Lecture # 9 outline and homework (5-7-2023)

- **Review**
  - ✓ **Constant member functions**
  - ✓ **Initializer list**
  - ✓ **Static data members and functions**

Review 1:

//////////. . . . . code1.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);

	// .......  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) : Id(i), name(n), size(s)
{

	cout << "\nInside parameterized Constructor";

	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
}
```

Content:

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

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.

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.

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
  - in a constant member function we cannot call a non-constant member function of class

- 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 definations.
- **Remember**: Constructors / Destructors cannot be constant as they need to initialize variables, therefore modifying them.

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

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**)

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

Task: Read below class carefully and under all concepts that you have studied during lectures.

```
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 = 'N', int * = NULL);

	// .......  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
	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";

	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];
		}

	}
	Average = 10;
	player::calAverage();//calculating average in constructor
	count++;
}

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

player& player::calAverage(void)
{
	cout << count;
	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 = 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;
}

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

///.......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";

}


int main()
{
	// write your implementstion code here
	player p1(3, 'A', 4);
	p1.calAverage().print();///cascading call enabled by using this pointer see
defination of functions

}
```