UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 9: Class Inheritance and introduction to libraries

mihasan@liverpool.ac.uk

Previous lecture

- Operator overloading for classes has been discussed.
- The concept of polymorphism has been defined as part of OOP.
- The two-file implementation of classes has been discussed.
- Class templates were discussed.

This lecture

- What is covered in this lecture?

  1. Classes inheritance
  2. Introduction to C++ libraries

- Why it is covered?

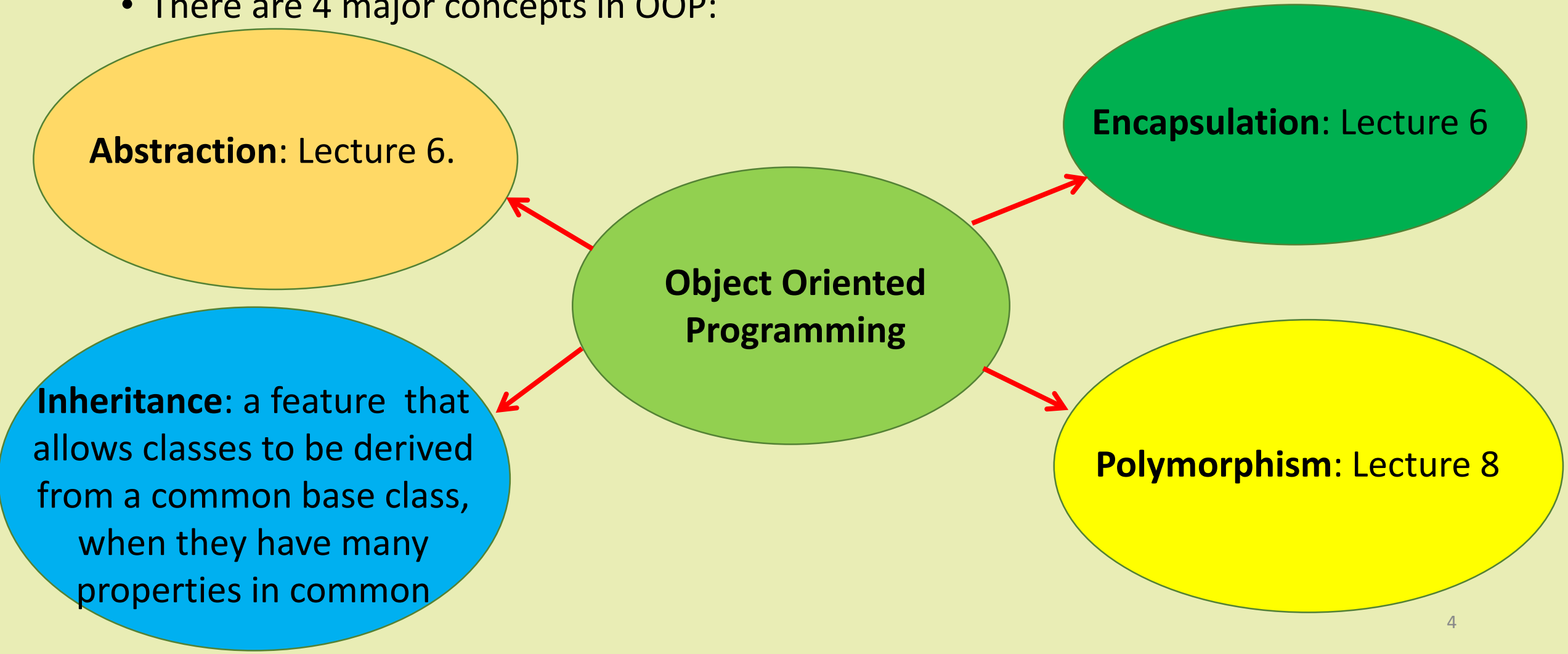
  1. Class inheritance is a standard procedure in application development in C++.
  2. GUI application development is done on Library level.

- How are topics covered in this lecture:

  3 source codes and a live demonstration.

Object Oriented concepts revisited

- There are 4 major concepts in OOP:



Class Inheritance

- When discussing class inheritance, classes are divided into two types:

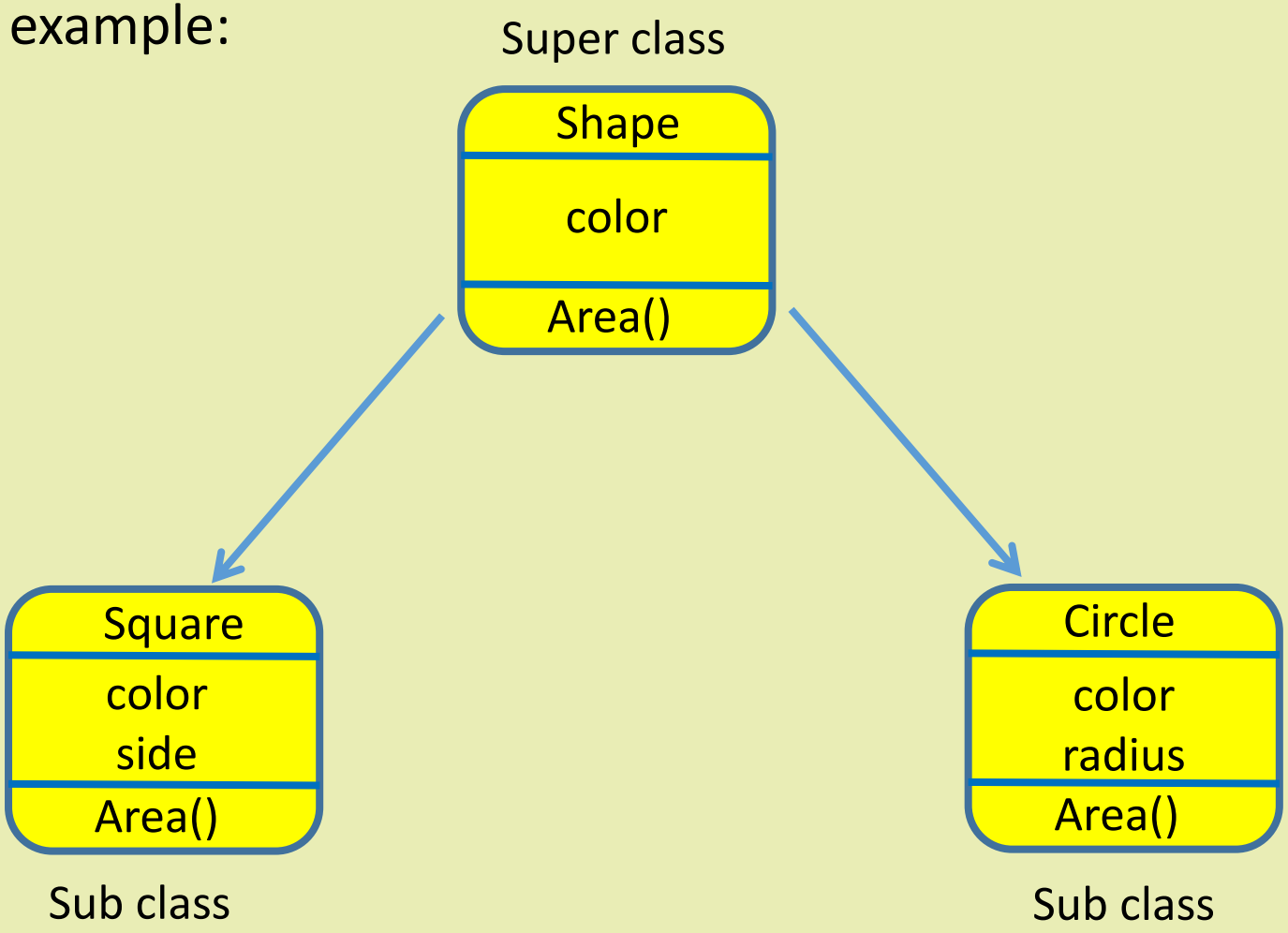
1. Derived/Sub class: the class that inherits properties from another class.

2. Base/Super class: the class whose properties are inherited by sub class.

- A sub class has the same properties as the super class, in addition to special added properties.

Class Inheritance

- A conceptual example:



Super class

- Any class can be a super class.
- Inheritance can move through several levels:

Shape → Rectangle → Square

- What do classes exactly inherit?

```
class Super {
//Constructor
//Destructor
//Data members
//function members
//operator=
// friend functions
//other overloaded operators
};
```

→

```
class Sub {
//Constructor ✗
//Destructor ✗
//Data members
//function members
//operator= ✗
// friend functions ✗
//other overloaded operators
// OWN data or function members
};
```

Class inheritance syntax

- Syntax:

```
class Sub_class : access_specifier Super_class {
/*statements*/  };
```

- Example:

```
#include "Box.h"

class MyBox : public Box {  /* Statements */ };

class MyotherBox : protected Box {  /* Statements */ };

class MythirdBox : private Box {  /* Statements */ };
```

- Accessibility depends on type of derivation!

Access specifiers revisited

- Access specifiers control access to class members from outside the class:

1. `public:` allows access from outside the class.
2. `private:` gives access only within the class.
3. `protected:` gives access within the class and to sub classes.

- One can control the accessibility of the sub classes by using the appropriate access specifier.
- The constructor of the sub class can make use of the constructor of the super class.
- Go to L9D1.cpp

Access specifiers revisited

```
class A {
 public:
    int x;
 protected:
    int y;
 private:
    int z;
};
```

```
class B : public A
{
 // x is public
 // y is protected
 // z is not accessible from B
};
```

```
class C : protected A
{
 // x is protected
 // y is protected
 // z is not accessible from C
};
```

```
class D : private A
{
 // x is private
 // y is private
 // z is not accessible from D
};
```

Preventing class derivation

- To prevent further derivation of a given class, the keyword "**final**" is used when the class is defined.
- Example:

```
#include "Box.h"
class Cube final : public Box {  /* Statements */ };
class MyCube : protected Cube {}; //Error!!
```

- This is a feature in C++ 11 and subsequent releases.

Virtual functions

- Occasionally one needs to change the definition of an inherited function without changing its name, parameters or return type.
- The keyword "**`virtual`**" indicates that a function should be defined in a sub class.
- It is vital when superclass pointers are used with subclass objects.
- The "**`final`**" keyword will prevent further overriding of "**`virtual`**" functions.
- The "**`override`**" keyword can be used to verify that a sub class function overrides a virtual function in the super class.
- Go to L9D2.cpp

C++ Libraries

- A C++ library is a composition of functions and classes written in template form.
- Most libraries are highly specialised.

Lectures 10-18

- Standard libraries:

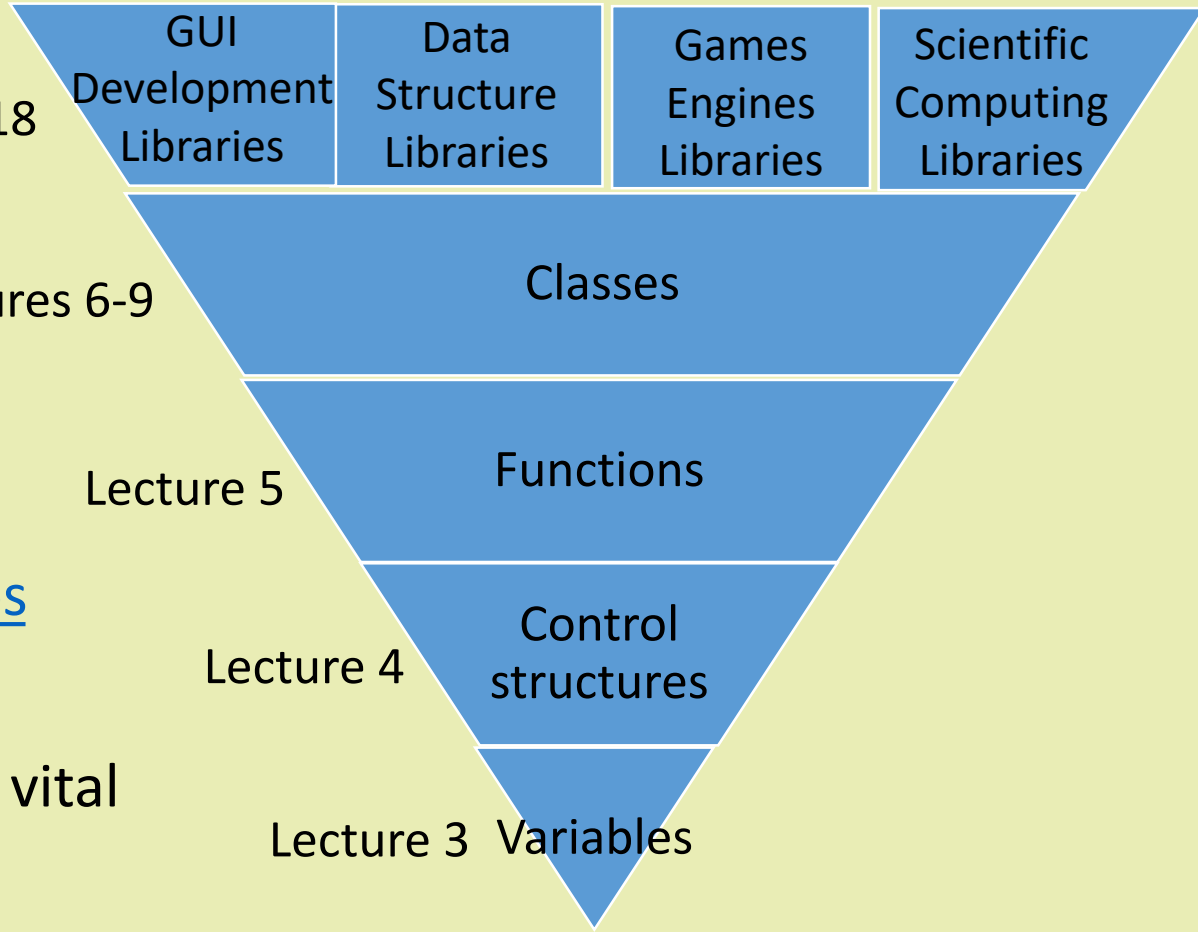
http://www.cplusplus.com/reference/

Lectures 6-9

- Non-standard:

https://en.cppreference.com/w/cpp/links/libs

- Efficient use of a library's documentation is vital to maximise its use.



Linking libraries to a code

- Libraries can be categorized into static libraries and dynamic libraries.
- Static libraries (has .lib extension) create a copy of their code into the code under development, at compile time, leading to large executable file with minimal dependencies.
- Dynamic libraries (has .dll extension) do not copy the code, they are linked at run time to the main executable, leading to small executable file but with many dependencies.
- Whether the linking is static or dynamic, the header of the library must be included in the source code.
- Demonstration: http://glew.sourceforge.net/

Linking libraries to a code

- First step: consult the library's documentation on how to install/use.
- Assuming Windows binaries are available for a given library:

1. Copy the headers of the library into a "header" folder.
2. Copy the object files (have extension of .lib) into a "libraries" folder.
3. Copy any dynamic libraries (have extension of .dll) into the source codes folder.
4. Configure VS dependencies and linker to connect to the folders mentioned above.

- Go to L9D3.cpp

Practical note: When adding a 3rd party library check what are its requirements before adding it to your project.

Summary

- The concept of class Inheritance was discussed.
- Different types of class inherence were discussed.
- An introduction to C++ libraries was given, and the different types of linking were discussed.