# Separate Files, Constructors

ITP 165 – Fall 2015

Week 10, Lecture 1

## Our Clock code

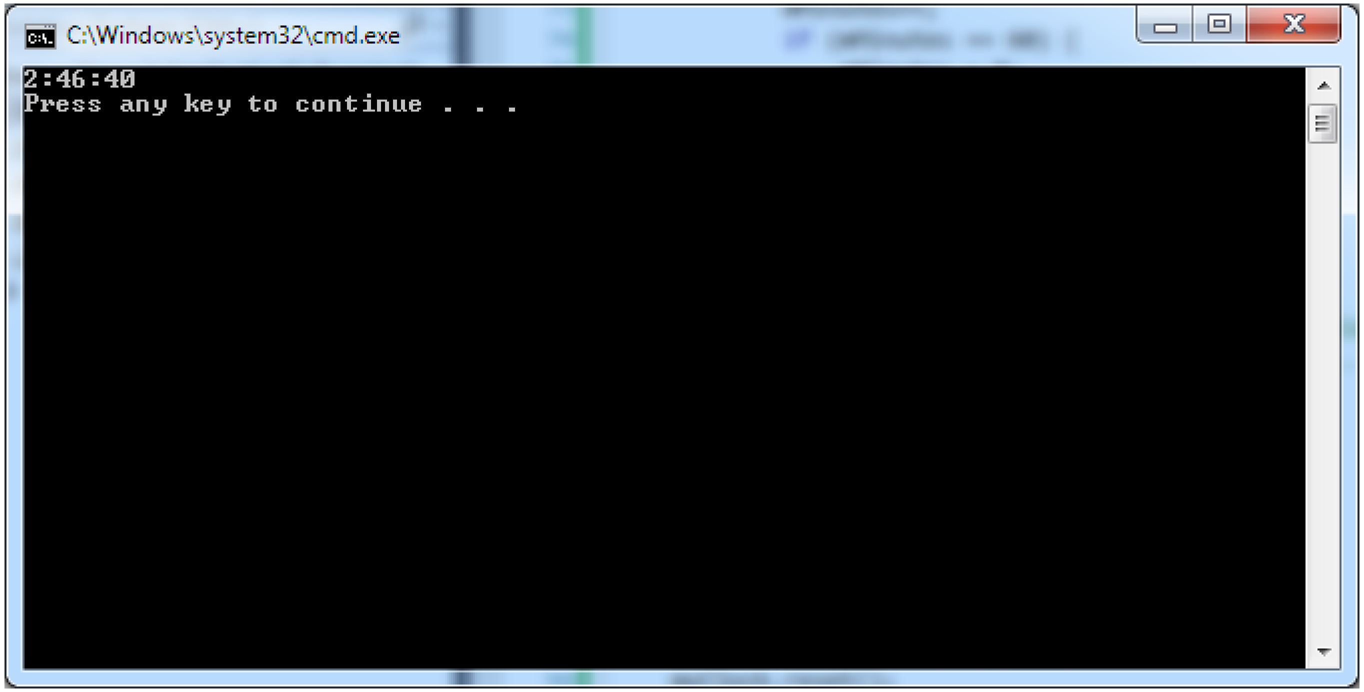
- This is where we left off…

```cpp
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   void reset() {
      mHours = 0;
      mMinutes = 0;
      mSeconds = 0;
   }

   void print() {
      std::cout << mHours << ":";
      std::cout << mMinutes << ":";
      std::cout << mSeconds << std::endl;
   }

   void tick() {
      mSeconds++;
      if (mSeconds == 60) {
         mSeconds = 0;
         mMinutes++;
         if (mMinutes == 60) {
            mMinutes = 0;
            mHours++;
            if (mHours == 24) {
               mHours = 0;
            }
         }
      }
   }
};
```

## Full Clock class in Action

```
int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   // Call print member function
   myClock.print();

   return 0;
}
```

## Full Clock class in Action

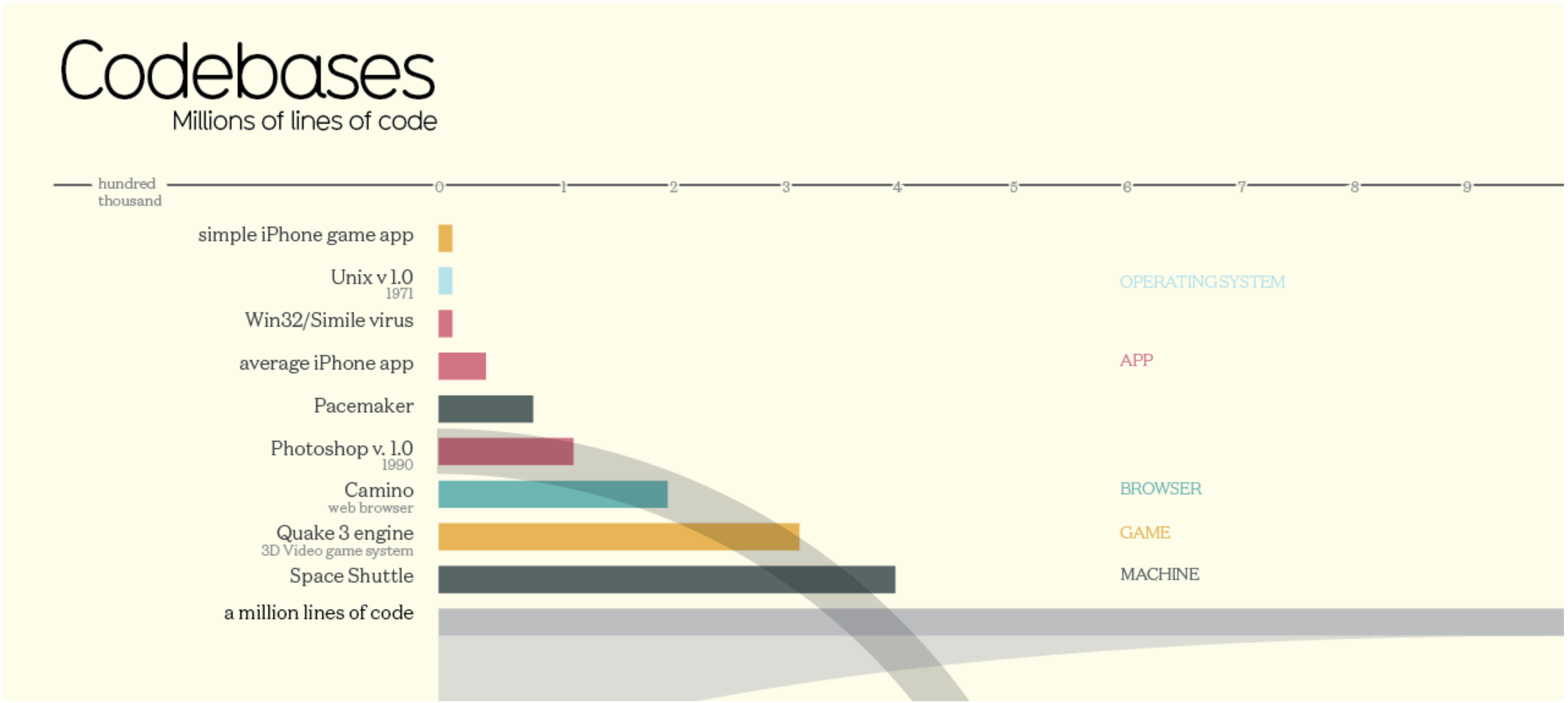


```
2:46:40
Press any key to continue . . .
```

# A Problem…

- Check it out…

  http://www.informationisbeautiful.net/visualizations/million-lines-of-code/



## Multiple Files

- Putting “millions” of lines of code in one file would be a nightmare

- So most serious programming languages support more than one file.

- Let’s look at how it’s done in C++…

## Multiple Files, Cont'd

- Usually, if we want to separate a class out into another files, we create two new files…
- A ***header file*** (ends in .h)
- An ***implementation file*** (ends in .cpp)
- So in the case of Clock, if we wanted to move it to separate files, we'd create:
  - Clock.h
  - Clock.cpp

## Header File

- The header file typically contains:
  - The name of the class you want to declare
  - The member variables
  - The return types and parameter types of the functions

- But…it does not contain any of the code inside of the functions!!!

## Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
};
```

## Clock.h – A closer look

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
};
```

You should always put #pragma once at the top of every header file!

## Clock.h – A closer look

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
};
```

The class and member variables are declared in the same way as they were previously.

## Clock.h – A closer look

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
};
```

Notice how we are just specifying the name and parameters of the functions, but no actual code inside of them!

## Clock.h – A closer look

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
};
```

Ends the same

## Implementation File

- The implementation file (.cpp) contains the actual code inside of each member function
- We don’t need to re-declare the name of the class or member variables, those are handled by the header

## Clock.cpp

```cpp
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

# Clock.cpp – A closer look

```
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

The first line should always be a #include of the header associated with this .cpp. (Notice the quotes instead of less than/greater than)

## Clock.cpp – A closer look

```
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

We then need to include any additional libraries that are needed by the function implementations – in this case, we need iostream for the print function in Clock.

# Clock.cpp – A closer look

```
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

This is the implementation for the reset function in Clock.

Notice how we specify the function as being Clock::reset, not just reset!

## Clock.cpp – A closer look

```
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

Same idea for print/tick functions

# Clock.cpp – A closer look

```
#include "Clock.h"
#include <iostream>

void Clock::reset() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
void Clock::print() {
   std::cout << mHours << ":";
   std::cout << mMinutes << ":";
   std::cout << mSeconds << std::endl;
}
void Clock::tick() {
   mSeconds++;
   if (mSeconds == 60) {
      mSeconds = 0;
      mMinutes++;
      if (mMinutes == 60) {
         mMinutes = 0;
         mHours++;
         if (mHours == 24) {
            mHours = 0;
         }
      }
   }
}
```

Notice how Clock.cpp just ends after the last declaration.

## Using Clock in our main program file

```
#include "Clock.h"

int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   myClock.print();

   return 0;
}
```

## Using Clock in our main program file

```
#include "Clock.h"

int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   myClock.print();

   return 0;
}
```

We can now include the Clock.h library in our main program file (though notice the quotes)

## Using Clock in our main program file

```
#include "Clock.h"

int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   myClock.print();

   return 0;
}
```

We can now use the Clock class same as before.

## Adding More Functions – Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
};
```

## Adding More Functions – Clock.cpp

- Add on to the bottom of Clock.cpp, after the other declarations…

```
int Clock::getHours() {
   return mHours;
}

void Clock::setHours(int newHours) {
   if (newHours >= 0 && newHours <= 23) {
      mHours = newHours;
   }
}
```

## Main Program File

```
#include "Clock.h"
#include <iostream>

int main() {
   Clock myClock;

   // Call reset member function
   myClock.reset();

   // Call tick member function 10,000 times
   for (int i = 0; i < 10000; i++) {
      myClock.tick();
   }

   myClock.print();

   std::cout << myClock.getHours() << std::endl;

   return 0;
}
```

## Member function that takes in another instance

- Suppose we want a member function that can compare two instances of `Clock`
- For example, we want to see if two `Clock`s have the same time…
- Unfortunately, code like this would not work:

```
Clock clockOne;
Clock clockTwo;

// Error: == is not defined for 'Clock'
bool same = (clockOne == clockTwo);
```

## isEqual – Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor
   Clock();
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
   // Returns true if this object has the same
   // time as the other object.
   bool isEqual(Clock& other);
};
```

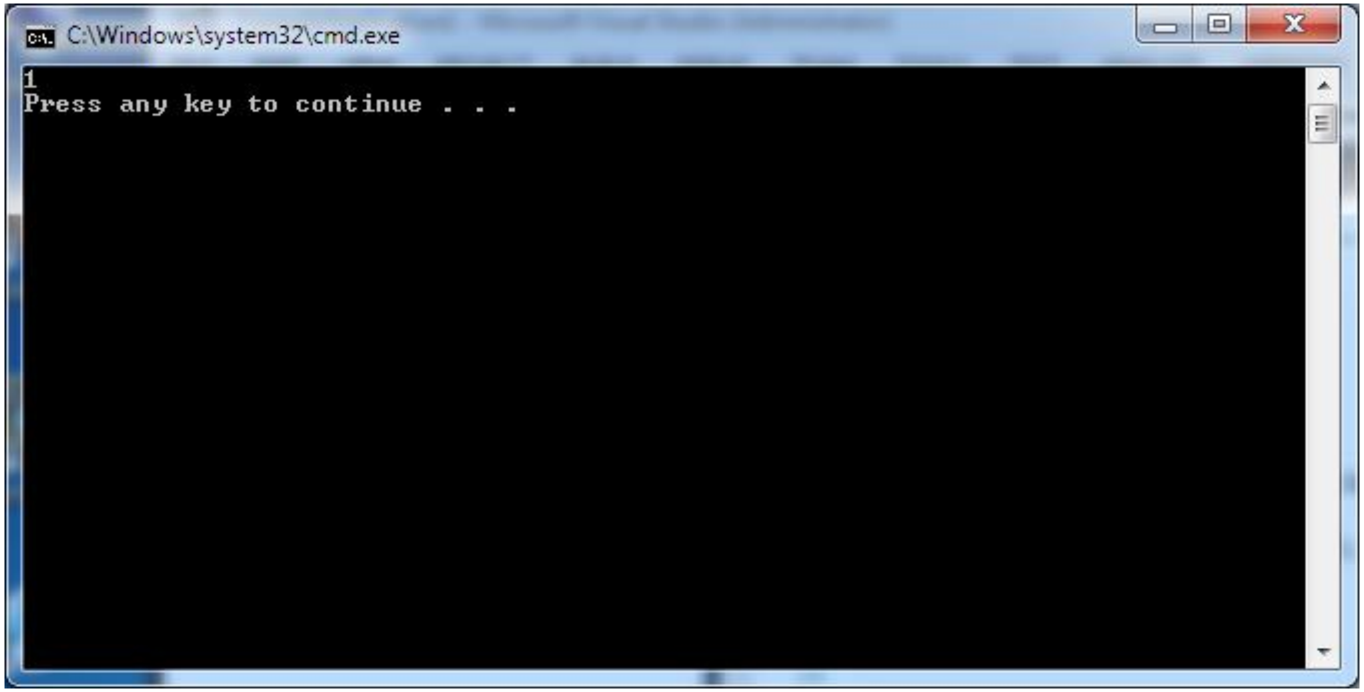
## isEqual – Clock.cpp

```cpp
// Returns true if this object has the same
// time as the other object.
bool Clock::isEqual(Clock& other)
{
   if (mHours == other.mHours &&
       mMinutes == other.mMinutes &&
       mSeconds == other.mSeconds)
   {
      return true;
   }
   else
   {
      return false;
   }
}
```

## isEqual in Action

```
#include "Clock.h"
#include <iostream>

int main() {
   Clock clockOne;
   Clock clockTwo;

   clockOne.reset();
   clockTwo.reset();

   bool same = clockOne.isEqual(clockTwo);

   std::cout << same << std::endl;

   return 0;
}
```

## isEqual in Action



```
1
Press any key to continue . . .
```

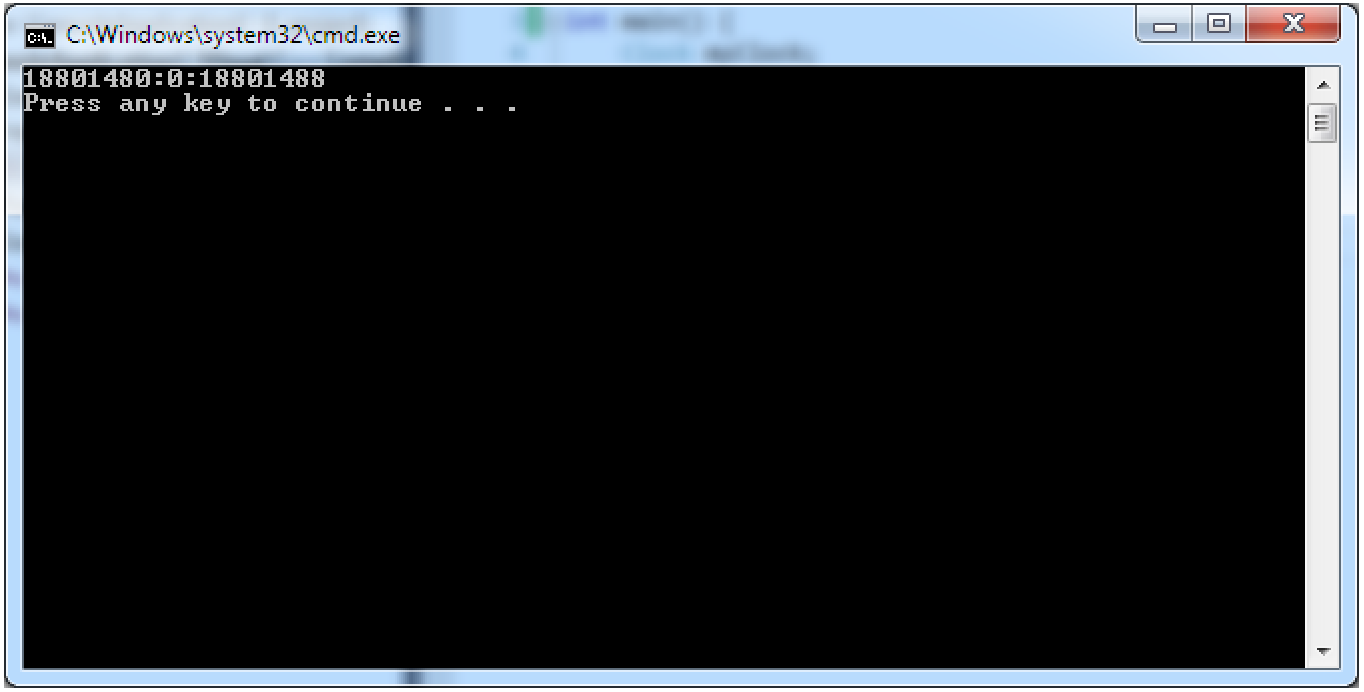
## Issue: Uninitialized member variables

- What happens if we run this?

```
#include "Clock.h"

int main() {
   Clock myClock;

   myClock.print();

   return 0;
}
```

## Issue: Uninitialized member variables

- The member variables never get set, so we get random garbage data:



```
18801480:0:18801488
Press any key to continue . . .
```

## Default Constructor

- A ***constructor*** is a special type of member function that is automatically called when an instance of the class is created
- A ***default constructor*** is a constructor that takes no parameters
- So in this case, we could create a default constructor that automatically sets the time to midnight.

## Default Constructor – Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor
   Clock();
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
};
```

## Default Constructor – Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor
   Clock();
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
};
```

Name of constructor must be the same name as the class.

## Default Constructor – Clock.h

```cpp
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor
   Clock();
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
};
```

We don't specify any return type, not even void. This is because constructors can't return anything, period.

## Default Constructor – Clock.h

```
#pragma once

class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor
   Clock();
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
};
```

Since this is a default constructor, no parameters.

## Default Constructor – Clock.cpp

- We could add the following to Clock.cpp

```
Clock::Clock() {
   mHours = 0;
   mMinutes = 0;
   mSeconds = 0;
}
```

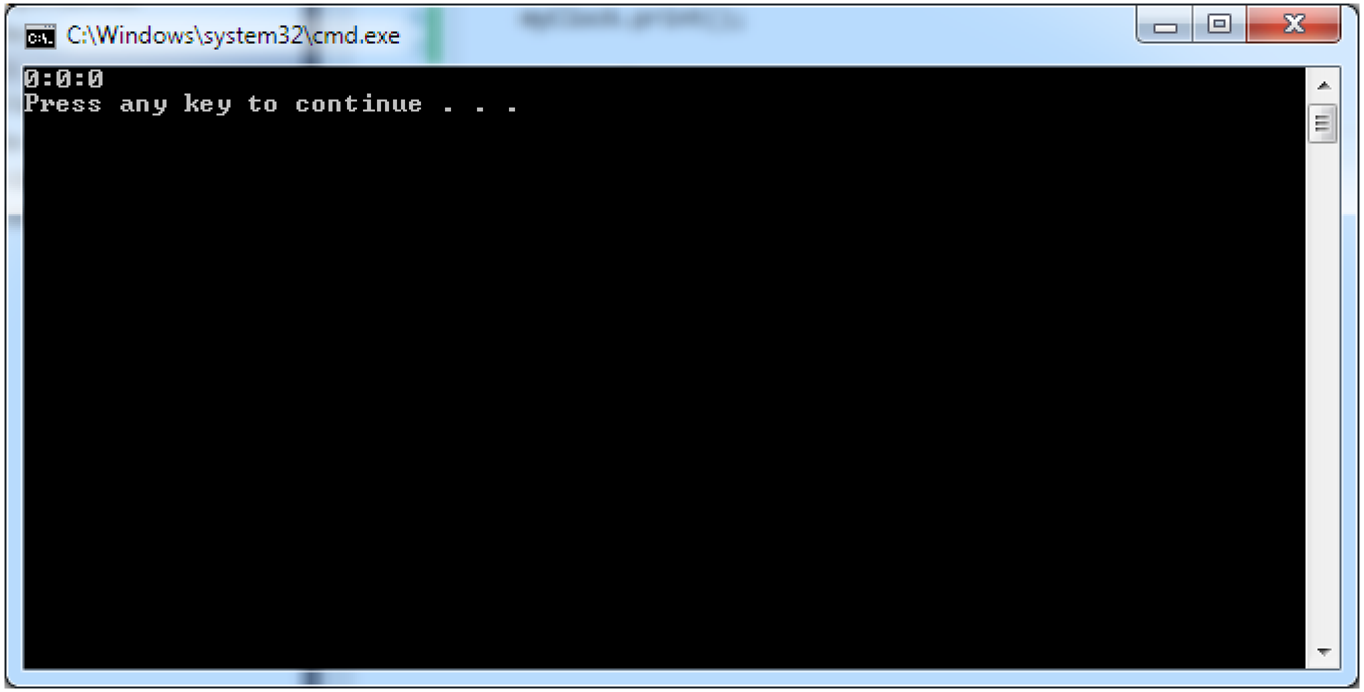
- Notice that again, we don't specify any return type here.

## Now if I run this program…

```
#include "Clock.h"

int main() {
   Clock myClock;

   myClock.print();

   return 0;
}
```

## Result

- I get all 0s, because the default constructor is automatically called



```
0:0:0
Press any key to continue . . .
```

## Default Constructor, Cont'd

- Since a default constructor is just any other member function, it can:
  - Access all member variables
  - Access all member functions

- So a better way to write the code for `Clock`'s default constructor might be:

```
Clock::Clock() {
   // Call the reset member function,
   // which resets to midnight!
   reset();
}
```

# A Constructor with Parameters – Clock.h

```
#pragma once

// Represents a clock in military time
class Clock {
private:
   int mHours;
   int mMinutes;
   int mSeconds;
public:
   // Default constructor (sets to midnight)
   Clock();
   // Constructor to set to specific time
   Clock(int hours, int minutes, int seconds);
   // Resets clock to midnight
   void reset();
   // Prints the clock H:M:S
   void print();
   // Advances clock by one second
   void tick();
   // Get/set hours
   int getHours();
   void setHours(int newHours);
   // Returns true if this object has the same
   // time as the other object.
   bool isEqual(Clock& other);
};
```

## Implementation of Constructor w/ Parameters

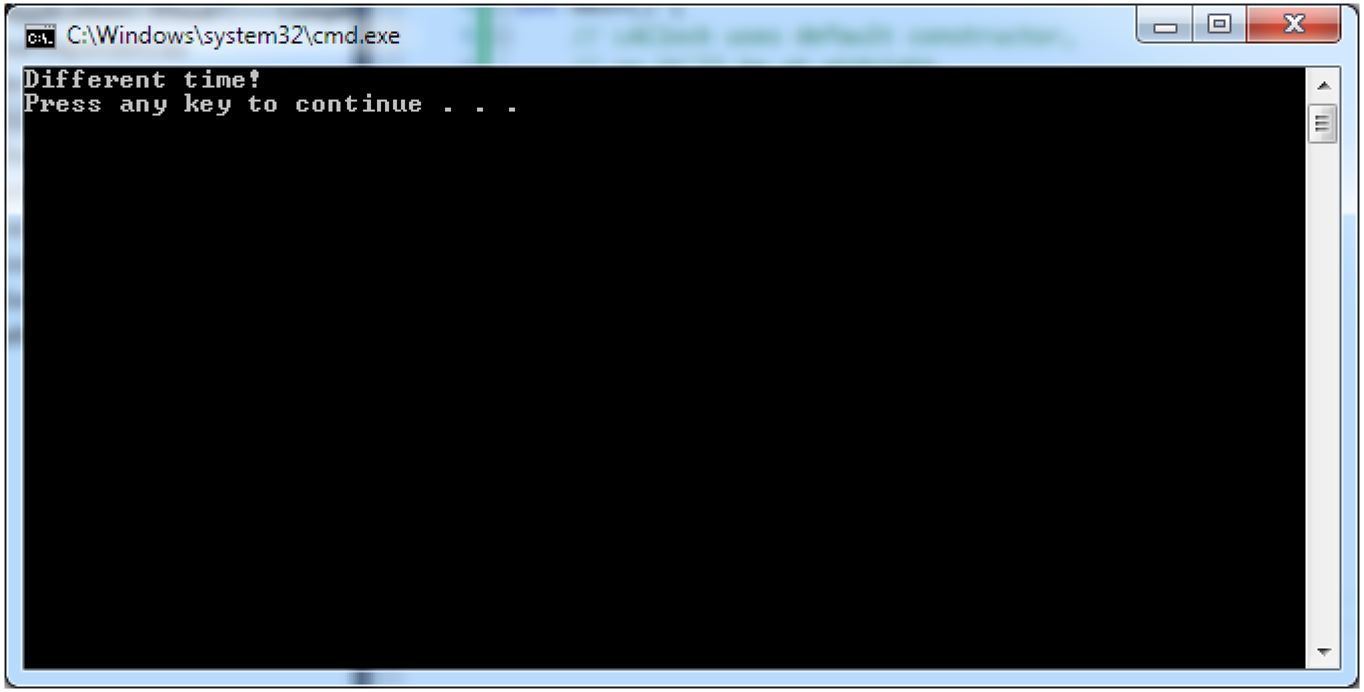
- In Clock.cpp, we then can add the following implementation:

```
Clock::Clock(int hours, int minutes, int seconds)
{
    mHours = hours;
    mMinutes = minutes;
    mSeconds = seconds;
}
```

## Using the Different Constructors

```cpp
#include "Clock.h"
#include <iostream>

int main() {
   // LAClock uses default constructor,
   // so it'll be at midnight.
   Clock LAClock;

   // NYClock uses the constructor that
   // takes parameters, so it'll be 3 o'clock
   Clock NYClock(3, 0, 0);

   bool same = LAClock.isEqual(NYClock);

   if (same == true) {
      std::cout << "Same time!" << std::endl;
   } else {
      std::cout << "Different time!" << std::endl;
   }

   return 0;
}
```

## Using Different Constructors



```
Different time!
Press any key to continue . . .
```

## Requiring Parameters at Construction

- For some classes, you may not want to allow default construction (eg. you want to require parameters)

- If this is the case, simply declare your constructor with parameters in the .h/.cpp, but don't declare a default constructor

## Lab Practical #16