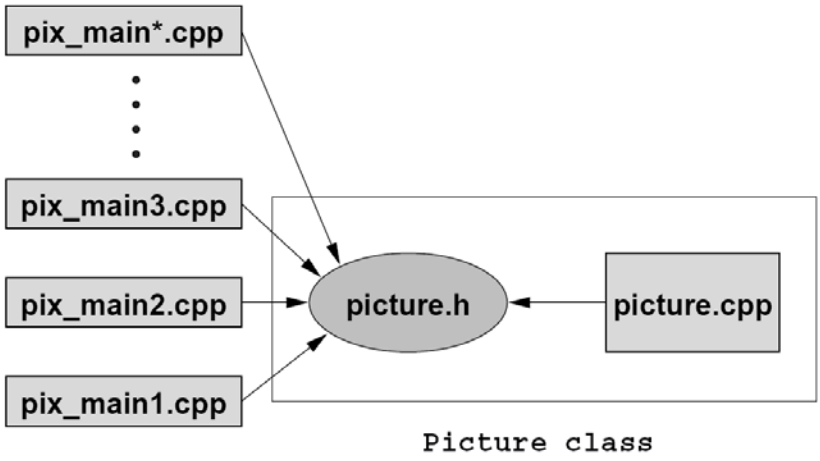
C++ Header Files



Build library of classes

- Separate from "using" programs
- Re-used by many different programs
- Just like predefined libraries

Header File Inclusion Rules

Basic rules of C++ header files:

- The header file inclusion mechanism should be tolerant to duplicate header file inclusions.
- A header file should be included only when a forward declaration would not do the job.
- The header file should be so designed that the order of header file inclusion is not important.

Examples:

- **Rule 1: The header file inclusion mechanism should be tolerant to duplicate header file inclusions.**

```
#ifndef ESTIMATE_FUNCTION_H
#define ESTIMATE_FUNCTION_H
#include <iostream>
using namespace std;

double estimateOfTotal(int minPeas, int maxPeas, int podCount);
//Returns an estimate of the total number of peas harvested.
//The formal parameter podCount is the number of pods.
//The formal parameters minPeas and maxPeas are the minimum
```

```
//and maximum number of peas in a pod.

#endif
```

- **C++ INCLUDE Rule : Use forward declaration when possible**

Suppose you want to define a new class B that uses objects of class A.

- B only uses references or pointers to A. Use forward declaration then : you don't need to include <A.h>. This will in turn speed a little bit the compilation.

```
class A ;

class B {
  private:
    A* fPtrA ;
  public:
    void mymethod(const& A) const ;
} ;
```

- B derives from A or B explicitely (or implicitely) uses objects of class A. You then need to include <A.h>

```
#include <A.h>

class B : public A {

} ;

class C {
  private:
    A fA ;
  public:
    void mymethod(A par) ;
}
```

- **The following sections will explain these rules with the help of an example.**

The following example illustrates different types of dependencies. Assume a class A with code stored in a.cpp and a.h.

a.h

```
#ifndef A_H
```

```
#define A_H
#include "abase.h"
#include "b.h"

// Forward Declarations/definition
class C;
class D;

class A : public ABase
{
 B m_b;
 C *m_c;
 D *m_d;

public:
 void SetC(C *c);
 C *GetC() const;

 void ModifyD(D *d);
};
#endif
```

a.cpp

```
#include "a.h"
#include "d.h"

void A::SetC(C* c)
{
 m_c = c;
}

C* A::GetC() const
{
 return m_c;
}

void A::ModifyD(D* d)
{
 d->SetX(0);
 d->SetY(0);
 m_d = d;
}
```

- **Why each class shall have it's own header and implementation file.**

Let’s introduce a term “translation unit”, where each unit has one class definition, and one .cpp file has a class implementation with a corresponding .h file of the same name.

It's efficient (from a compile/link) standpoint to organize your source code in translation units. This approach is very efficient if you're doing incremental link, because when one translation unit changes, you need to rebuild a lot of stuff.

Another benefit of translation units is to help with finding compilation/run-time errors, because many errors are reported via file name (e.g., "Error in Myclass.cpp, line 22"). It helps if there's a one-to-one correspondence between files and classes.

Reference:

1. C++ Header File Include Patterns
http://www.eventhelix.com/realtimemantra/headerfileincludepatterns.htm

2. Use forward declaration when possible
http://www-subatech.in2p3.fr/~photons/subatech/soft/carnac/CPP-INC-1.shtml

3. Multiple classes in a header file vs. a single header file per class
http://stackoverflow.com/questions/28160/multiple-classes-in-a-header-file-vs-a-single-header-file-per-class