UNIVERSITY OF LIVERPOOL

Application Development with C++ (ELEC362)

Lecture 8: Even more classes

mihasan@liverpool.ac.uk

Previous lecture

- The definitions and functionalities of constructors and destructors of classes were discussed.

- The following keywords and their use were discussed:

Keyword	Use
`explicit`	Prevents the compiler from implicit casting when one-argument constructor is called.
`static`	Defines a static data member (common among all objects of a given class).
`friend`	Gives access to private data members of a class for a non-member function.
`this`	A default pointer pointing to the object that is making a function call.

This lecture

- What is covered in this lecture?

  1. Polymorphism. 2. Practical aspects of working with classes

- Why it is covered?

  Polymorphism greatly simplifies object-object interactions.

- How are topics covered in this lecture:

  3 source codes and a demonstration.

Operators revisited

- Operators are used in arithmetic and Boolean operations.
- Consider the following code:

```
//Primitive datatypes:
int x = 1, y = 2,z;
double a = 1.5, b = 2.5,c;
z = x + y;
c = a + b;
//User-defined datatypes:
Box B1(2,3,4), B2(1,1,1),B3;
B3 = B1 + B2; //??
```

- We need to define what “adding” two boxes means!

Operator overloading

- The functionality of all operators encountered so far (+ , -, *, /, <, >, = , ==, !=, <<, ++) and many more can be extended to classes.

- To define a function representing an operator action, the keyword "`operator`" is used such that:

a + b ⟷ operator+(a,b)

a - b ⟷ operator-(a,b)

a/b ⟷ operator/(a,b)

- Go to L8D1.cpp

Overloading assignment operator

- The assignment operator "=" copies the content of the right object to the left one.
- Example

```
class Ratio {
private:
    int num, denom;
public:
    Ratio (int n,int d) {num=n; denom=d;}
    void operator=(Ratio& a) { num=a.num; denom=a.denom;}
};
void main () {
Ratio x(1,2), y(4,5);
y = x; // Which function is called here?
Ratio z = x;}// Which function is called here?
```

A reference to avoid creating a copy object

Overloading stream operator

- Stream operator “<<“ can be used to output/input data of a basic data types.
- The functionality of this operator can be extended to classes by operator overloading.
- The output and the input datatypes of an overloaded stream function must be passed by reference, to enforce the continuity of the output stream of the programme.
- Overloading the “<<“ operator is done using a friend function, this is because it requires access to “ostream” object, which lies outside the object it is calling the function for.
- Go to L8D2.cpp

Object Oriented concepts revisited

- There are 4 major concepts in OOP:

Abstraction: Lecture 6.

Encapsulation: Lecture 6

Object Oriented Programming

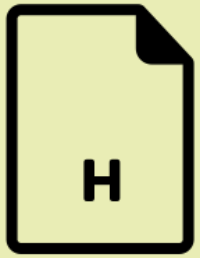
Inheritance: Lecture 9

Polymorphism: Using the same interface/function to perform different operations based on the input objects

Function and operator overloading are implementations of polymorphism concept

Working with classes

- In real codes there can be hundreds of classes, which makes working with classes difficult and messy.
- This can be simplified by working with header and source files instead of keeping everything in source file:



Header file: contains data members and functions declarations/prototypes

Class.h

Source file: contains function definitions/bodies

Class.cpp

The two-files implementation of classes

Box.h

```
class Box{

private:
        int num, denom;
public:
       Ratio (int n,int d);
       double Decimal(int n,int d);
}
```

Box.cpp

```
#include "Box.h"


Box::Ratio(int n, int d) {
         num=n; denom=d;}

double Box::Decimal(int n,int d){
        return double(n)/ double(d);}
```

- Visual Studio can generate the two files automatically (demonstration).

Practical note: Always follow the 2-file implementation of classes

Header files revisited

- One header file can contain multiple classes definitions.

- When writing a code depending on those classes, don’t forget to include the header file.

- Real life code example : https://www.dealii.org/current/doxygen/deal.II/step_24.html

https://www.dealii.org/current/doxygen/deal.II/full__matrix_8h_source.html

Class templates

- Similar to functions, classes can be derived from templates to work with different data types.
- Syntax:

```
template <class a_type>
class class_name {

private:
   a_type var;

public:
  class_name() {var=1}; // just an example
  a_type Out(a_type a) {/*function definition*/};
};
```

- Go to L8D3.cpp

Class alias

- When using class templates, it is inconvenient to keep re-using the template name when defining a new object.
- Class alias can be used to create a short name of a class derived from a template, by using either "using" or "typedef".
- Example:

```
// declare BoxSamples as alias for a sample of 'boxes'

 using BoxSamples = Samples<Box> ;

 typedef SampleBoxes = Samples<Box> ; // More common in modern C++

 // declare an array of 5 boxes in a sample class
 BoxSamples myBoxes[5];

 SampleBoxes mySBoxes[5];
```

Summary

- 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.