National University of Computer & Emerging Sciences, Karachi

Computer Science Department

Spring 2023, Lab Manual – 05

Course Code: CL-1004	**Course: Object Oriented Programming Lab**

Outline

- Access modifiers
- Private, Public and Protected
- **“This”** Keyword
- **“Static”** Keyword
- **“Const”** Keyword
- Revision of classes and objects, constructors & destructors

Access modifiers

Access specifiers in C++ are keywords used to determine the visibility and accessibility of class members. The three access specifiers are public, private, and protected, and they control whether class members can be accessed from anywhere in the program, only within the class itself, or within the class and its derived classes

Private

In C++, the private access specifier is used to restrict the access of class members to only the class itself. This means that private data members and functions cannot be accessed or modified from outside the class. In C++, the default access specifier for class members is 'private' (In Java its Private-package), meaning that class members are private

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

class Person {
private:
	string name;
	int age;
public:
	string getName() {
		return name;
	}
	int getAge() {
		return age;
	}
};

int main() {
	Person p1;
	p1.name = "John"; // This will result in a compilation error since "name" is private
	p1.getAge(); // This is allowed since "getAge()" is a public member function
	return 0;
}
```

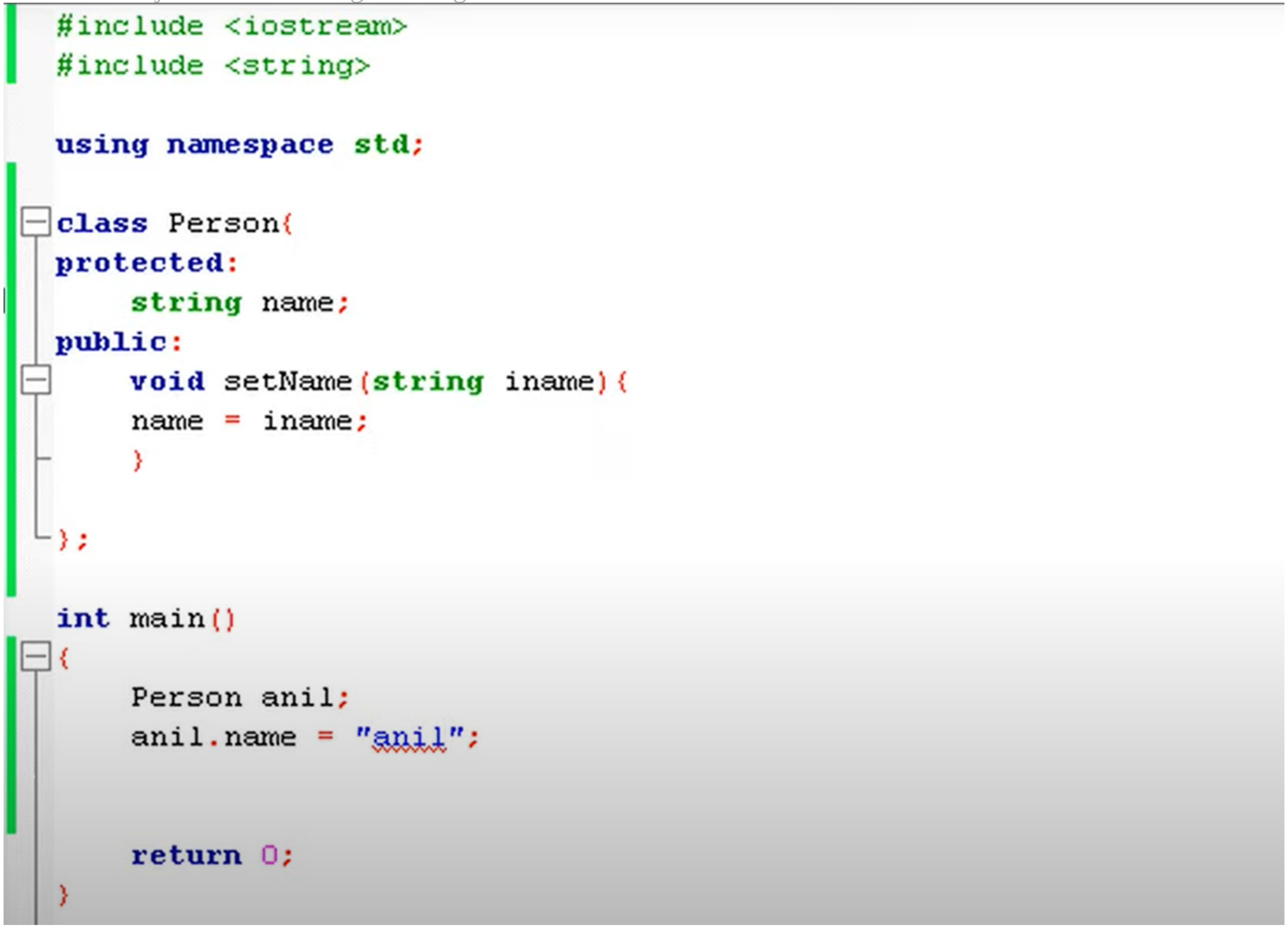
Public

The public access specifier is used to define class members that can be accessed and used by any part of the program. It means that the member functions and data can be accessed by objects of the class as well as other parts of the program

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

class Person {

public:
	string name;
	int age;
	string getName() {
		return name;
	}
	int getAge() {
		return age;
	}
};




int main() {
	Person p1;
	p1.name = "John";
	p1.getAge();
	return 0;
}
```

Protected:

The protected access specifier is way to mark some parts of a class as "semi-private". These parts are still hidden from the rest of the program, but are accessible to other classes that are closely related to the original class. This can be useful for sharing code between related classes, while still keeping some parts of the class hidden from the rest of the program.



```
#include <iostream>
#include <string>

using namespace std;

class Person{
protected:
    string name;
public:
    void setName(string iname){
    name = iname;
    }

};

int main()
{
    Person anil;
    anil.name = "anil";


    return 0;
}
```

This Keyword

This keyword is a pointer to the current object. It is used to refer to the member variables and member functions of the current object. It can be particularly useful when there is a naming conflict between a member variable and

a local variable or parameter with the same name in a member function. By using this, you can disambiguate between the two and access the member variable specifically.

```
#include <iostream>

using namespace std;

class MyClass {
private:
	int x;
public:
	void setX(int x) {
		this->x = x; // using 'this' to refer to the member variable
	}

	void printX() {
		cout << "The value of x is: " << this->x << endl; // using 'this' to refer to the
member variable
	}
};

int main() {
	MyClass obj;
	obj.setX(42);
	obj.printX();
	return 0;
}
```

In the above, the MyClass class has a private member variable x. The setX member function takes an integer parameter x and sets the value of the member variable to it using this keyword. The printX member function simply prints the value of the member variable to the console, again using the this keyword to refer to the member variable.

Static key word

Static in C++ OOP is used to declare class-level variables and member functions that are shared by all instances of the class. It means that the variable is initialized once and remains in memory throughout the execution of the program, and that the member function can be called directly on the class, rather than on an instance of the class.

```
#include <iostream>
using namespace std;
class Car {
public:
	static int count;
	Car() {
		count++;
	}
};

int Car::count = 0;

int main() {
	Car car1, car2, car3;
	cout << "Number of cars created: " << Car::count << endl;
	return 0;
}
```

In The above example, we have a Car class that has a static member variable called count. The static keyword indicates that this variable is shared among all instances of the class, rather than having a separate copy for each instance.

Const Keyword

In C++ OOP, const is a keyword used to make a variable or function "unchangeable" once it has been defined. It helps prevent accidental changes to the value of a variable or the state of an object, making your code more reliable and easier to understand.

```cpp
#include <iostream>
using namespace std;
class Circle {
public:
	const double PI = 3.14159;  // constant variable
	double radius;
	Circle(double r) {
		radius = r;
	}
	double getArea() const {  // const member function
		return PI * radius * radius;
	}
};

int main() {
	const Circle c(5);  // constant object
	c.radius = 10;  //Attempt to modify c's radius (this will result in a compile-time error)
	cout << "Area: " << c.getArea() << endl;
	return 0;
}
```

In The above example, the const keyword is used to create a constant variable PI, which cannot be modified after initialization. The const keyword is also used to declare a constant member function getArea(), which promises not to modify any non-static data members of the class

In "main function c object is declared as const. When you declare an object as const, it means that the object cannot be modified.

Working with Constructor and Destructor

Example 1

```
#include <iostream>
using namespace std;
class Test
{
public:
	// constructor
	Test()
	{
		cout << "Constructor called" << endl;
	}

	// destructor
	~Test()
	{
		cout << "Destructor called" << endl;
	}
};

int main()
{
	Test t1; // Constructor Called
	int a = 1;
	if (a == 1)
	{
		Test t2; // Constructor Called
	} // Destructor Called for t1
} // Destructor called for
```

In The above example we have a constructor and a destructor. The constructor is called when an object of the class is created and the destructor is called when the object is destroyed.

In the "main" function, two objects of the "Test" class are created - "t1" and "t2". The constructor is called for both objects when they are created. The object "t2" is created inside an if statement and only exists within the scope of that if statement. When the if statement ends, the object "t2" is destroyed and its destructor is called. At the end of the "main" function, the object "t1" is destroyed and its destructor is called.

Example 2

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


class Test2 {
public:
	Test2() {
		cout << "Constructor called" << endl;
	}
	~Test2() {
		cout << "Destructor called" << endl;
	}
};



int main() {
	cout << "Before creating the object" << endl;
	Test2 obj;
	cout << "After creating the object" << endl;
	return 0;
}
```

In the above Example the constructor is called when the object is created, and the destructor is called when the object is destroyed. In this case, the destructor is called automatically when the object goes out of scope at the end of the "main" function

Example 3

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

class Test3 {
private:
	int* value;
public:
	Test3(int val) {
		value = new int(val);
		cout << "Object created with value " << *value << endl;
	}
	~Test3() {
		cout << "Object destroyed with value " << *value << endl;
		delete value;
	}
};



int main() {
	Test3 obj(42);
	return 0;
}
```

This program defines a class called "Test3" with a constructor that takes an integer value and a destructor that deallocates the memory associated with that value when the object is destroyed. The "main" function creates an object of the "Test3" class with a value of 42, which is printed to the console when the object is created. When the object goes out of scope and is destroyed, the destructor is automatically called, and a message indicating the value of the destroyed object is printed to the console

Example 4

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

class Test4 {
private:
	int* myArray;
	int mySize;
public:
	// Constructor
	Test4(int size) {
		myArray = new int[size];
		mySize = size;
		cout << "Constructor called" << endl;
	}

	// Copy Constructor
	Test4(const Test4& other) {
		mySize = other.mySize;
		myArray = new int[mySize];
		for (int i = 0; i < mySize; i++) {
			myArray[i] = other.myArray[i];
		}
		cout << "Copy constructor called" << endl;
	}

	// Destructor
	~Test4() {
		delete[] myArray;
		cout << "Destructor called" << endl;
	}
};

int main() {
	Test4 obj1(5); // Constructor called
	Test4 obj2(obj1); // Copy constructor called
	return 0; // Destructor called twice (for obj2 and obj1)
}
```

In this example, the class "Test4" has a constructor, copy constructor, and destructor. The constructor allocates an array of integers and initializes the "mySize" member variable. The copy constructor creates a new object that is a copy of an existing object, by allocating a new array and copying the values from the original array. The destructor deallocates the memory used by the array.

In the "main" function, two objects of " Test4" are created: "obj1" using the constructor and "obj2" using the copy constructor. When the program finishes and the objects go out of scope, their destructors are called, which deallocates the memory used by the arrays and prints a message to the console.

LAB TASKS

1. Write a program in which a class named EMPLOYEE has private member variables named EMP_ID, EMP_DESIGNATION , EMP_PINCODE. Use a public function to initialize the variables and print all data.

2. Find out and specify where and why the static keyword should be used, and rectify if the program has any errors.

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

class Samsung{
private:
string ph_name;
public:
void name(){
cout << "Phone: "<< ph_name;
}

void set_name(string name){
ph_name = name;
}
};

//Initializing private static member
string Samsung::phone_name = "";

int main()
{
//no object has been created
//accessing static function directly with class name
Nokia::set_name("Samsung 2600");
Nokia::name();

return 0;
}
```

3. Write a program of your own in which you demonstrate the concept of constant keyword.

4. Where this-> operator must be used in the following program and why?

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

class Abc
{
string name;

public:
Abc(string name)
{

name = name;
}
void display()
{
cout << name << endl;
}
};

// Driver code
int main()
{
Abc gfg("GeeksforGeeks");
gfg.display();
return 0;
}
```

5. Make a **Rectangle class,** calculate its Length and Breadth , create a constructor and destructor for the same class.