

Lecture 2b: C++ Classes

CS 70: Data Structures and Program Development
Thursday, January 30

1

Learning Goals

1. I understand how to break C++ into code files and header files.
2. I can write C++ classes (as header and code files).

2

Terminology: Java → C++

- “superclass” → “base class”
- “subclass” → “derived class”
- “field” or “instance variable” → “data member”
- “method” → “member function”

4

5

Header Files

■ Separating *interface* and *implementation*

- .hpp file contains the class *declaration*
 - What it contains, what it can do
- A file that `#includes` the .hpp can use the class
 - The compiler knows about the class
- .cpp file contains the member function *definitions*
 - The instructions for each function

6

Header Files

- .cpp file is compiled to .o and linked into the final executable
 - Now the executable has the instructions
- `#include "cow.hpp"` in both `main.cpp` and `cow.cpp`. Why?

7

Separate Compilation

■ To make executable:

- compile `cow.cpp`
- compile `main.cpp`
- DO NOT compile `cow.hpp`
- link `cow.o` and `main.o`

8

Data Members

```
size_t spots_;  
size_t age_;
```

This is what defines what a Cow object looks like in memory.

9

Member Functions

Declare them in the .hpp:

```
void moo(size_t numMoos);
```

Implement in the .cpp:

```
void Cow::moo(size_t numMoos) {...}
```

Call them with dot(.):

```
bessie.moo(1)
```

10

Scope resolution operator ::

In the implementation file, need to say which class's method we are implementing.

We might have a cow and a sheep that both eat differently

```
void cow::eat(){  
    cout << "eating corn" << endl;  
}
```

```
void sheep::eat(){  
    cout << "eating grass" << endl;  
}
```

11

Semicolon at the end!

```
class ClassName{  
    //code  
}; // this semicolon is important
```

If you forget this semicolon, you could get “fun” errors.

12

13

Constructors

■ Default

- Parameterless constructor: `Cow()`
- Used for default initialization (e.g., `Cow bessie;`)
- Every class has one by default (default initializes members)

■ Parameterized

- Constructor with parameters:
`Cow(size_t numSpots, size_t age)`
- Must be invoked explicitly (e.g., `Cow bessie{numSpots, age};`)

■ Delete

- Used to disable the ability to call a function
- Most useful for implicitly/automatically defined functions
- e.g. `Cow() = delete;` ensures that there is no default constructor

14

Member initialization lists

```
Cow::Cow(size_t numSpots, size_t age)
: spots_{numSpots}, age_{age}
{
    cout << "Made a cow with " << spots_ <<
          " spots!" << endl;
}
```

16

15

17

Instantiating

```
Cow bessie{3,12}
```

- Use curly braces.
- This is modern style (different than Java and Python!).
- We will grade you on this in CS70.

18

The (C/C++) Preprocessor

```
#include <iostream>
#define C_STYLE_CONSTANT 42

int main()
{
    std::cout << C_STYLE_CONSTANT << "\n";
    #ifdef WINDOWS
        // ...code specific to Windows
    #else
        // ...alternate code for a Unix-based OS
    #endif
}
```

Processes your code BEFORE compiling.

19

Include Guards

```
#ifndef COW_HPP_INCLUDED
#define COW_HPP_INCLUDED

// more includes

class Cow{
    // data members and member functions
};

#endif // ifndef COW_HPP_INCLUDED
```

You are not allowed to declare something more than once!
Preprocessor trick that prevents code from being “copied” twice.

20

size_t

- Unsigned integer type.
- Need `#include<cstdint>` to use it.
- `typedef size_t = ... system dependent`

21

Extra Practice

Convert Point.java to C++.

```
public class Point {
    private int x_ = 0;
    private int y_ = 0;

    public Point(int x, int y) {
        x_ = x;
        y_ = y;
    }

    public void move(int deltaX, int deltaY){
        x_ += deltaX;
        y_ += deltaY;
    }
}
```

22

point.hpp

```
#ifndef POINT_HPP // C++ #include guard.
#define POINT_HPP

class Point {
public:
    Point(int x, int y);
    void move(int delta_x, int delta_y);

private:
    int x_;
    int y_;
};

#endif
```

point.cpp

```
#include "point.hpp"

Point::Point(int x, int y) : x_{x}, y_{y}
{
    // Nothing (left) to do!
}

void Point::move(int delta_x, int delta_y)
{
    x_ += delta_x;
    y_ += delta_y;
}
```

25