*player::player():gender('M')//commented it as default parameterized
{

}*/
///Defination of copy constructor
////Copy constructor
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];
		}
		player::calAverage();
		count++;
}



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

player& player::calAverage(void)
{

		cout << "\nInside CalculateAverage() Function\n";
		int s = 0;
		for (int i = 0; i < size; i++)
		{
			s += Scores[i];
		}
		this->Average = float(s) / size;

		return *this;
```

```
}
player& player::print()
{
	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:   " << this->Average;
	cout << "\n.............................\n";
	return *this;
}

//......  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";
	this->size = s;
}
void player::setScores(int *arr)
{
	cout << "\nInside setScores() function";
	delete[] Scores;
	Scores = NULL;
	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 this->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 (*this).size;//return this->size;
}

//Definition of Destructor
player::~player() //Destructor
{
	cout << "\nInside Destructor that Delete Dynamic Memory\n";
	delete[] Scores;
	count--;
}
void player::operator=(const player &p)
{
	this->Id = p.Id;
	this->size = p.size;
	delete[] Scores;
	this->Scores = new int[size];
	cout << "\nEnter " << size << " Values for scores";
	for (int i = 0; i < this->size; i++)
	{
		cout << "\nEnter " << i + 1 << " Value : ";
		cin >> this->Scores[i];
	}
	player::calAverage();
}
////////////////////////////////////////////operator
overloading//////////////////////////////
player& player::operator+(player &p)
{
	player pt;

	pt.Average = this->Average + p.Average;

	return pt;
}

bool player::operator<(player &p)
{
	if (this->Average < p.Average)
		return true;
	else
		return false;
```

```
}

player& player::operator++(void)
{
	cout << "\ninside operator++(void) function ";
	for (int i = 0; i < this->size; i++)
		Scores[i]++;
	return *this;

}

player& player::operator++(int)
{
	cout << "\ninside operator++(int) function ";
	for (int i = 0; i < this->size; i++)
		Scores[i]++;

	return *this;

}

///.......Defination of printall() global function ..........//
//Read this function carefully and implement it in main ..............///
void printall()
{
	cout << "\n......Inside printall() global function.......\n";
	player p;
	cout << "\n.............................";
	cout << "\nInside print() function";
	cout << "\nName of player is :   " << p.name;
	cout << "\nID of player is :   " << p.Id;
	cout << "\nTotal matches played are : " << p.size;
	cout << "\nScores of player is:   ";
	for (int i = 0; i < p.size; i++)
	{
		cout << p.Scores[i] << " ";
	}
	cout << endl;

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


///implementation of operator<<()
ostream& operator<<(ostream& out, player &p)
{
	out << "\n.............................";
	out << "\nInside operator<<() function";
	out << "\nName of player is :   " << p.name;
	out << "\nID of player is :   " << p.Id;
	out << "\nTotal matches played are : " << p.size;
	out << "\nScores of player is:   ";
	for (int i = 0; i < p.size; i++)
	{
		out << p.Scores[i] << " ";
	}
	out << endl;
```

```
	out << "\nAverage of player is:   " << p.Average;
	out << "\n.............................\n";
	return out;

}

int main()
{
	// write your implementstion code here
	player p1(3, 'A', 4);// , p2;

	int i;

	i = p1;//i = int(p1);

	p1 = 3.2//p1 = player(3.2)

	cout << "\nID of player   is    " << i;

	//cout <<"\n Value is :  "<< p1[2];

	//p1[2] = 10;
//	cout << "\n Value is :  " << p1[2];

	//p2 | p1; //p2.operator|(p1);




	/*
	player p2;

	p2 = ++p1;//see what will happen

	cout << p1;


	if (p1 < p2)
		cout << "\nhello";
	else cout << "\nNot hello";
	*/
}
```

Problem: Consider the following code for a **library management system** that maintains the record of the books and its members. As mentioned in the comments below, you may assume that all getter and setter functions are implemented in the classes (Book, Member, and Library). Now, you are required to implement/overload all the operators that the main function needs to run correctly. Note that the correct prototype and the body of the function carry equal marks for each overloaded operator.

```
#include <iostream>
using namespace std;

class Book{
        string sTitle;
        string sISBN;
        int nCopiesAvailable;
public:
        Book(string title, string isbn, int copies){
                sTitle = title;
                sISBN = isbn;
                nCopiesAvailable = copies;
        }
        // getter/setter functions for title, isbn, and copies are available
        // do not implement them
};

class Member{
        string sID;
        string sName;
public:
        Member(string id, string name){
                sID = id;
                sName = name;
        }
        // getter/setter functions for id, and name are available
        // do not implement them
};

class Library{
        Book *oBooksArray[1000];
        Member *oMembersArray[1000];

        // data members to track the size of the filled array
        int nBooksSize; // count for books
        int nMembersSize; // count for members

        Library(){
                nBooksSize = nMembersSize = -1; // initialized with -1

                // assign NULL to all the pointers
                for(int i = 0; i < 1000; i++){
                        oBooksArray[i] = NULL;
                        oMembersArray[i] = NULL;
                }
        }

        ~Library(){
                for(int i = 0; i < 1000; i++){
                        if(oBooksArray[i]) delete oBooksArray[i];
                        if(oMembersArray[i]) delete oMembersArray[i];
                }
        }
};

int main() {
        Library oLibrary;

        // creating two books
        Book oBook1("C++ How to Program", "11111-2222", 10);
```

```
        Book oBook2("C++ Black Book", "11111-2223", 15);

        oBook1--; // decrease the count of the available copies
        ++oBook2; // increase the count of the available copies

        // adding books in the library
        oLibrary += oBook1; // add book1 in the books' array at the last index
        oLibrary += oBook2; // same
        oLibrary = oLibrary - oBook2; // remove book2 from the array and adjust the array to
fill the free slot

        // creating two members
        Member oMember1("1", "Imam Ul Haq");
        Member oMember2("2", "Fakhar Zaman");

        // adding members to the library
        oLibrary += oMember1; // add member1 in the members' array at the last index
        oLibrary += oMember2; // same
       cout<<oBook1<<"\n"<<oBook2;

        return 0;
}
```