

# CSC 211: Computer Programming

## Header Files and Constructors

Michael Conti

Department of Computer Science and Statistics  
University of Rhode Island

Spring 2025



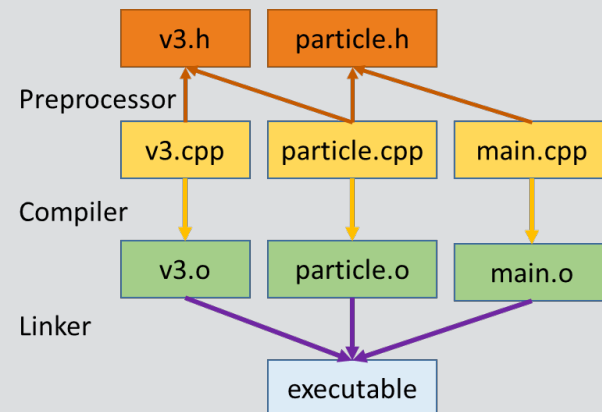
# Header Files

## Separate compilation

- Source code can be divided into multiple files
  - ✓ source files can be compiled separately
  - ✓ enterprise code files can take hours to compile
    - Source code separation eliminates the need to compile everything, all the time
- Classes can be implemented in their own files
  - ✓ allows reusing codes in multiple programs
  - ✓ source files including class methods and function definitions
  - ✓ header files including declarations and global constants

3

## Compiling multiple files



```
g++ v3.cpp particle.cpp main.cpp -o executable
```

<https://devblogs.nvidia.com/separate-compilation-linking-cuda-device-code/>

4

## #include

- Used for including header files
  - usually contains class declarations, function prototypes, or global constants
- When used with < >
  - The preprocessor searches in an implementation dependent manner, normally in search directories pre-designated by the compiler/IDE. This method is normally used to include standard library header files
- When used with " "
  - The preprocessor searches first in the same directory as the file containing the directive, and then follows the search path used for the #include <filename> form. This method is normally used to include programmer-defined header files.
- Cannot compile header files directly!

5

## Multiple declarations of classes

- With large projects, multiple declaration of classes must be prevented

- Use #ifndef

```
#ifndef DATE_H  
#define DATE_H
```

```
class Date {  
    // ...  
};
```

```
#endif
```

6

## Multiple declarations of classes

- Do header guards need to be capital or use an underscore instead of a dot?
- Preprocessor definitions have to use valid identifiers.
- Dots are not valid in identifiers. There is also a convention that preprocessor definitions (especially preprocessor macros) use all-uppercase names, to distinguish them from non-preprocessor identifiers.
- Not a hard and fast rule, just convention

7

## Constructors

# Constructors

- Special `methods` used to initialize data members when objects are created
- A constructor ...
  - ✓ ... is a member function (usually **public**)
  - ✓ ... must have the same name as its class
  - ✓ ... is automatically called when an object is created
  - ✓ ... does not have a return type (not even **void**)

constructors cannot be called as other methods

9

# Example

```
class Date {  
    private:  
        int month;  
        int year;  
        int day;  
  
    public:  
        Date();  
        // ...  
};
```

No return  
value

10

# Example: Date

```
#ifndef DATE_H  
#def DATE_H  
class Date {  
    private:  
        int month;  
        int year;  
        int day;  
  
    public:  
        Date();  
        void print();  
};  
#endif
```

```
#include "date.h"  
  
int main() {  
    Date mydate;  
  
    mydate.print();  
}
```

```
#include "date.h"  
#include <iostream>  
  
Date::Date() {  
    month = 1;  
    day = 1;  
    year = 1970;  
}  
  
void Date::print() {  
    std::cout << month << '-' <<  
    day << '-' << year << '\n';  
}
```

g++ date.cpp main.cpp -o exec

11

# Overloading constructors

- A constructor with no parameters is also known as the **default constructor**
- Classes may have multiple constructors
  - ✓ constructors are **overloaded** by defining constructors with different parameter lists

```
Date();  
Date(int m, int d, int y);
```

12

## Synthesized default constructor

- If you don't define any constructor, C++ will define one default constructor for you
- If you define at least one constructor, C++ will not add any other (not even the default constructor)

13

## Initialization lists

- C++ allows for optional initialization lists as part of the constructor definition

```
Date::Date(int _d, int _m, int _y) {  
    day = _d;  
    month = _m;  
    year = _y;  
    // more statements  
}
```

Same as...

```
Date::Date(int _d, int _m, int _y) : day(_d), month(_m), year(_y) {  
    // more statements  
}
```

14

## Lets Try it

- Modify Point2D.cpp (on GitHub at ~/code) so it includes the following:
  - ✓ Default Constructor
  - ✓ Parameterized Constructor
- Once working, break it up into:
  - ✓ Class file (Point2D.cpp)
  - ✓ Header/Interface file (Point2D.h)
  - ✓ Driver (main.cpp)

15