Object – Oriented Programming
Week 2, Spring 2009

Weng Kai

http://fm.zju.edu.cn

Project

- Write a CLI program that reads scores and name of students, and prints out a summary sheet.
- The user can input as many students as possible. One students can have as many courses as possible.One course consists the name of the course and the marks the student got.

Point

```
typedef struct point {
    float x;
    float y;
} Point;


void print(const Point* p){
    printf(“%d %d\n”,p->x,p->y);
}
```

move (dx,dy)?

```
void move(Point* p,int dx, int dy)
{
        p->x += dx;
        p->y += dy;
}
```

Prototypes

```
typedef struct point {
    float x;
    float y;
 } Point;
void print(const Point* p);
void move(Point* p,int dx, int dy);
```

Usage

```
Point a;
Point b;
a.x = b.x = 1; a.y = b.y = 1;
move(&a,2,2);
print(&a);
print(&b);
```

C++ version

```
class Point {
public:
    void init(int x,int y);
    void move(int dx,int dy);
    void print() const;

private:
    float x;
    float y;
} ;
```

implementations

```
void Point::init(int ix, int iy) {
    x = ix; y = iy;
}
void Point::move(int dx,int dy) {
    x+= dx; y+= dy;
}
void Point::print() const {
    cout << x << ' ' << y << endl;
}
```

:: resolver

- <Class Name>::<function name>
- ::<function name>

```
void S::f() {
    ::f(); // Would be recursive otherwise!
    ::a++; // Select the global a
    a--; // The a at class scope
}
```

C vs. C++

```
typedef struct point {
    float x;
    float y;
} Point;

void print(const Point* p);
void move(Point* p,int dx,
int dy);

Point a;
a.x = 1; a.y = 2;
move(&a,2,2);
print(&a);
```

```
class Point {
public:
    void init(int x,int y);
    void print() const;
    void move(int dx,int dy);

private:
    float x;
    float y;
} ;

Point a;
a.init(1,2);
a.move(2,2);
a.print();
```

Definition of a class

- In C++, separated .h and .cpp files are used to define one class.
- Class declaration and prototypes in that class are in the header file (.h).
- All the bodies of these functions are in the source file (.cpp).

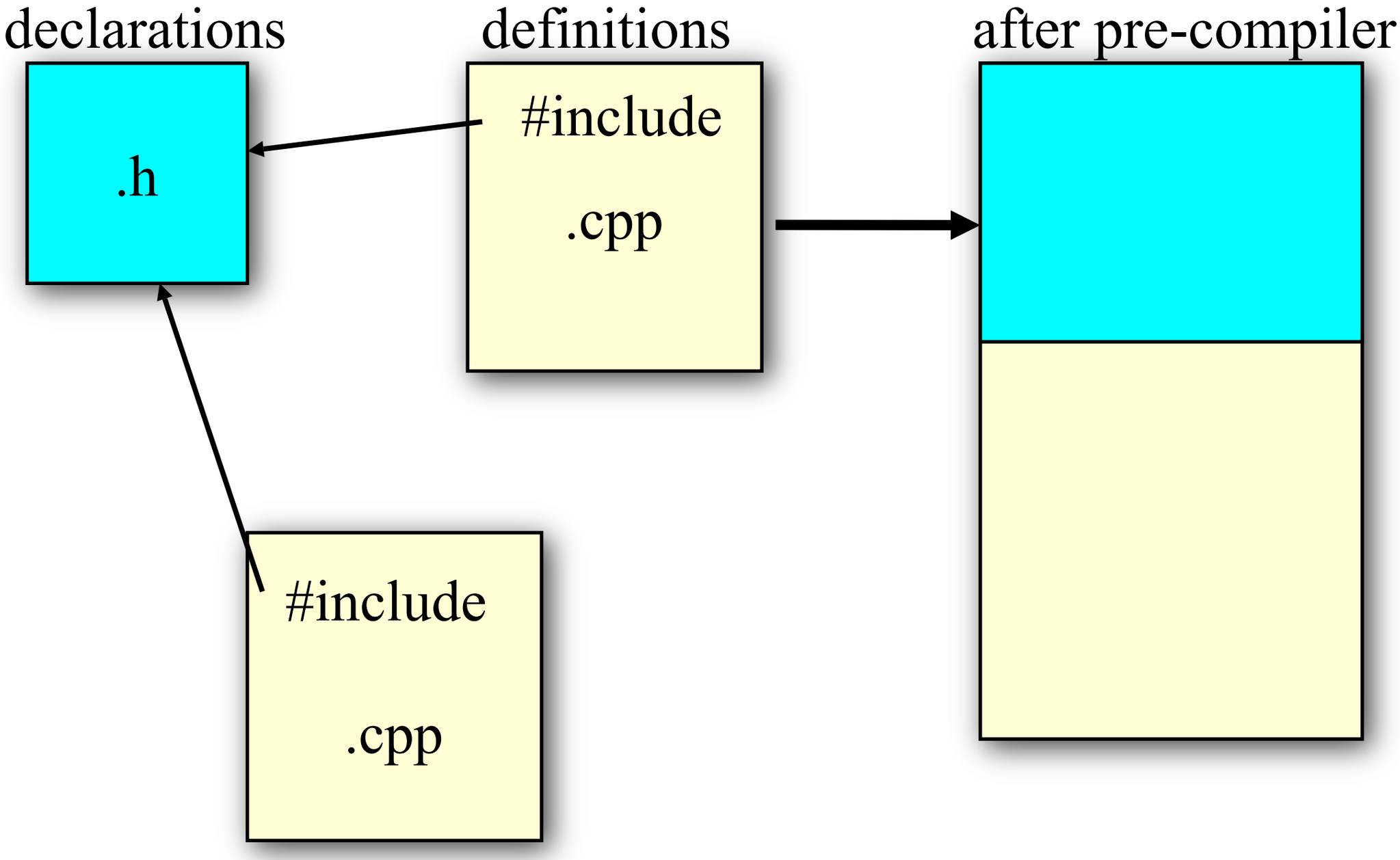
The header files

- If a function is declared in a header file, you *must* include the header file everywhere the function is used and where the function is defined.
- If a class is declared in a header file, you *must* include the header file everywhere the class is used and where class member functions are defined.

Header = interface

- The header is a contract between you and the user of your code.
- The compile enforces the contract by requiring you to declare all structures and functions before they are used.

Structure of C++ program



Other modules that use the functions

Declarations vs. Definitions

- A .cpp file is a compile unit
- Only declarations are allowed to be in .h
  - extern variables
  - function prototypes
  - class/struct declaration

Standard header file structure

```
#ifndef HEADER_FLAG
#define HEADER_FLAG

// Type declaration here...

#endif // HEADER_FLAG
```

#include

- #include is to insert the included file into the .cpp file at where the #include statement is.
  - #include “xx.h”:first search in the current directory, then the directories declared somewhere
  - #include <xx.h>:search in the specified directories
  - #include <xx>:same as #include <xx.h>

Tips for header

1. One class declaration per header file
2. Associated with one source file in the same prefix of file name.
3. The contents of a header file is surrounded with `#ifndef #define #endif`

Call functions in a class

```
Point a;

a.print();
```

- There is a relationship with the function be called and the variable calls it.
- The function itself knows it is doing something with the variable.

this: the hidden parameter

- **this** is a hidden parameter for all member functions, with the type of the struct

```
void Stash::initialize(int sz)
```

➔ (can be regarded as)

```
void Stash::initialize(Stash*this, int sz)
```

this: the hidden parameter

- To call the function, you must specify a variable

```
Stash a;
a.initialize(10);
```

➔ (can be regarded as)

```
Stash::initialize(&a,10);
```

- Example: this.cpp

this: pointer to the caller

- Inside member functions, you can use **this** as the pointer to the variable that calls the function.
- **this** is a natural local variable of all structs member functions that you can not define, but can use it directly.

- Example: Integer.h, Integer.cpp

The size of an object

- The size of a **struct** is the combined size of all of its members.
- Structures with no data members will always have some minimum nonzero size.

Examples:Sizeof.cpp

Guaranteed initialization with the constructor

- If a class has a constructor, the compiler automatically calls that constructor at the point an object is created, before client programmers can get their hands on the object.
- The name of the constructor is the same as the name of the class.

How a constructor does?

```
class X {
      int i;
public:
      X();
};
```

constructor

```
void f() {
      X a;
      // ...
}
```

a.X();

Constructors with arguments

- The constructor can have arguments to allow you to specify how an object is created, give it initialization values, and so on.

```
Tree(int i) {…}

Tree t(12);
```

The destructor

- In C++, cleanup is as important as initialization and is therefore guaranteed with the destructor.
- The destructor is named after the name of the class with a leading tilde (~). The destructor never has any arguments.

```
class Y {
public:
    ~Y();
};
```

When is a destructor called?

- The destructor is called automatically by the compiler when the object goes out of scope.
- The only evidence for a destructor call is the closing brace of the scope that surrounds the object.

Example

- Constructor1.cpp