Classes

Creating a class creates a new data type. Provide a mechanism to group data and functions.

Definition

Classes are defined using the `class` keyword

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

Data are referred to as "members", i.e. `x, y` are members of the `Point` class. Members are accessed with the `.`, or `->` if using a pointer to the object.

Member access

- `public`, members are accessible by users
- `private`, default when no keyword is used, members not accessible.
- `proctected`, members accessible in subclasses.

Each object have their own copy of the data members (except for static members). All functions are shared.

Functions

Can be defined outside the class using

```
void Point::set_x(double nx( {
    ...
}
```

Static Members

Data member shared by all the objects of the same class. Define once and only once outside the class.

Constructors

Initialisation function called after an object is allocated in memory. Constructors have the same name as the class itself and do not have a return type.

`class_name(args)`

The default constructor is a constructor defined without any arguments. If a constructor is not supplied, then the complier supplies a constructor that does nothing. If a constructor is provided, default or not, the compiler will not supply a constructor.

Conversion functions are implicitly defined when a object can be created implicitly. The `explict` keyword can be used to prevent the constructor from being used as a conversion function.

Another constructor is the copy constructor, which takes only an object of the same class as an argument

```
Point(const Point &p)
```

Notice the `&`, i.e. a reference is passed

Destructor

Releases resources upon object termination.

```
~Point() {
    npoints--;
    cout << "Object deleted << endl;
}
```

`this` pointer

Gives access to the members of an object

```
this->x // gives access to the x member
*this // returns object
```

Operator Overloading

Operations can be defined such as addition, subtraction, multiplication etc. on the object.

```
return_type operator op() // unary
return_type operator op(right_arg) // binary
```

Commutativity has to be implemented separately, i.e. `int + Point != Point + int`.

`friend` keyword

Operators can also be implemented outside the class i.e. as a normal function. These cases apply when mixing types i.e. defining the `*` operator for `Point` and `int`.

Global functions defined on their own do not have access to private members of the class. The `friend` operator can be used to get around this.

`=` operator

If not supplied, the complier will supply its own version. The `copy` constructor is invoked by default.

`()` operator

Used to define a function. Can be specified to define a function object. The benefit over a function is that function object have an internal state; allowing for customisation of the function.

`[]` operator

Subscript operator used to access element in an array.

```
type& operator [] (const int index)
```

A reference is returned. A `const` reference can also be returned, but this has to be defined separately. It is good practise to define both types of references, `const` and non-`const`.

`++` / `--` operators

```
class_name& operator ++ ()  // ++cls
class_name operator ++ (int)    // cls++
```

`int` is a dummy variable to distinguish the prefix and postfix versions.

Conversion functions

Data conversion between instances of the class and another type.

```
operator type ()
```

Deriving Classes

Class inheritance allows for the creation of class hierarchies.

```
class class_name : public base_class {
    ...
};
```

Inherits all members except for constructors. Multiple inheritance is also supported.

Inheritance keyword

- `public`: all members are inherited (exc. constructors). Member access is preserved
- `private`: members become private in the subclass regardless of access level.
- `protected`: private and protected members inherited as is, public become protected.

Derived classes can inherit all the constructor via

```
using base_class::base_class
```

Note (C++11)

Up-casting

A pointer of type parent can be associated with an address of the child type. The pointer can perform any functionality of the parent class

```
class Animal { ... };
class Rabbit : public Animal { ... };

Animal *ptrAnimal;
ptrAnimal = &objRabbit;
```

Virtual Functions and Overriding

Declarations may declare members that are already members of the base class. Such members are said to override the old declarations. `virtual` keyword used to ensure declaration for class is called at runtime.

- Any function that might be overridden should be declared virtual
- Types must match
- `virtual` keyword only need once (for base class)

The `override` keyword can be used to self-document when a function is overridden. (C++11)

```
funciton_declaration override;
```

Bit Fields

Bit fields are class member. They are useful when space is at a premium / for manipulating individual bits.

```
unsigned field_name : integer;
```

Unions

A data type that in which the members share the same address in memory.