Introduction to C++

2501ICT/7421ICT Nathan

René Hexel and Joel Fenwick

School of Information and Communication Technology
Griffith University

Semester 1, 2012
Outline

1. C++ Objects and Classes
2. Compiling C++ Code
Objective-C vs C++

- **Size of language**
  - **ObjC** C with minimal additions to support OO. Most features provided via methods in base class.
  - **C++** More language features. C approaches still work but C++ often provides a better alternative to plain C. Many language features can be redefined but you do not need to know all the rules to write useful programs.

- **Division of labour**
  - **ObjC** pushes resolving calls to the runtime (eg: missing methods are warnings not errors)
  - **C++** requires everything to make sense at compile time.

- **Reflection**
  - **ObjC** Methods and protocols can be tested at runtime.
  - **C++** No reflection capabilities.
A Point Class Interface

Objective-C: `Point.h`

```c
#import <Foundation/Foundation.h>

@interface Point : NSObject
{
    int x; // member variables
    int y; // protected by default
}
- init; // constructor

- (int) x; // access methods
- (void) setX: (int) newx;
@end
```

C++: `Point.h`

```c
#ifndef POINT_H // include file protection
#define POINT_H

class Point
{
    int x; // member variables
    int y; // private by default

    public:
    Point(); // public methods
    int getX(); // access methods

    void setX(int newx);
}; // semicolon is important!
#endif // POINT_H
```

René Hexel and Joel Fenwick
Introduction to C++
A Point Class Implementation

**Objective-C: Point.m**

```objective-c
#import "Point.h"

@implementation Point

- init // initialiser
{
    x = 0; y = 0;
    return self;
}

- (int) x // get method
{
    return x;
}

- (void) setX: (int) newx
{
    x = newx;
}
@end
```

**C++: Point.cc**

```cpp
#include "Point.h" // no #import

Point::Point() // constructor
{
    x = 0; // does not
    y = 0; // return self
}

int Point::getX() // Point:: prefix
{
    return x;
}

void Point::setX(int newx)
{
    x = newx;
}
```

René Hexel and Joel Fenwick

Introduction to C++
### Objective-C: Main.m

```objective-c
#import "Point.h"

int main(int argc, char *argv[]) {
    Point *pt = [Point new];

    int x = [pt x]; // get x
    [pt setX: x + 5]; // set x

    return 0;
}
```

### C++: Main.cc

```cpp
#include "Point.h"

int main(int argc, char *argv[]) {
    Point *xy = new Point();

    int x = xy->getX();
    x = xy->setX(x + 5);

    return 0;
}
```
Summary (1)

- Classes are split into interface `file.h` and implementation `file.cc`
  - the name of the `file` should always be the class name
- Typed Object references are Pointers `*`
  - `Point *p` (like in Objective-C)
- No generic object of type `id`!
  - → methods are resolved at compile time
  - ⇒ casting needed!
- Method invocations use `->` instead of `[]` (or `. in Java`
  - `object->method();` instead of `[object method];` in Objective-C
Dedicated Constructor
- name of the class
  → does not need to return `self`
- `this` refers to the current object
  - like `self` in Objective-C
Compiling C++ Code
The Clang compiler frontend `clang++` knows C++
- `clang++ -c -Wall -o file.o file.cc`

Linking also works with `clang++`
- standard C++ runtime `libc++` is automatically linked

Different add-on API setups have different locations
- STL, boost, libqt, etc.
Example Makefile for C++

Example (C++ example Makefile)

# An example Makefile for C++
#
# -- this Makefile is for a project containing a CppMain with main() and
# -- a CppModule.cc and CppModule.h class header and implementation file
#
# CPLUS=g++

.SUFFIXES: .o .cc

.cc.o:
    $(CPLUS) -c -Wall -o $*.o $*.cc

Program: CppMain.o CppModule.o
    $(CPLUS) -o Program CppMain.o CppModule.o

CppModule.o: CppModule.cc CppModule.h