Introduction to Scientific Programming with C++

Session 3: Object oriented programming

Martin Uhrin

UCL

February 11-13th 2013

Table of Contents

Classes

Hold on to your hats

Definition

class a user defined compound data type that can hold both data and functions.

Classes are a powerful way of grouping data and functions to create custom data types that have certain *responsibilities*.

Classes

Hold on to your hats

Definition

class a user defined compound data type that can hold both data and functions.

Classes are a powerful way of grouping data and functions to create custom data types that have certain *responsibilities*.
Let's start with an example:

```
class Vector2
{
public:
  double x, y;
};
```

This defines a class called `Vector2` that has two member variables, `x` and `y`, of type `double` which are publicly accessible.

Classes

Let's play with `Vector2`:

```
int main()
{
  Vector2 r;
  r.x = 3.0;
  r.y = 10.0;
  std::cout << "x: " << r.x << " y: " << r.y;
}
```

Here we have created an *object* of type `Vector2` and used the dot operator (`.`) to access members of our vector, `r`.

Classes

Let's play with `Vector2`:

```
int main()
{
  Vector2 r;
  r.x = 3.0;
  r.y = 10.0;
  std::cout << "x: " << r.x << " y: " << r.y;
}
```

Here we have created an *object* of type `Vector2` and used the dot operator (`.`) to access members of our vector, `r`.

Definitions

object	an instance of a class. A class is to an object as blueprints are to a house.
instantiation	creating an instance of a class.
member variable	a variable that is contained within a class.

Class methods

What's all the fuss about?

As well as member variables, classes can have member functions.

Definition

member function (or *method*) a function that is contained within a class.

Class methods

What's all the fuss about?

As well as member variables, classes can have member functions.

Definition

member function (or *method*) a function that is contained within a class.

For example:

```
class Vector2
{
public:
  double length()
  { return sqrt(x * x + y * y); }

  double x, y;
};
int main()
{
  Vector2 r;
  r.x = 3;  r.y = 4;
  std::cout << "|r| = " << r.length() << "\n";
}
```

Output: `|r| = 5`

Classes

Fuller example

```
class Vector2
{
public:
  double length()
  { return std::sqrt(x * x + y * y); }

  void sub(const Vector2 toSub)
  { x -= toSub.x; y -= toSub.y; }

  void mul(const double k)
  { x *= k; y *= k; }

  double dot(const Vector2 b)
  { return x * b.x + y * b.y; }

  void printCoords()
  { std::cout<< x << " "<< y << "\n";}

  double x, y;
};
```

Classes

Fuller example

```
class Vector2
{
public:
  double length()
  { return std::sqrt(x * x + y * y); }

  void sub(const Vector2 toSub)
  { x -= toSub.x; y -= toSub.y; }

  void mul(const double k)
  { x *= k; y *= k; }

  double dot(const Vector2 b)
  { return x * b.x + y * b.y; }

  void printCoords()
  { std::cout<< x << " "<< y << "\n";}

  double x, y;
};
```

```
int main()
{
  Vector2 r1, r2;
  r1.x = 2; r1.y = 1;
  r2.x = 4; r2.y = 10;

  std::cout << "r1: ";
  r1.printCoords();
  std::cout << "r2: ";
  r2.printCoords();
  std::cout << "Performing
       r2.sub(r1)\n";
  r2.sub(r1);
  std::cout << "r2: ";
  r2.printCoords();

  return 0;
}
```

../code/3_oop/lectures/simple_vector2.cpp

Output:
```
r1:  2 1
r2:  4 10
Performing r2.sub(r1)
r2:  2 9
```

Accessibility

So what's this `public:` statement all about? `public` is known as an access specifier.

Definition

access specifier tells the compiler who (i.e. which parts of the code) have access to the members below it in the class.

The three access specifiers have the following meanings:

[1] We'll see what these are later.

Accessibility

So what's this `public:` statement all about? `public` is known as an access specifier.

Definition

access specifier tells the compiler who (i.e. which parts of the code) have access to the members below it in the class.

The three access specifiers have the following meanings:

`public` Any part of the code where the object is visible can access these.

[1] We'll see what these are later.

Accessibility

So what's this `public:` statement all about? `public` is known as an access specifier.

Definition

access specifier tells the compiler who (i.e. which parts of the code) have access to the members below it in the class.

The three access specifiers have the following meanings:

`public` Any part of the code where the object is visible can access these.

`private` Only members of this class or members of friends[1] can access these.

[1] We'll see what these are later.

Accessibility

So what's this `public:` statement all about? `public` is known as an access specifier.

Definition

access specifier tells the compiler who (i.e. which parts of the code) have access to the members below it in the class.

The three access specifiers have the following meanings:

`public` Any part of the code where the object is visible can access these.

`private` Only members of this class or members of friends[1] can access these.

`protected` Same as private but also allows members of classes derived from this one [1] to access these.

[1] We'll see what these are later.

Accessibility

So what's this `public:` statement all about? `public` is known as an access specifier.

Definition

access specifier tells the compiler who (i.e. which parts of the code) have access to the members below it in the class.

The three access specifiers have the following meanings:

`public` Any part of the code where the object is visible can access these.

`private` Only members of this class or members of friends[1] can access these.

`protected` Same as private but also allows members of classes derived from this one [1] to access these.

By default classes should have `private` access to all members.

[1] We'll see what these are later.

Encapsulation

The aim of a good object oriented design is to break the problem up into chunks that have well defined responsibilities which can be implemented as independently. This way details, like data members, can be hidden in each class and we interact with classes via their member functions. This is called *encapsulation*.

Encapsulation

The aim of a good object oriented design is to break the problem up into chunks that have well defined responsibilities which can be implemented as independently. This way details, like data members, can be hidden in each class and we interact with classes via their member functions. This is called *encapsulation*.

An example:

```
class Vector2
{
public:
  double getX()
  { return x; }
  double getY()
  { return y; }
private:
  double x, y;
};
```

Encapsulation

The aim of a good object oriented design is to break the problem up into chunks that have well defined responsibilities which can be implemented as independently. This way details, like data members, can be hidden in each class and we interact with classes via their member functions. This is called *encapsulation*.

An example:

```
class Vector2
{
public:
  double getX()
  { return x; }
  double getY()
  { return y; }
private:
  double x, y;
};
```

```
class Vector2
{
public:
  double getX()
  { return coords[0]; }
  double getY()
  { return coords[1]; }
private:
  double coords[2];
};
```

Same interface (public members), different implementation details.

Encapsulation

Encapsulation separates implementation details from the way the class is used, this has several advantages:

Encapsulation

Encapsulation separates implementation details from the way the class is used, this has several advantages:

- Gives you more flexibility for making improvements in the future as other parts of the code can remain unchanged.

Encapsulation

Encapsulation separates implementation details from the way the class is used, this has several advantages:

- Gives you more flexibility for making improvements in the future as other parts of the code can remain unchanged.
- Makes code more maintainable: a bug will likely be localised to one class.

Encapsulation

Encapsulation separates implementation details from the way the class is used, this has several advantages:

- Gives you more flexibility for making improvements in the future as other parts of the code can remain unchanged.
- Makes code more maintainable: a bug will likely be localised to one class.
- Developing complex code can be easier as a common set of interfaces can be defined and then implemented separately in due time. So long as each class fulfills its responsibilities the whole will work.

Encapsulation

Encapsulation separates implementation details from the way the class is used, this has several advantages:

- Gives you more flexibility for making improvements in the future as other parts of the code can remain unchanged.
- Makes code more maintainable: a bug will likely be localised to one class.
- Developing complex code can be easier as a common set of interfaces can be defined and then implemented separately in due time. So long as each class fulfills its responsibilities the whole will work.

Do

Make all member variables `private`. Provide a clear set of methods to interact with the class such that its responsibilities can be fulfilled.

Accessor methods

Definitions

getter a method that gets the value of an internal variable.

setter a method that sets the value of an internal variable.

Accessor methods

Definitions

getter a method that gets the value of an internal variable.

setter a method that sets the value of an internal variable.

You've seen getters, we can now add setters:

```
void setX(const double newX)
{ x = newX; }
void setY(const double newY)
{ y = newY; }
```

Accessor methods

Definitions

getter a method that gets the value of an internal variable.

setter a method that sets the value of an internal variable.

You've seen getters, we can now add setters:

```
void setX(const double newX)
{ x = newX; }
void setY(const double newY)
{ y = newY; }
```

But wait, doesn't this undermine encapsulation? Yes! So use it only for small classes with simple members variables.

Constructors

A class may want to initialise variables or dynamic memory when it is instantiated, this is done in the constructor.

Definition

constructor A special member function that has the same name as the class and no return type.

Constructors

A class may want to initialise variables or dynamic memory when it is instantiated, this is done in the constructor.

Definition

constructor A special member function that has the same name as the class and no return type.

For example:

```
class Vector2
{
public:
  Vector2(const double x0,
    const double y0)
  {
    x = x0; y = y0;
  }
  /*..and the rest..*/
private:
  double x, y;
};
```

```
int main()
{
  Vector2 r1(3., 10.4);
  r1.printCoords();
  return 0;
}
```

Output: `3 10.4`

Default constructors

Definition

default constructor — A zero argument constructor provided automatically by the compiler if, and only if, no custom constructors are supplied.

As soon as you write one constructor the default is no longer provided!

Default constructors

Definition

default constructor A zero argument constructor provided automatically by the compiler if, and only if, no custom constructors are supplied.

As soon as you write one constructor the default is no longer provided! Let's try using our last version of `Vector2`:

```
int main()
{
  Vector2 r1; // error 'Vector2': no appropriate
  return 0;         // default constructor available
}
```

Constructor overloading

You can have more than one constructor, this is called *constructor overloading*. For example:

```
class Vector2
{
public:
  Vector2()
  {
    x = 0.0; y = 0.0;
  }
  Vector2(const double x0,
    const double y0)
  {
    x = x0;  y = y0;
  }
private:
  double x, y;
};
```

```
int main()
{
  Vector2 r1(3., 10.4);
  Vector2 r2; // No brackets!
  std::cout << "r1: "
  r1.printCoords();
  std::cout << "r2: "
  r2.printCoords();
  return 0;
}
```

Output:
```
r1:  3 10.4
r2:  0 0
```

Destructors

A class may want to release resources like dynamic memory when it is destructed, this is done in the destructor.

Definition

destructor A special member function that has the same name as the class but prefixed with a ~(tilde), takes no parameters and has no return type.

Destructors

A class may want to release resources like dynamic memory when it is destructed, this is done in the destructor.

Definition

destructor A special member function that has the same name as the class but prefixed with a ~(tilde), takes no parameters and has no return type.

An example:

```
class ClassicalSpinString {
public:
  ClassicalSpinString(const int length)
  {
    vectors = new Vector2[length];
  }
  ~ClassicalSpinString()
  {
    delete[] vectors; // Have to delete to avoid leak!
  }
private:
  Vector2 * vectors;
};
```

Destructors

We've constructed ourselves, why not destruct ourselves?

When does a destructor get called? As soon as the scope that the object was defined in comes to an end.

Destructors

We've constructed ourselves, why not destruct ourselves?

When does a destructor get called? As soon as the scope that the object was defined in comes to an end.

For example:

```
int main()
{
  bool calcSpinStringProperties = true;
  // do stuff...
  if(calcSpinStringProperties)
  {
    SpinString spins;
    // do stuff with spins...
    // ...report results.
  } // <- Here spins is destructed
  return 0;
}
```

Destructors

We've constructed ourselves, why not destruct ourselves?

When does a destructor get called? As soon as the scope that the object was defined in comes to an end.

For example:

```
int main()
{
  bool calcSpinStringProperties = true;
  // do stuff...
  if(calcSpinStringProperties)
  {
    SpinString spins;
    // do stuff with spins...
    // ...report results.
  } // <- Here spins is destructed
  return 0;
}
```

Do

Practice safe programming: use destructors to clean up after yourself.

Pointers to classes

Pointers to classes work as expected:

```
int main()
{
  Vector2 * r1 = new Vector2;
  (*r1).printCoords();

  delete r1; // Clean up
}
```

Pointers to classes

Pointers to classes work as expected:

```
int main()
{
  Vector2 * r1 = new Vector2;
  (*r1).printCoords();

  delete r1; // Clean up
}
```

But using `(*name).member` can get a bit annoying. Kindly, C++ provides us with an alternative: the indirection operator (->).

Pointers to classes

Pointers to classes work as expected:

```
int main()
{
  Vector2 * r1 = new Vector2;
  (*r1).printCoords();

  delete r1; // Clean up
}
```

But using `(*name).member` can get a bit annoying. Kindly, C++ provides us with an alternative: the indirection operator (->). Once more:

```
int main()
{
  Vector2 * r1 = new Vector2;
  r1->printCoords();

  delete r1; // Clean up
}
```

Operator overloading

We've overloaded functions, why stop there...

C++ provides certain operators for built in types e.g.:

```
int a = 5, b = 10;
int c = a + b;

bool haveMyCake = true, eatIt = true;
bool haveMyCakeAndEatIt = haveMyCake && eatIt;
```

But wouldn't it be cool to be able to do:

```
Vector2 r1(3, 14), r2(23, 1);
Vector2 r12 = r2 - r1;
```

With operator overloading we can!

Operator overloading

Also a medical condition suffered by overworked elevator operators

What should a `Vector2` minus method look like? Probably:

- Take a `Vector2` as a parameter to subtract.

Operator overloading

Also a medical condition suffered by overworked elevator operators

What should a `Vector2` minus method look like? Probably:

- Take a `Vector2` as a parameter to subtract.
- Return a `Vector2` that is the result.

Operator overloading

Also a medical condition suffered by overworked elevator operators

What should a `Vector2` minus method look like? Probably:

- Take a `Vector2` as a parameter to subtract.
- Return a `Vector2` that is the result.

Let's try:

```
class Vector2
{
public:
  Vector2 operator -(Vector2 toSub)
  {
    return Vector2(x - toSub.getX(), y - toSub.getY());
  }
/* ..other stuff */
};
```

Hey, that's not bad!

Operator overloading

So how does this work? We haven't called a method! Let's breakdown the statement: `Vector r12 = r2 - r1;`

Operator overloading

So how does this work? We haven't called a method! Let's breakdown the statement: `Vector r12 = r2 - r1;`

1. The compiler sees you've used the - operator on an object that has a `operator -` method and interprets this as if you had written the, also valid, statement:

```
r2.operator -(r1);
```

Operator overloading

So how does this work? We haven't called a method! Let's breakdown the statement: `Vector r12 = r2 - r1;`

1. The compiler sees you've used the - operator on an object that has a `operator -` method and interprets this as if you had written the, also valid, statement:

```
r2.operator -(r1);
```

2. Now it's more obvious: `r2`'s `operator -` method is invoked with `r1` as a parameter.

Operator overloading

So how does this work? We haven't called a method! Let's breakdown the statement: `Vector r12 = r2 - r1;`

1. The compiler sees you've used the - operator on an object that has a `operator -` method and interprets this as if you had written the, also valid, statement:

```
r2.operator -(r1);
```

2. Now it's more obvious: `r2`'s `operator -` method is invoked with `r1` as a parameter.
3. Finally the returned `Vector2` is copied into our `r12`.

Warning!

This may seem *cool* but it should be used sparingly. Use only when the effect of the operator is very clear, otherwise using seemly normal operators will lead to unexpected results.