Classes in C++

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand how to use classes in C++
2. Demonstrate the ability to use classes in C++

Headers and Source Files

Headers and Source Files (1)

- C++ classes are split into two files:
  - header file, with a `.h` file extension
  - source file, with a `.cpp` file extension
- **Header** file consists of the class definitions and function headers
- **Source** file consists of the implementation of the functions
- If the class implementation does not change, then the class will not need to be recompiled
  - only classes that change will be recompiled

Header and Source Files (2)

Header Files i

- These are files that end with the `.h` file extension
- Contains the declaration of the following:
  - variables
  - function headers
- These files are referenced to in the C++ implementation source file using `#include`
  - i.e. `#include "myfile.h"`
- The files for a header are plain-text and contain rules for defining syntax
- Header files are useful for the following reasons:
  1. Speeds up compilation
  2. Keeps the code organised
  3. Allows for the separation of the interface from the implementation

Header and Source Files (3)

Header Files ii

- Header files can cause complex errors
- Generally when multiple header declarations from other files are included in the same file
  - this will raise a **compiler error**
  - it can be avoided by using *header guards*

```
#ifndef UNIQUE_NAME
#define UNIQUE_NAME
...
#endif
```

- Preprocessing on compilation will check for header guards and their unique name
  - if the header is repeatedly included in the same file, then the contents will be ignored

Header and Source Files (4)

Source Files

- These are files that end with the `.cpp` file extension
- Known to be the implementation file
  - contains the implementation of the functions declared in the header file
- Source files help split the interface from the implementation

Header and Source Files (5)

Example: Header File

```
#ifndef EXAMPLE_FILE_H
#define EXAMPLE_FILE_H
#include <iostream>
#include <string>

class ExampleClass {
public:
  int intExample1;
  char get_char();
  string say_hello(int tmpInt);
private:
  char charExample1;
}

#endif //EXAMPLE_FILE_H
```

Example: Source File

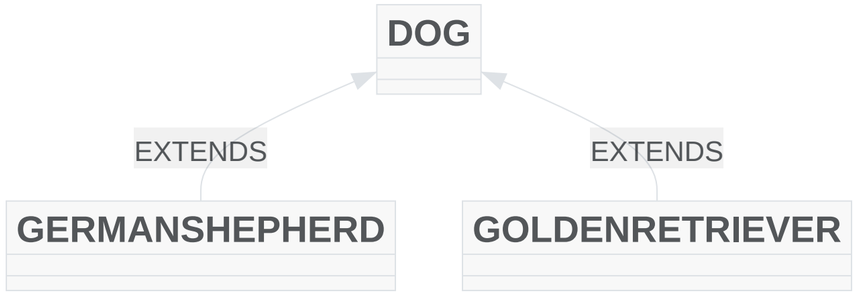
```
#include "myfile.h"

std::string say_hello(int tmpExample) {
  return "Hello " + std::to_string(tmpExample);
}

char get_char() {
  return charExample1;
}
```

Classes

Classes (1)

Recap: Classes

- **Recap**:
- Classes provide a structure for the objects
- They are used for defining:
  - a set of properties, represented by variables
  - the behaviour, which is represented by functions



Classes (2)

- Creating classes in C++ is different to Python
- Source code is split between two files:
  - header (`.h`)
  - source (`.cpp`)
- The **header** contains information about the *variables* and *functions*
- The **source** contains the actual implementation of the functions
- Classes will be defined using the `class` keyword followed by a name
- The body of a class is defined inside a set of curly brackets (`{}`) and terminated by a semicolon (`;`)

```
class Student {
  ...
};
```

Classes (3)

Access Specifiers

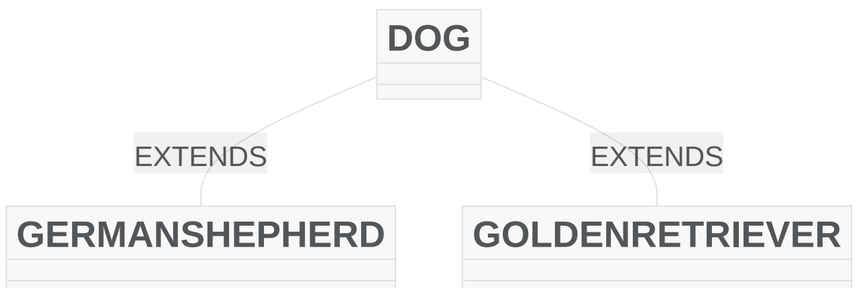
- Access specifiers control how members of a class are accessed
- There are three access specifiers in C++:
  - `public`: members are accessible outside the class
  - `private`: members cannot be accessed (or viewed) outside the class
  - `protected`: members cannot be accessed outside the class; *but* can be accessed with inheritance

```
class Lecturer {
  public:
    std::string name;
  private:
    int age;
  protected:
    std::string dateOfBirth;
};
```

Classes (4)

Creating an Object i

- **Recap**:
  - Objects are instances that are created from a class
  - Objects created from classes will be referred as *instance(s)*
    - the process of creating an object from a class is instantiation
  - Objects will have a property
    - a set of values that are associated with a real-world entity
  - For example:
    - **Class**: Dog
    - **Objects**: German Shepherd, Golden Retriever



Classes (5)

Creating an Object ii

- An object can be created from the class by calling upon the class name
  - followed by the name of the variable, i.e. `Lectuer lecturer1;`

```
#include <iostream>
class Lecturer {
  public:
    std::string name;
  private:
    int age;
};
int main() {
  Lecturer lecturer1;
  Lecturer lecturer2;
  lecturer1.name = "Ian Cornelius";
  lecturer2.name = "Terry Richards";
```

```
lecturer1.name -> Ian Cornelius
lecturer2.name -> Terry Richards
```

Classes (6)

Class Constructors

- **Recap**:
  - Constructors are a special type of function and is called automatically when an object is created
  - They have the same name as the class and do not return a type

```
class Lecturer {
  public:
    Lecturer() {
      ...
    }
};
```

Classes (7)

Default Constructors

- Constructors with no parameters are known as a *default constructor*
- A class `Lecturer` has been created
  - the default constructor sets the `name` and `age` of the lecturer
- An object is created and the variable name is accessed
  - the name of the lecturer is returned
  - the age of the lecturer *cannot* be accessed as it is **private**

```
#include <iostream>
class Lecturer {
  public:
    std::string name;
    Lecturer() {
      name = "Ian Cornelius";
      age = 34;
    }
  private:
    int age;
};
int main() {
```

```
lecturer1.name -> Ian Cornelius
```

Classes (8)

Parameterised Constructors

- Constructors with parameters are known as a *parameterised constructor*
- A class `Lecturer` has been created
  - the parameterised constructor sets the `name` and `age` of the lecturer
- The variable name is accessed
  - the name of the lecturer is returned
  - the age of the lecturer *cannot* be accessed as it is **private**

```
#include <iostream>
class Lecturer {
  public:
    std::string name;
    Lecturer(std::string _name, int _age) {
      name = _name;
      age = _age;
    }
  private:
    int age;
};
int main() {
```

```
lecturer1.name -> Ian Cornelius
lecturer2.name -> Terry Richards
```

Classes (9)

Class Functions

- Classes can also consist of functions, and they will belong to the object that is created
- A class `Lecturer` has been created
  - the parameterised constructor sets the `name` and `age` of the student
- The function `greeting()` is accessed on the `lecturer1` object

```
#include <iostream>
class Lecturer {
  public:
    std::string name;
    Lecturer(std::string _name, int _age) {
      name = _name;
      age = _age;
    }
    std::string greeting() {
      return "Hello " + name + ", and welcome to 5062CEM!";
    }
  private:
```

```
lecturer1.greeting() -> Hello Ian Cornelius, and welcome to 5062CEM!
```

Classes (10)

Accessing and Modifying Private Members i

- The variable `age` in this class is *private*
- There is no method of accessing this variable outside the class
  - therefore, a method of accessing and modifying this variable is required
- It can be achieved by creating a function inside the class

```
class Lecturer {
  public:
    std::string name;
    Lecturer(std::string _name, int _age) {
      name = _name;
      age = _age;
    }
  private:
    int age;
};
```

Classes (11)

Accessing and Modifying Private Members ii

- A class `Lecturer` has been created
  - the parameterised constructor sets the `name` and `age` of the student
- The function `get_age()` is accessed on the `lecturer1` object
  - access the **private** variable to return the age of the student
- The function `change_age()` is accessed on the `lecturer` object
  - changes the **private** variable to a new age

```
#include <iostream>
class Lecturer {
  public:
    std::string name;
    Lecturer(std::string _name, int _age) {
      name = _name;
      age = _age;
    }
    int get_age() {
      return age;
    }
    void change_age(int _age) {
```

```
lecturer1.get_age() -> 34
lecturer1.get_age() -> -9
```

Classes (12)

Class Inheritance i

- **Recap**:
- Inheritance allows you to define a class that will inherit all the functions and properties from another class
- There are two important terminologies to know:
  - **parent class** which is the class from which another class is being inherited from
  - **child class** which is the class that is inherited from the parent class
- Inheriting a class in C++ can be achieved with the colon (`:`) character

```
class childClass : accessSpecifier parentClass {
...
}
```

Classes (13)

Creating a Parent and Child Class

- Any class you create in C++ can be a parent class
  - the syntax is the same as creating any other class
- A child class is created by include the parent class after the colon when creating the child class
  - the child class will inherit all properties and functions of the parent class

```
#include <iostream>
// Parent Class
class Person {
  public:
    std::string name;
    Person(std::string _name, int _age) {
      name = _name;
      age = _age;
    }
  protected:
    int age;
};
```

```
lecturer1.name -> Ian Cornelius
lecturer1.get_age() -> 34
```

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk