Moving from Java to C++

Nicolas Pope and Alan Hazelden

Overview

1. Compilers
2. Classes
3. Header files
4. Memory and Pointers
5. Inheritance, polymorphism
6. Namespaces
7. Templates

Where to begin...

Compilers: g++ (linux), mingw (Dev-C++) and Visual Studio C++ (Microsoft).

Simple “hello world” example:

```
#include <iostream>

int main(int argc, char *argv[]) {
    std::cout << “hello world” << std::endl;
    return 0;
}
```

```
g++ -o hello hello.cpp
```

Class Declarations

```
class Entity {
    public:
    Vector3d position();
    void position(Vector3D pos);

    int health();

    private:
    Vector3d m_position;
    int m_health;
}; // don't forget this semicolon
```

Different use of public and private.

No implementation of methods yet.

So where does this go...

File Organisation

- Not as strict as Java: classes don't need their own file
- Header files (.h): Contain declarations and very little implementation.
- CPP files (.cpp): Implementation goes here.
- Reason: Each CPP file must have access to all classes it needs for type checking. This is done by including header files.

Header Files

Every header file you create should have the following form where 'MYHEADER' is replaced with you filename:

```
#ifndef _MYHEADER_H_
#define _MYHEADER_H_

... declarations etc go here ...

#endif
```

Implementing Methods

```
#include “entity.h”

Vector3D Entity::position() {
    return m_position;
}

void Entity::position(Vector3D pos) {
    m_position = pos;
}
```

This is a CPP file.

But... these one line statements can be inlined in the header.

Memory and Pointers

```
Entity ship1 = Entity();

Entity ship2;

Entity ship3 = Entity(position);

Entity ship4(position);

// Entity ship5(); I'm pretty sure this is illegal



// Pointer to an Entity
Entity* ship6 = new Entity();

delete ship6; // reclaim memory

Entity* ship7 = &ship1;
```

Arrays

```
//Default constructor called (I think)
Entity *ships = new Entity[5];


// Array of pointers
Entity **ships = new Entity*[5];
for (int i=0; i<5; i++;) ships[i] = new Entity();




delete [] ships;

for (int i=0; i<5; i++) delete ships[i];
delete [] ships;
```

Memory and Pointers

Accessing member functions and variables for pointer and non-pointer objects:

```
ship.position();

ship->position();

ship[2].position();

ship[2]->position();

Ship::staticMethod();
```

Passing by Reference

- Does not copy data but make sure to use const to make read-only.

```
void foo(const Vector3D &bar) {
    std::cout << "Vector: " << bar.x << "," << bar.y << "\n";
}

void square(int &p) {
    p = p*p;
}
```

Const

```
const float gravity = 9.8f;


const Entity entity;

const Entity* entity;

const Entity& entity;


void Entity::someMethod ();

void Entity::someMethod () const;
```

Constructors and Destructors

- Constructors similar to Java
- Destructors called just before memory is deleted. Use to clean up memory you allocated inside the object.

```
Entity::Entity(const Vector3D &pos)
 : m_position(pos) {
    sprite = new Sprite(...);
}

Entity::~Entity() {
    delete sprite;
}
```

Simple Inheritance

```
class Ship : public Entity {
   public:
   Ship();
   ~Ship();
};



Entity *ship = new Ship();

...

delete ship;
```

Polymorphism

```
class Entity {
    public:
    Entity();
    virtual ~Entity();
    ...

    virtual void update();

    ...
};

class Ship : public Entity {
    public:
    Ship();
    ~Ship();

    void update();
};
```

Namespaces

- Group together related classes and methods.
- Reduces aliasing problems.

```
namespace MyGame {
    class Ship {
        ...
    };
};


    Entity *ship = new MyGame::Ship();

using namespace MyGame;
...
    Entity *ship = new Ship();
...
```

Using Templates

The Standard Template Library (STL) provides lists, vectors and other useful classes which rely upon templates.

```
#include <list>
...

int main() {
    std::list<Entity*> ships;
    ships.push_front(new Ship());
    ...
}
```

Misc Things

- Use static const not macros.
-

Advanced Topics

- Templates and meta-programming
- Multiple inheritance
- Function pointers
- Operator overloading
- Optimisations and inner workings
- Makefiles
- Design Patterns

Questions?

Wednesday Workshops (2pm in the Learning Grid)

Talk next week: Game programming in C++

Advanced topics?