

CS2510

MODERN PROGRAMMING LANGUAGES

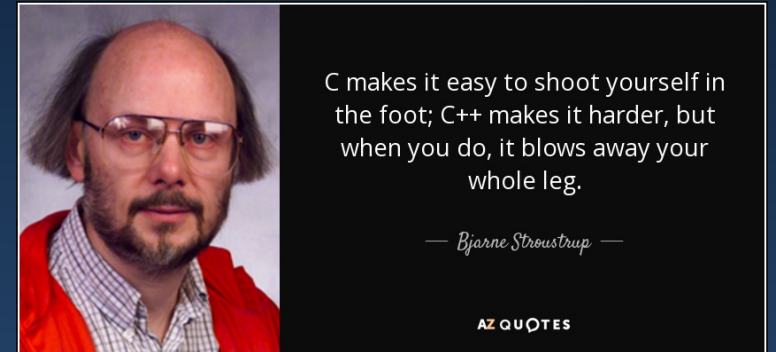
Object-Oriented Programming 3

Prof. Peter Edwards
p.edwards@abdn.ac.uk

C++

Introduction to C++

- (Mostly) an extension of C to include:
 - Classes
 - Inheritance and Multiple Inheritance
 - Method and Operator Overloading
 - Templates
 - Default Parameters
 - New (and better) Standard Library
 - etc.
- A few small syntactic differences from C.



Bjarne Stroustrup, *The C++ Programming Language: Special Edition*

Classes in C++

- First, let's look at support in C for complex types:
 - struct is a means used to represent a heterogeneous collection of data;
 - typedef creates an alias to refer to the struct
- C++ introduced `class`:

```
typedef struct {  
    int layers;  
    float price;  
    char *baker_name;  
} Cake;
```

```
Cake cake1 = {2, 14.50, "John"};  
printf("Layers %d", cake1.layers);
```

```
class classname {  
  
    // instance variables and methods  
    // members private by default  
  
};
```

public: modifier
forces contents to be
visible – not a good
idea!

```
class Cake {  
  
    public:  
  
        int layers;  
        float price;  
        string baker_name;  
  
};
```

```
Cake cake1;  
cake1.baker_name = "Owen";  
cake1.layers = 4;  
cake1.price = 35.50;  
cout << cake1.baker_name << endl;
```

Methods (Functions) in C++

- Method syntax:

```
return_type [class_name::]method_name(parameters ...) {  
    // body  
}
```

- Methods can be defined as part of the class or outside, using the scope operator ::

OUTPUT:

Owen is baking the cake at 200C.

```
class Cake {  
  
public:  
    int layers;  
    float price;  
    string baker_name;  
  
    void icing();    // prototype (declaration)  
                    // definition is elsewhere  
    void bake(int temp) {  
        cout << baker_name << " is baking the cake at "  
            << temp << "C." << endl;  
    }  
};
```

```
void Cake::icing() {  
    cout << "The cake now has icing" << endl;  
}
```

```
Cake cake1;  
cake1.baker_name = "Owen";  
cake1.layers = 4;  
cake1.price = 35.50;  
cake1.bake(200);
```

Methods (Functions) in C++


- **Constructors & *Destructors***

- Special purpose methods used to initialise or destroy an object.

- **Constructor**

- Called when an object is created.
 - Method with same name as the class, no return type.

Method
overloading
here!




```
class Cake {  
  
    int layers;  
    float price;  
    string baker_name;  
  
public:  
    Cake() {layers = 1; price = 10.0; baker_name = "Unknown";}   
    Cake(int p) {price = p; ...etc }  
  
    // other methods  
};
```

Methods (Functions) in C++

– Destructor

- Called automatically whenever an object is destroyed.
 - for local variables when they go out of scope;
 - for global variables when the program ends.
- Only needed in classes that have a pointer to dynamically allocated memory.
- Method with name of the class preceded by ~
 - no return type or parameters.

```
class array {  
  
    double *a;  
    int len;  
  
public:  
    array(int l = 0) { len = l; a = new double[len]; }  
    ~array() { delete[] a; }  
};
```



Note use of delete
here!

Access Controls (Visibility)

- Keywords specify the access level of the members of the class.
 - `private`:
 - `public`:
 - `protected`:
- Usually, instance variables of a class are declared in the `private` section of the class and the methods are in the `public` section.

private	Class members declared as <code>private</code> can be used only by member methods and friends (classes or methods) of the class.
protected	Class members declared as <code>protected</code> can be used by member methods and friends (classes or methods) of the class. Additionally, they can be used by classes derived from the class.
public	Class members declared as <code>public</code> can be used by any method.

friend classes/methods - declared with `friend` keyword;
friend methods or methods from a class declared as `friend`
are granted access to private elements of a class.

Inheritance

- A class need not be the subclass of any class.
- Subclassing process can be declared with access controls (private, public, protected), which define potential changes in access by subclasses.
- Method overriding is permitted.
 - Method on derived class must have same signature and visibility as version in base class.

```
class derived_class: access_modifier base_class {  
    // class body  
};
```

private derivation	Inherited public and protected members are private in the subclasses.
protected derivation	Inherited public and protected members are protected in the subclasses.
public derivation	Inherited public and protected members are also public and protected in subclasses.

C++ Inheritance - Example

```
class Cake {  
  
    int layers;  
    float price;  
    string baker_name;  
  
protected:  
    float profit;  
  
public:  
    int calories;  
    // constructors & other methods not shown here  
};
```

```
class BirthdayCake : public Cake {  
  
    // In this case, profit is protected  
    // and calories is public  
  
};
```

```
class WeddingCake : private Cake {  
  
    // In this case, profit and calories are  
    // private, and no derived class  
    // has access to any member of cake class  
  
};
```

Virtual Methods

- Object of a derived class can be referred to using a pointer or reference to base class type.
- If base class methods are overridden by the derived class - which version is called?
 - Early (compiler): according to declared type of pointer/reference.
 - Late (run-time): according to actual type referred to.
- A C++ method can be defined to be *virtual*, meaning dynamically bound (run-time).

Inheritance & Abstract Classes

- ***Abstract Class:***

- Class that is too general to create real objects.
- Used to provide an implementation guide for other classes.
- Many inheritance hierarchies have abstract superclasses occupying the top few levels.

- C++ realises abstract classes through ***pure virtual functions***.

```
class Food {  
  
public:  
    virtual void eat() = 0;  
    // pure virtual function indicated by "=0"  
};
```



An abstract class.

- Pure virtual function must be overridden by derived class; if not - derived class is also abstract...

Multiple Inheritance

- Multiple inheritance is supported

```
class BirthdayCake : public Cake, public LuxuryItem {  
  
    // class definition as usual  
    // data members, methods  
  
};
```

- If there are two inherited members with the same name, they can both be referenced using the scope operator (::)

```
class Cake {  
  
    int vat() {  
        // Cake class vat method behaviour  
    }  
};  
  
class LuxuryItem {  
  
    int vat() {  
        // LuxuryItem class vat method behaviour  
    }  
};
```

```
BirthdayCake bcake1;  
bcake1.vat();           // error - ambiguous!
```

```
class BirthdayCake : public Cake, public LuxuryItem {  
  
    int vat() { LuxuryItem::vat(); }  
  
}
```

```
BirthdayCake bcake1;  
bcake1.vat();           // OK
```

Templates

- Templates introduce *generic programming* into C++.
- Act as a blueprint for creating a class, method independent of any particular type.
- Three kinds of C++ templates:
 - Method templates
 - Class templates
 - Variable templates (*since 2014*)
- *Parametric polymorphism.*

```
template <class type_parameter> class class_name {  
    // template body  
};
```


Template - Example

```
template <class T> class Stack {  
  
private:  
    T* stackPtr ;  
    int size ; // number of items on Stack  
  
public:  
    Stack(int = 10);  
    ~Stack() { delete [] stackPtr ; }  
    int push(const T&);  
    int pop(T&) ;  
};  
  
template <class T>  
int Stack<T>::push(const T& item)  
{  
    if (!isFull())  
    {  
        stackPtr[++top] = item ;  
        return 1 ; // push successful  
    }  
    return 0 ; // push unsuccessful  
}
```



Template class
definition

Using template to create a
Stack of floats.



```
typedef Stack<float> FloatStack;  
FloatStack fs(5);  
fs.push(1.1);
```

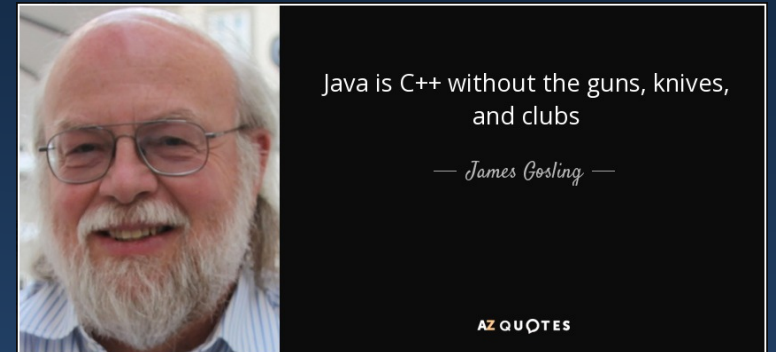
C++ Summary

- C++ provides extensive access controls (unlike Smalltalk).
 - nested classes permitted
- C++ supports single and multiple inheritance.
 - overriding permitted
 - abstract classes
- C++ supports **ad hoc**, **subtype** and **parametric polymorphism**.
- In C++, the programmer must decide at design time which methods will be statically bound and which must be dynamically bound.
 - Static binding is faster!

Java

Introduction to Java

- Originally conceived as a language for intelligent consumer electronics - 1991.
- Sun Microsystems project “Green” developed language Oak -> Java!
 - *Write Once – Run Anywhere*
- Java and C++ syntax very close.



James Gosling, Bill Joy, Guy L Steele Jr, Gilad Bracha, *The Java Language Specification*

