• Coursework 2 due in 5pm today
• First long project set tomorrow
• Material in this lecture covered in example sheet 4
C++ classes

- The purpose of classes
- Access specifiers
- Constructors
- Inheritance
Global variables

Recall that if a variable is to be used in more than one function, it must be a *global variable*, defined outside of both functions

```cpp
// output is a global variable
std::ofstream outputFile;

void DoSomething()
{
    // code here
    outputFile << x << " " << y << std::endl;
    // code here
}

int main()
{
    outputFile.open("my_output_file.txt");
    DoSomething();
    outputFile.close();
}
Global variables

This situation is problematic:

- Large programs end up with *lots* of global variables
- Not clear which functions modify which global variables
- Only one copy of each global variable allowed

A solution:

- The C++ solution to these problems is *classes*
- Classes facilitate writing code in an *object-oriented* way
Classes

Classes are a mechanism used in C++ to:

- Collect together (and limit access to) variables and the functions that operate on them
  - Limiting the ways in which variables can be accessed or changed makes programs simpler to understand
- Define new types
  - For example, we could write a `Matrix` class, which would be used like `int`, `double` etc.
  - We can customise how operators (+, * etc.) act on class types
- Allow existing code to execute new code without modification
  - For example, we might want to use an existing function for numerical integration on a new integrand.
  - How can the existing integration code call the new function which evaluates the new integrand?
- See example sheet 3, question 3.3
Classes

We have already seen several examples of classes from the standard library:

- `std::vector`, a class which stores an indexed list of values
- `std::ofstream`, a class representing a file on disk

In the variable definition:

```cpp
std::vector<double> myVector;
```

- The class `std::vector<double>` is the type, representing the concept of a vector of doubles
- The variable `myVector` is a particular instance of the class `std::vector<double>`, also referred to as an object of type `std::vector<double>`
We define a class (which stores a point \((x, y) \in \mathbb{R}^2\)) as follows:

```cpp
class Point2D // define a class with the name "Point2D"
{
public: // an 'access specifier'
    double x, y; // class member variables
    double DistanceFromOrigin() // class member function
    {
        return std::sqrt(x*x + y*y); // std::sqrt from <cmath>
    }
}; // remember the semicolon at the end of a class definition
```

The variables and functions inside a class are called its *members*.

We declare (or *instantiate*) objects of type `Point2D` in the same way that we declare other variables:

```cpp
Point2D a, b;
```
Using classes

Class member variables and member functions are accessed through the . (dot) operator:

```cpp
Point2D a, b;
a.x = 3; a.y = 4; // set member variables in instance a
b.x = 5; b.y = 12; // set member variables in instance b
std::cout << a.DistanceFromOrigin() << std::endl;
std::cout << b.DistanceFromOrigin() << std::endl;
```

The -> operator is a shorthand used to access member variables and functions from pointers to an instance of a class:

```cpp
Point2D *pta = &a;
pta->x = 3; pta->y = 4; // could use (*pta).x and (*pta).y
std::cout << pta->DistanceFromOrigin() << std::endl;
```
Member functions

class Point2D
{
public:
    double x, y;
    double DistanceFromOrigin() // function defined inside class
    {
        return std::sqrt(x*x + y*y);
    }
};

class Point2D
{
public:
    double x, y;
    double DistanceFromOrigin(); // prototype inside class
};

double Point2D::DistanceFromOrigin() // body outside class
{
    return std::sqrt(x*x + y*y);
}
Using classes

Open the example code ‘Classes’.

Two Point2D objects a and b are created, their member variables x, y are set, and the DistanceFromOrigin() member function is called on each.

1. Change a.x and a.y to represent the point (1, 1) and verify that the output from a.DistanceFromOrigin() returns $1.414 \ldots \approx \sqrt{2}$.

2. Create a new class instance c, with c.x=7 and c.y=24. What does c.DistanceFromOrigin() return?

3. What does the statement a=b; do to the member variables of a?
Access specifiers

In the previous example, all member functions and variables were defined under the **public** access specifier:

```cpp
public:
    double x, y;
    double DistanceFromOrigin();
```

There are three types of access-specifiers:

- **public** member variables/functions can be accessed from any code (provided there is an instance of the class, here `a`):

```cpp
int main()
{
    Point2D a; // members declared as public
    a.x = 3; a.y = 4; // access member variables in main,
    // i.e. 'outside' the class
}
```
Access specifiers

In the previous example, all member functions and variables were defined under the **public** access specifier:

```cpp
public:
    double x, y;
    double DistanceFromOrigin();
```

There are three types of access-specifiers:

- **private** member variables/functions can only be accessed only within member functions of this class:

```cpp
int main()
{
    Point2D a; // members declared as private
    a.x = 3; a.y = 4; // forbidden!
}
```
Access specifiers

In the previous example, all member functions and variables were defined under the `public` access specifier:

```cpp
public:
    double x, y;
    double DistanceFromOrigin();
```

There are three types of access-specifiers:

- **protected** members can only be accessed within member functions of this class *and any classes derived from it*.

```cpp
int main()
{
    Point2D a; // members declared as protected
    a.x = 3; a.y = 4; // forbidden!
}
```
class Point2D
{
    public: // members below can be accessed outside the class
        void SetXY(double x_, double y_)
        {
            x = x_; y = y_; // set class members to parameter values
        }
    
    // Public functions to return values of private variables
    double GetX() {return x;}
    double GetY() {return y;}

    private: // these members only accessible by
        double x, y; // members of this class
    
};

Point2D a;
a.SetXY(3, 4); // ok, Point2D::SetXY is public
std::cout << a.x; // ERROR! Point2D::x is private
std::cout << a.GetX(); // ok, Point2D::GetX() is public
Open the example code ‘Public and private’.

Two Point2D class is defined with private member variables x and y, as in the previous slide.

1. Uncomment the line `std::cout << a.x;` in `main` and try to compile the program. What compiler error do you get and why?

2. Now move the member variables x and y into the `public:` section of the class. Check that this allows the program to compile and run. What is the disadvantage of making these member variables public?
Constructors

A constructor is a special member function that is called when a class instance is created. A constructor has:

- the same name as the class
- no return value

```cpp
class Point2D
{
public:
    Point2D(double x_, double y_) // Constructor, with 2 args
    {
        x = x_; y = y_; // set class members to parameter values
    }
private:
    double x, y;
};
```

Point2D a(3, 4); // Call our constructor above
Point2D b; // Error! Does not match any constructor
Constructors

• Constructors are usually **public**

• In a class without any constructors, a *default constructor* (that takes no arguments and does nothing) is defined implicitly.

• To allow the definition `Point2D b;`, we need to add a constructor with no arguments to the class definition above:

```java
Point2D()
{
    x = y = 0.0; // set member variables to zero
}
```

• You may see code that instead initialises (rather than assigns) member variables using a constructor *initialiser-list*:

```java
Point2D() : x(0.0), y(0.0) {}  
```

Quicker for member variables that are themselves classes.
Open the example code ‘Constructors’.

Two `Point2D` class is defined with a constructor that takes two parameters, as in the previous slide.

1. Verify that `Point2D a(3,4);` compiles and runs, but that `Point2D b;` gives an error.

2. Now remove the constructor from the `Point2D` class. Verify that `Point2D a(3,4);` no longer compiles. Does the command `Point2D b;` now work? Why?
Classes – what is wrong with the following?

class Point2D
{
    public
        Point2D(double x, double y) : data(2, 0.0)
        {
        }
    private
        std::vector<double> data;
        double X() {return data[0];}
        double Y() {return data[1];}
}

Point2D a(3, 4), b;
std::cout << a.X << " , " << a.Y << std::endl;
std::cout << b.X << " , " << b.Y << std::endl;
Classes – what is wrong with the following?

class Point2D
{
public:  // missing colon
    Point2D(double x, double y) : data(2)
    {
        // elements of vector are at [0] and [1]
        data[0] = x; data[1] = y;
    }

    double X() {return data[0];}  // X and Y should be public
    double Y() {return data[1];}

private:  // missing colon
    std::vector<double> data;
};  // missing semicolon

Point2D a(3, 4), b(0, 0);  // b needs constructor args
// X() and Y() need function call brackets()
std::cout << a.X() << ", " << a.Y() << std::endl;
std::cout << b.X() << ", " << b.Y() << std::endl;
Summary

• Classes
  • Used to collect together variables and functions
  • Allow definition of new types
  • Allow polymorphism

• Access specifiers
  • public, private, protected

• Constructors