Part 2

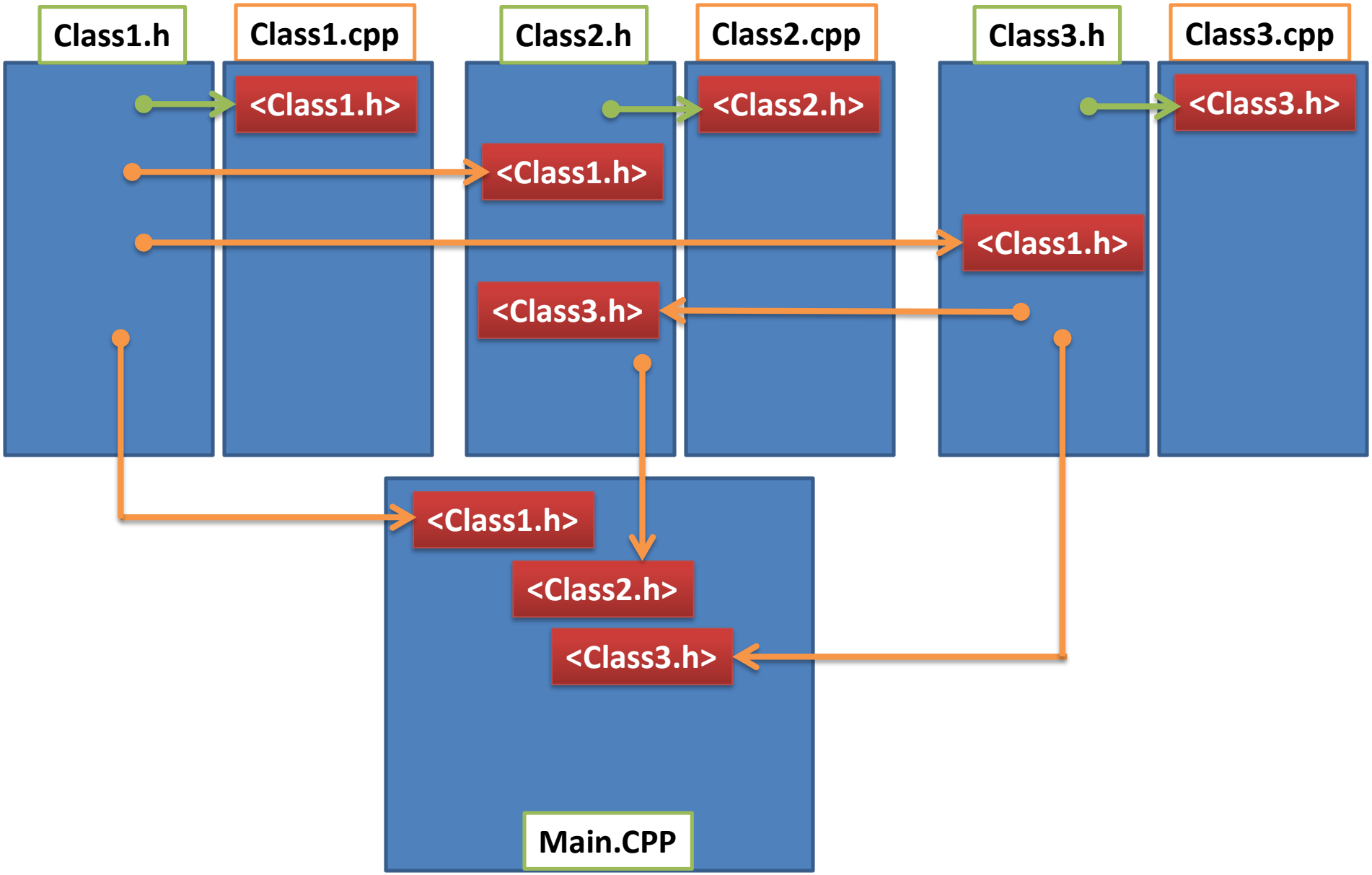
Classes: A Deeper Look

Class Construction: C++

Contents

- Review: Preprocessor wrappers
- Review: Order of Constructors and Destructors Calling
- Constants Objects and Member Functions
- friend Functions and friend Classes

Preprocessor Wrapper



Constructors and Destructors Calling

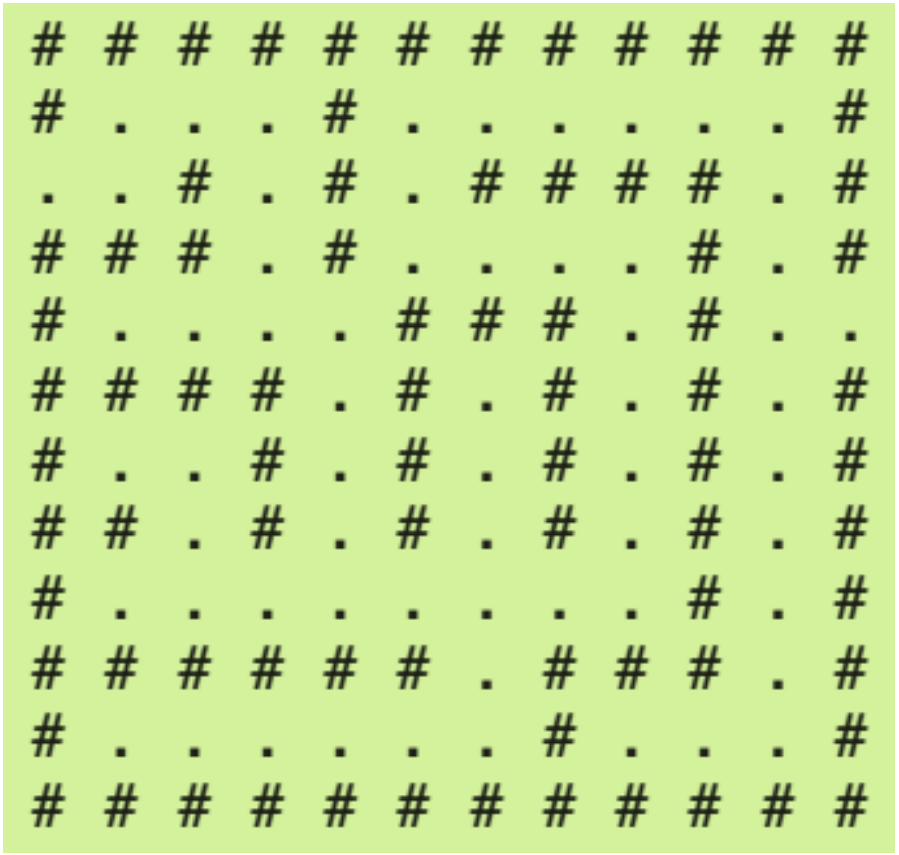
```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)

Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```

Earn Score!

Maze Traversal

- The grid of hashes (#) and dots (.) is a two-dimensional array representation of a maze.
- In the two-dimensional array, the hashes represent the walls of the maze and the dots represent squares in the possible paths through the maze.
- Moves can be made only to a location in the array that contains a dot.

- There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming that there is an exit).
- If there is not an exit, you'll arrive at the starting location again.



Earn Score!

- Write recursive function mazeTraverse to walk through the maze.
- The function should receive arguments that include a 12-by-12 character array representing the maze and the starting location of the maze.
- As mazeTraverse attempts to locate the exit from the maze, it should place the character X in each square in the path.
- The function should display the maze after each move, so the user can watch as the maze is solved

Generating Maze Randomly

- Write a function mazeGenerator that randomly produces a maze.
- The function should take as arguments a two-dimensional 12-by-12 character array and pointers to the int variables that represent the row and column of the maze's entry point.
- Try your function mazeTraverse, using several randomly generated mazes.

const (Constant) Objects and const Member Functions

- Some objects need to be modifiable and some do not.

```
const Time noon( 12, 0, 0 );
```

- C++ disallows member function calls for const objects unless the member functions themselves are also declared const.

- A member function is specified as const *both* in its prototype and in its definition

```
// Fig. 10.1: Time.h
// Time class definition with const member functions.
// Member functions defined in Time.cpp.
#ifndef TIME_H
#define TIME_H

class Time
{
public:
   Time( int = 0, int = 0, int = 0 ); // default constructor

   // set functions
   void setTime( int, int, int ); // set time
   void setHour( int ); // set hour
   void setMinute( int ); // set minute
   void setSecond( int ); // set second

   // get functions (normally declared const)
   int getHour() const; // return hour
   int getMinute() const; // return minute
   int getSecond() const; // return second

   // print functions (normally declared const)
   void printUniversal() const; // print universal time
   void printStandard(); // print standard time (should be const)
private:
   int hour; // 0 - 23 (24-hour clock format)
   int minute; // 0 - 59
   int second; // 0 - 59
}; // end class Time

#endif
```

```cpp
// Fig. 10.2: Time.cpp
// Time class member-function definitions.
#include <iostream>
#include <iomanip>
#include "Time.h" // include definition of class Time
using namespace std;

// constructor function to initialize private data;
// calls member function setTime to set variables;
// default values are 0 (see class definition)
Time::Time( int hour, int minute, int second )
{
   setTime( hour, minute, second );
} // end Time constructor

// set hour, minute and second values
void Time::setTime( int hour, int minute, int second )
{
   setHour( hour );
   setMinute( minute );
   setSecond( second );
} // end function setTime
```

```cpp
// set hour value
void Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
} // end function setHour

// set minute value
void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0; // validate minute
} // end function setMinute

// set second value
void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ? s : 0; // validate second
} // end function setSecond

// return hour value
int Time::getHour() const // get functions should be const
{
   return hour;
} // end function getHour

// return minute value
int Time::getMinute() const
{
   return minute;
} // end function getMinute

// return second value
int Time::getSecond() const
{
   return second;
} // end function getSecond

// print Time in universal-time format (HH:MM:SS)
void Time::printUniversal() const
{
   cout << setfill( '0' ) << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute << ":" << setw( 2 ) << second;
} // end function printUniversal

// print Time in standard-time format (HH:MM:SS AM or PM)
void Time::printStandard() // note lack of const declaration
{
   cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
      << ":" << setfill( '0' ) << setw( 2 ) << minute
      << ":" << setw( 2 ) << second << ( hour < 12 ? " AM" : " PM" );
} // end function printStandard
```

```
// Fig. 10.3: fig10_03.cpp
// Attempting to access a const object with non-const member functions.
#include "Time.h" // include Time class definition

int main()
{
   Time wakeUp( 6, 45, 0 ); // non-constant object
   const Time noon( 12, 0, 0 ); // constant object

                            // OBJECT      MEMBER FUNCTION
   wakeUp.setHour( 18 );    // non-const   non-const

   noon.setHour( 12 );      // const       non-const

   wakeUp.getHour();        // non-const   const

   noon.getMinute();        // const       const
   noon.printUniversal();   // const       const

   noon.printStandard();    // const       non-const
} // end main
```

Initializing a const Data Member with a Member Initializer

- All data members *can* be initialized using member initializer syntax

- But const data members and data members that are references *must* be initialized using member initializers.

```
class Increment
{
public:
   Increment( int c = 0, int i = 1 ); // default constructor

   // function addIncrement definition
   void addIncrement()
   {
      count += increment;
   } // end function addIncrement

   void print() const; // prints count and increment
private:
   int count;
   const int increment; // const data member
}; // end class Increment
```

Initializing a const Data Member with a Member Initializer

```
#include <iostream>
#include "Increment.h" // include definition of class Increment
using namespace std;

// constructor
Increment::Increment( int c, int i )
   : count( c ), // initializer for non-const member
     increment( i ) // required initializer for const member
{
   // empty body
} // end constructor Increment
```

friend Functions and friend Classes

- A **friend function** of a class is defined outside that class's scope, yet has the right to access the non-public (and public) members of the class.
- Standalone functions, entire classes or member functions of other classes may be declared to be friends of another class.

- Purpose: enhance performance.

To declare all member functions of class **ClassTwo** as friends of class **ClassOne**, place a declaration of the form

```
friend class ClassTwo;
```

in the definition of class ClassOne

1) Friendship is granted, not taken

2) the friendship relation is neither symmetric nor transitive

Modifying a Class's private Data with a Friend Function

```
// Fig. 10.15: fig10_15.cpp
// Friends can access private members of a class.
#include <iostream>
using namespace std;

// Count class definition
class Count
{
   friend void setX( Count &, int ); // friend declaration
public:
   // constructor
   Count()
      : x( 0 ) // initialize x to 0
   {
      // empty body
   } // end constructor Count
```

```
   // output x
   void print() const
   {
      cout << x << endl;
   } // end function print
private:
   int x; // data member
}; // end class Count

// function setX can modify private data of Count
// because setX is declared as a friend of Count (line 9)
void setX( Count &c, int val )
{
   c.x = val; // allowed because setX is a friend of Count
} // end function setX

int main()
{
   Count counter; // create Count object

   cout << "counter.x after instantiation: ";
   counter.print();

   setX( counter, 8 ); // set x using a friend function
   cout << "counter.x after call to setX friend function: ";
   counter.print();
} // end main
```