EMBSYS 110, Spring 2017 Week 2

- Week 2
  - Object-oriented design.
  - Essential C++ techniques.
  - UML.

Copyright (C) 2017 Lawrence Lo

Object Oriented Design

- Encapsulation.
- Inheritance.
- Polymorphism.

Copyright (C) 2017 Lawrence Lo

Encapsulation

- Encapsulation
  - Before OOD, we use *structure* to group related data set together.
  - Functions that operate on a *structure* are separated from the structure.
  - An explicit pointer to the *structure* to be operated on is passed as the 1st arguments to the function.
  - C++ makes it easier with the *class* type. Data and functions operating on the data are grouped together in a *class*.

Copyright (C) 2017 Lawrence Lo

Encapsulation

- Member variables (attributes) and member functions (methods).
- Information hiding (access specifier).
- *Class* vs *object* (*instance*).
- Memory allocation – static vs dynamic.
- Static data members are shared by all objects.
- Implicit *this* pointer is provided to refer to the object to be operated on in a member function.
- Constructors and destructors.

Copyright (C) 2017 Lawrence Lo

Inheritance

- Inheritance
  - Base class and derived class.
  - “Is-a” relationship.
  - Promotes reuse. Derived class inherits members of base class (transitively).
  - Compare to aggregation which is a “has-a” relationship.

Copyright (C) 2017 Lawrence Lo

Polymorphism

- Polymorphism
  - *Upcasting* (to base class) vs *downcasting* (to derived class).
  - *Upcasting* is automatic.
    - Casting from a derived class object to a base class object.
    - Why does it work?
  - *Downcasting* requires explicit *static_cast*<> operator.
    - Programmers' responsibility to ensure correctness.
    - Otherwise, what would happen?

Copyright (C) 2017 Lawrence Lo

Polymorphism

- Through a pointer to a base class object, polymorphism invokes the “correct” version of a member function (when declared *virtual*) depending on the actual class of the object (assigned to the pointer).
- Also called dynamic binding.
- Useful for implementing interfaces.
- Important in framework design. Examples.

Copyright (C) 2017 Lawrence Lo

Essential C++

- Example 1

```
// Base class (qpcpp\include\qep.h)
class QEvt {
public:
    QSignal sig; //!< signal of the event instance
    // poolId_/refCtr_ intentionally uninitialized
    QEvt(QSignal const s) : sig(s) {}
    virtual ~QEvt() {}
private:
    uint8_t poolId_;          //!< pool ID (0 for static event)
    uint8_t volatile refCtr_; //!< reference counter
}
```

Copyright (C) 2017 Lawrence Lo

Essential C++

```
// Derived class (MyApp\Inc\fw_evt.h)
class Evt : public QP::QEvt {
public:
    static void *operator new(size_t s);
    static void operator delete(void *evt);
    Evt(QP::QSignal signal, uint16_t seq = 0) :
        QP::QEvt(signal), m_seq(seq) {}
    ~Evt() {}
    uint16_t GetSeq() const { return m_seq; }
protected:
    uint16_t m_seq;
};
```

Copyright (C) 2017 Lawrence Lo

Essential C++

```
// Another derived class (MyApp\Inc\event.h)
class UartActStartReq : public Evt {
public:
    enum {
        TIMEOUT_MS = 200
    };
    UartActStartReq(uint16_t seq, Fifo *outFifo, Fifo *inFifo) :
        Evt(UART_ACT_START_REQ, seq), m_outFifo(outFifo), m_inFifo(inFifo) {}
    Fifo *GetOutFifo() const { return m_outFifo; }
    Fifo *GetInFifo() const { return m_inFifo; }
private:
    Fifo *m_outFifo;
    Fifo *m_inFifo;
};
```

Copyright (C) 2017 Lawrence Lo

Essential C++

- Example 2 (MyApp\Src\System\Test.h)

```
class TestBase {
public:
  TestBase(uint32_t id = 0) : m_id(id) {}
  virtual void Print() {
    PRINT("TestBase m_id = %d\n\r", m_id);
  }
protected:
  uint32_t m_id;
};
```

Copyright (C) 2017 Lawrence Lo

Essential C++

```
class TestDerived1 : public TestBase {
public:
  TestDerived1(uint32_t id = 127) :
      TestBase(id) {}
  void Print() {
    PRINT("TestDerived1 m_id = 0x%x\n\r", m_id);
  }
};
```

Copyright (C) 2017 Lawrence Lo

Essential C++

- Test drive (MyApp\Src\System\System.cpp)

```
TestBase tb(100);
TestDerived1 td1;
TestBase *pb;
pb = &tb;
pb->Print();
pb = &td1;
pb->Print();
```

- Output:

```
TestBase m_id = 100
TestDerived1 m_id = 0x7f
```

Copyright (C) 2017 Lawrence Lo

Essential C++

- For further example, see MyApp\Src\fw_pipe.h
- At this point, we should know:
  - Encapsulation, inheritance and polymorphism.
  - Constructors, member initialization list and order of initialization.
  - The difference between function overriding and virtual function.
  - Access protection.
  - Template.

Copyright (C) 2017 Lawrence Lo

C++ Performance

- C++ features to avoid (Rule of thumb):
  - RTTI.
  - Exception.
  - Multiple inheritance.

Copyright (C) 2017 Lawrence Lo

UML

- Drawing tool (Visio, MagicDraw, UMLet, etc.)
- Class diagram.
  - Shows member variables and member functions (with their visibility).
  - Shows static relationship between classes, such as aggregation, composition and inheritance.
  - Does not show the internal of a class, such as what does a function do or how is a variable used.
  - http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html

Copyright (C) 2017 Lawrence Lo

UML

- Sequence diagram.
  - Shows the interaction between two of more objects.
  - Interaction can be synchronous (function calls) or asynchronous (events/messages).
  - Each sequence diagram only shows one particular scenario which can be a success case or a certain failure path.
  - Does not show what happens if the order of messages is different or a failure occurs at a different place.
  - A metaphor – it only shows a cross section of a 3D object.

Copyright (C) 2017 Lawrence Lo

UML

- You may need an ~infinite number of sequence diagrams to completely represent all possible scenarios.
- One may be tempted to start coding based on one sequence diagram, usually a success case and worry about failure cases later.
- Good for analysis, but hard to show complete design.
- https://www.ibm.com/developerworks/rational/library/3101.html

Copyright (C) 2017 Lawrence Lo

UML

- Statechart
  - Shows the complete behavior of an object.
  - Complete, precise and concise.
  - More on it next week.
  - Next lecture will be on Quantum Platform (a statechart framework) presented by guest instructor, Katie Elliott.

Copyright (C) 2017 Lawrence Lo