EECS168/169-Lab9

Classes & Objects
Object Oriented Programming
OOP

Classes – Public vs Private

Object oriented programming (OOP) languages allows programmers to create user-defined data types by creating **classes**. Instances of these data types are **objects**. Objects may contain **instance variables(member)** and **methods** defined by the programmer.

Public Scope VS Private Scope

- The variable and methods that belong to a class can be either public or private. Whoever is using your class (e.g. main or another class) can only access the public members of the class.

- Each class is defined in a separate cpp file. In addition, each class needs a separate header file.

Class

Block of code with variables & functions with a name

name {
//block of code with variables & functions
}

- Syntax:

```
class Circle {

    double radius;

public:
    double diameter();
    double area();
    double circumference();
    double getRadius();
    void setRadius(double r);
};
```

- Declare variables of type Circle, e.g. in your main:
- `Circle circle_1, circle_2; // circle_1, circle_2 are OBJECTS of class Circle`
- Accessing PUBLIC member variables & functions from class variables or objects or instances
- `circle_1.diameter();`

MakeFile – case of MyClass

Lots of files, variables, functions!

Files: class header file, class definition file, main.cpp

Makefile update:

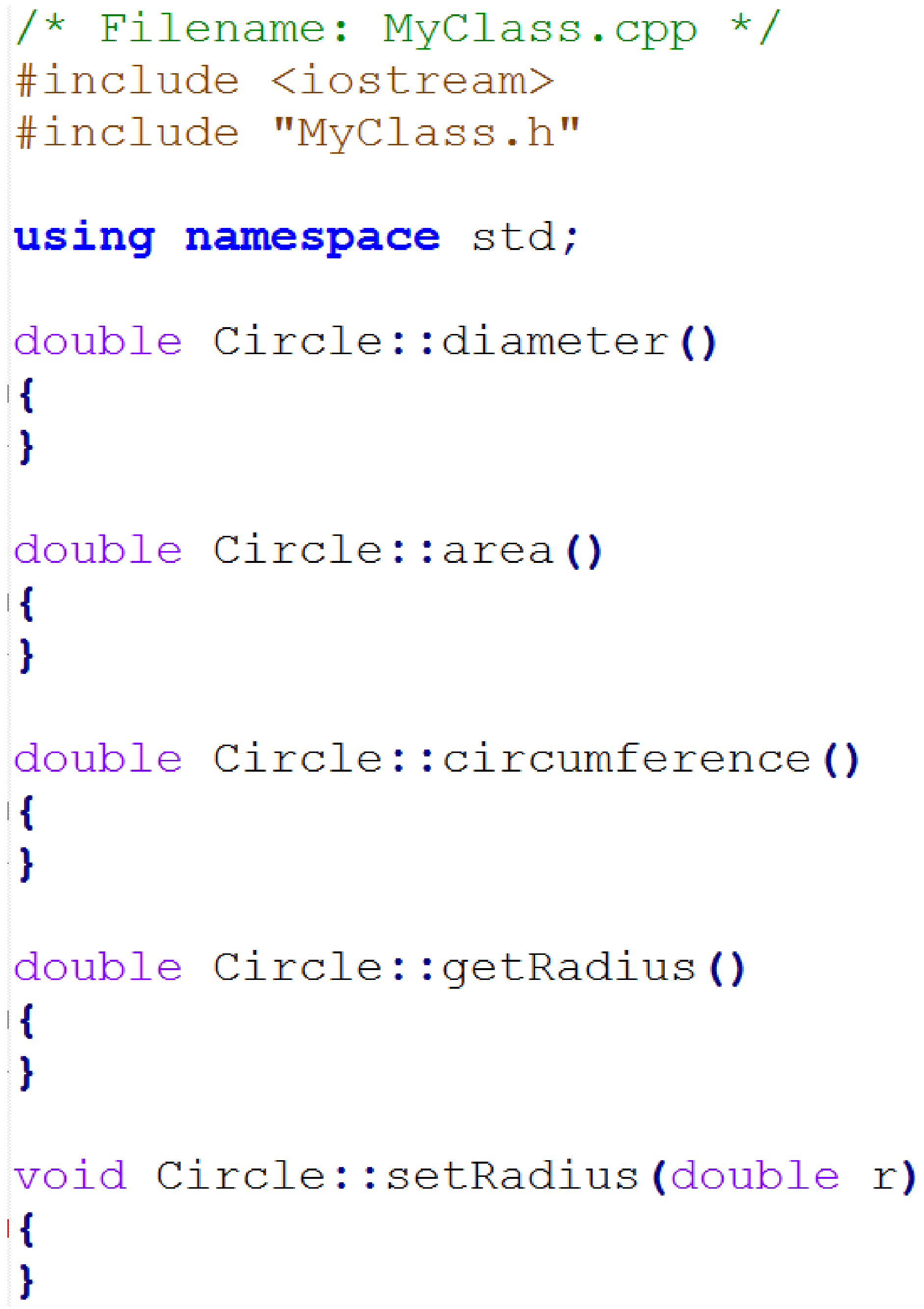
```
Circle: main.o MyClass.o
	g++ -std=c++11 -g -Wall main.o MyClass.o -o Circle
main.o: main.cpp MyClass.h
	g++ -std=c++11 -g -Wall -c main.cpp
MyClass.o: MyClass.h MyClass.cpp
	g++ -std=c++11 -g -Wall -c MyClass.cpp
clean:
	rm *.o Circle
```

Code the skeletal of files – 1. MyClass.h , 2. MyClass.cpp and finally 3. main.cpp

Circle Example - MyClass.h

```
#ifndef _MyClass
#define _MyClass

 /* Filename: MyClass.h */
 using namespace std;

 class Circle {

     double radius;

 public:
     double diameter();
     double area();
     double circumference();
     double getRadius();
     void setRadius(double r);
 };

#endif
```

Circle Example - MyClass.cpp



```cpp
/* Filename: MyClass.cpp */
#include <iostream>
#include "MyClass.h"

using namespace std;

double Circle::diameter()
{
}

double Circle::area()
{
}

double Circle::circumference()
{
}

double Circle::getRadius()
{
}

void Circle::setRadius(double r)
{
}
```

Summary – Example with one class

```cpp
// in myclass.h

class MyClass
{
public:
  void foo();
  int bar;
};
```

```cpp
// in myclass.cpp
#include "myclass.h"

void MyClass::foo()
{
}
```

```cpp
//in main.cpp
#include "myclass.h"  // defines MyClass

int main()
{
  MyClass a; // no longer produces an error, because MyClass is defined
  return 0;
}
```

Exercise – Employee Class

private members

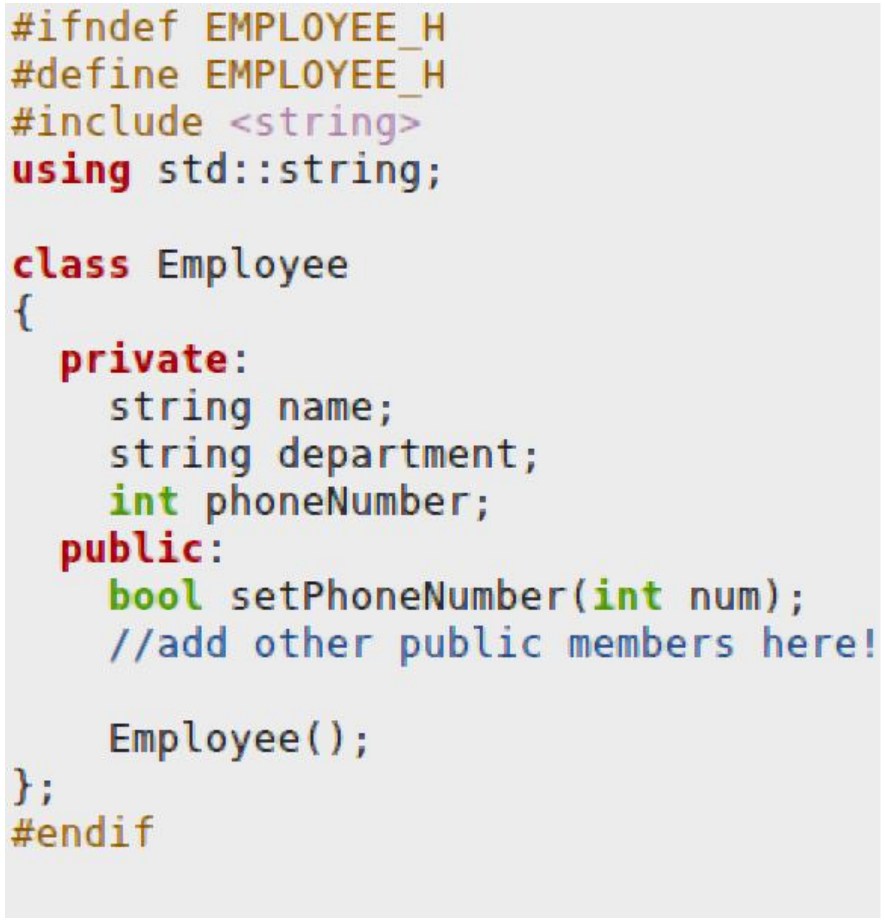
- int phoneNumber; //must be 7-digits
- std::string name; //cannot be the empty string
- std::string department; //must be a valid department code (see below)
  - All department codes are case-sensitive and these are the only valid codes:
  - "MARKETING"
  - "R&D"
  - "GLOBAL"
- double salary; //cannot be negative
- You may also create some private helper method to do things like...
- count the number of digits in a phone number

public members

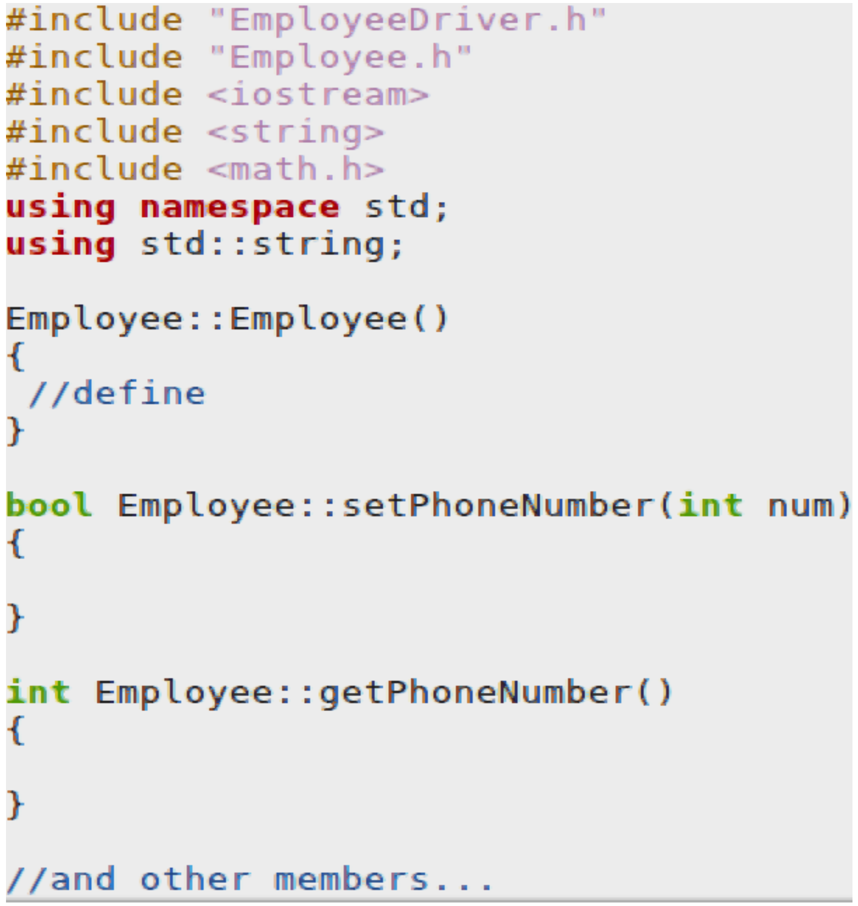
- The following public setters should return false and NOT set the value if the parameter passed in is invalid
- Return true and set the value if the parameter is valid
  - bool setPhoneNumber(int num)
  - bool setName(std::string name)
  - bool setDepartment(std::string dept)
  - bool setSalary(double salary)
- bool isSameDept(const Employee& otherEmployee)
  - Take another Employee by const reference (see more notes below)
  - Returns true if the other Employee works in the same department; return false otherwise

<u>NOTE: The Employee class should not do any input or output.</u>

Employee.cpp and Employee.h



```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
#include <string>
using std::string;

class Employee
{
  private:
    string name;
    string department;
    int phoneNumber;
  public:
    bool setPhoneNumber(int num);
    //add other public members here!

    Employee();
};
#endif
```



```
#include "EmployeeDriver.h"
#include "Employee.h"
#include <iostream>
#include <string>
#include <math.h>
using namespace std;
using std::string;

Employee::Employee()
{
 //define
}

bool Employee::setPhoneNumber(int num)
{

}

int Employee::getPhoneNumber()
{

}

//and other members...
```

main.cpp and EmployeeDriver.h

EmployeeDriver class

- private members
  - **Employee emp1;**
  - **Employee emp2;**
  - **void obtainEmployee()**
    - Talk with the user to obtain the data needed to create two employees
    - It does not validate the values, but rather checks the return value from a call to Employee's methods
  - **void printEmployeeInfo()**
    - Prints the following information about each of the Employees to the screen:Their names, phone numbers, salaries, and departments
    - Lastly print whether or not the two Employee work in the same department
      - Again, you must use the Employee's methods to discern this, not just check the department strings locally

- public members
  - void run()
    - run merely calls all the other methods
- Next, define the header members in the driver cpp file.

main does very little:

```
int main() {
EmployeeDriver EmpD;
EmpD.run();
return(0); }
```

The only files you submit should be your Makefile, cpp files and header files (.h)

Exercise 169

- Make the following additions to your Employee class:
- An employee's name must...
  - Start with a capital letter
  - Must have at least one space (to indicate the separation between first and last name)
  - Contain only letters and spaces (a person can have more than just first and last names e.g. Sir Patrick Stewart)
- Make the following additions to your EmployeeDriver class:
  - The two employees cannot have the same phone number
  - Always print the employee with the smaller salary first

Pass by Constant Reference

- Passing by value is expensive, as the compiler must often
  - allocate a temporary local variable of the type,
  - copy the bytes of the argument to the temporary,
  - pass a pointer to the temporary into the function,
  - access the bytes of the parameter indirectly, and
  - deallocate the temporary on return.
- Solution: pass objects by const reference
- The performance difference, coupled with the convenience, has resulted in an automatic tendency of programmers to pass classes by const reference.

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3445.html

Passing to Functions in C++

Pass by value:

- to prevent the function from modifying the original variable as well as to prevent other threads from modifying its value while the function is being executed.
- Extra memory spent to copy the object.

Pass by const reference:

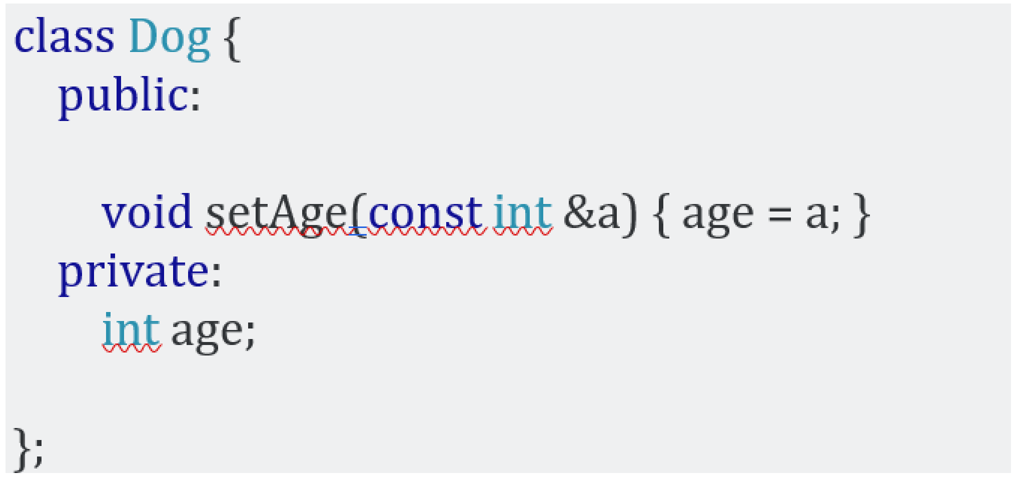
- The function gets read access to the original object, but cannot modify its value.
- Any change made to the original object by another thread will show up inside the function while it's still executing.

Pass by non-const reference:

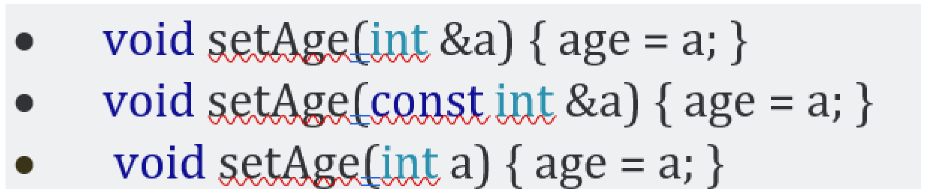
- Use this when the function has to write back some value to the variable, which will ultimately get used by the caller.

https://stackoverflow.com/questions/2139224/how-to-pass-objects-to-functions-in-c

Example



```
class Dog {
  public:

    void setAge(const int &a) { age = a; }
  private:
    int age;

};
```



```
•    void setAge(int &a) { age = a; }
•    void setAge(const int &a) { age = a; }
•     void setAge(int a) { age = a; }
```

https://stackoverflow.com/questions/30558077/passing-const-references-to-functions