CS& 132 Lecture Topic 2 Intro to Classes

Lonnie Heinke

- Todays Agenda
  - OOP
    - Idea of **O**bject **O**riented **P**rogramming
  - Introduction to Classes
    - creating and using

- Start up Visual Studio 201?
  - Create a new project called "Sandbox2" that you will use for your activities.
  - Keep this project so you can use them as code examples in the future or you could expand on them.

Things that you should already know

Basic Topics that you should already know and be able to use well:

Primitive Data Types: int, double, char, bool

Operators: +, -, /, *, %, ++(pre and post), --(pre and post),
=, = =, >, <, >=, <=, !, !=, &&, | |, ::

User I/O: cin, cout, <<, >>, setw( ), setprecision( ), left, right, fixed, scientific, endl,

Boolean Expressions: if, if else, ?: operator

Loops: while, do while, for, *for each*

Switch blocks: switch, case, default, break, what data types can you "switch" on

Arrays: creating, indexing, single and multi-dimension
(ex: int student[5][20])

Functions: parameters {pass by value, pass by reference},
return type, function prototype, function definition,

Things that you should already know

Functions: parameters {pass by value, pass by reference, pass by pointer},
return type, function prototype, function definition,
parameters with a default value, function overloading

Vectors: create and use, [ ] vs at( ), size, capacity, 2 dimension vector

Variable Scope: local, global, local variables hiding global, ::

Pointers and dynamic memory: operators: *, &, ->, new, delete, how pointers can be used with arrays

File I/O: opening a file, checking for fail, reading, writing, closing, exit( )

Enumerations: creating and using

Structures: creating and using

Keywords: continue, break, return, const

Quick Review

What is the advantage/purpose of the following:

array:

pointer:

structure:

class: .....not a review.....a preview

Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer:

structure:

class:

Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer: can be used to access dynamic memory

structure:

class:

Quick Review

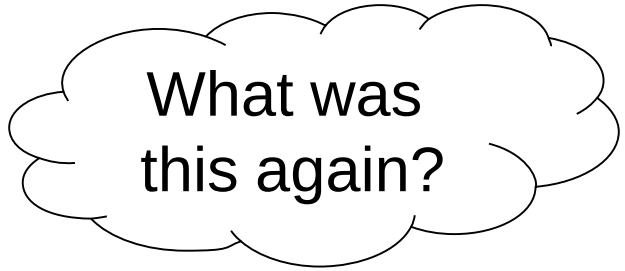
What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer: can be used to access **dynamic memory**

structure:

class:



Quick Review

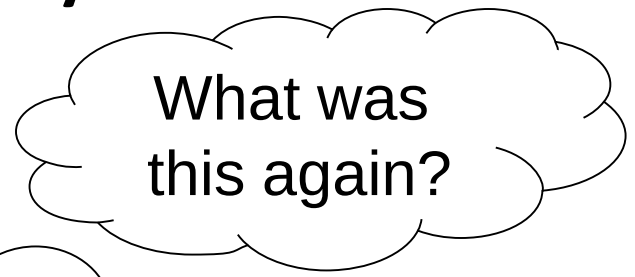
What is the advantage/purpose of the following:

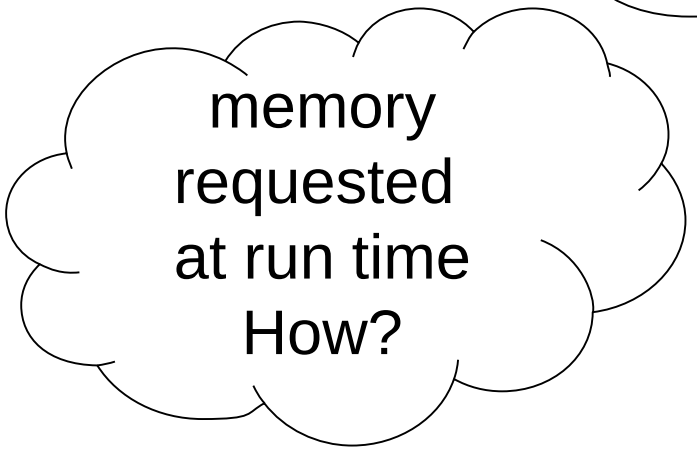
array: hold many pieces of data of the same type

pointer: can be used to access **dynamic memory**

structure:

class:





Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer: can be used to access **dynamic memory**

structure:

class:

What was this again?

memory requested at run time How?

new keyword

```
char * ptr = new char[20];
```

Quick dynamic memory review

```
char* ptr = new char[20] {"hello there"};   // create dyn memory

cout << ptr << endl;      // char pointer so it outputs the cstring
cout << *ptr << endl;     // ???

delete[] ptr;             // releases the dynamic memory
```

Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer: can be used to access dynamic memory

structure:

class:

Defining and Using a Structure

```
struct Time {    // before main
 int hour;
 int minute;
} ;


// in main...
Time startTime = { 9, 55 }, endTime;
endTime.hour = 11;
endTime.minute = 30;
```

Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

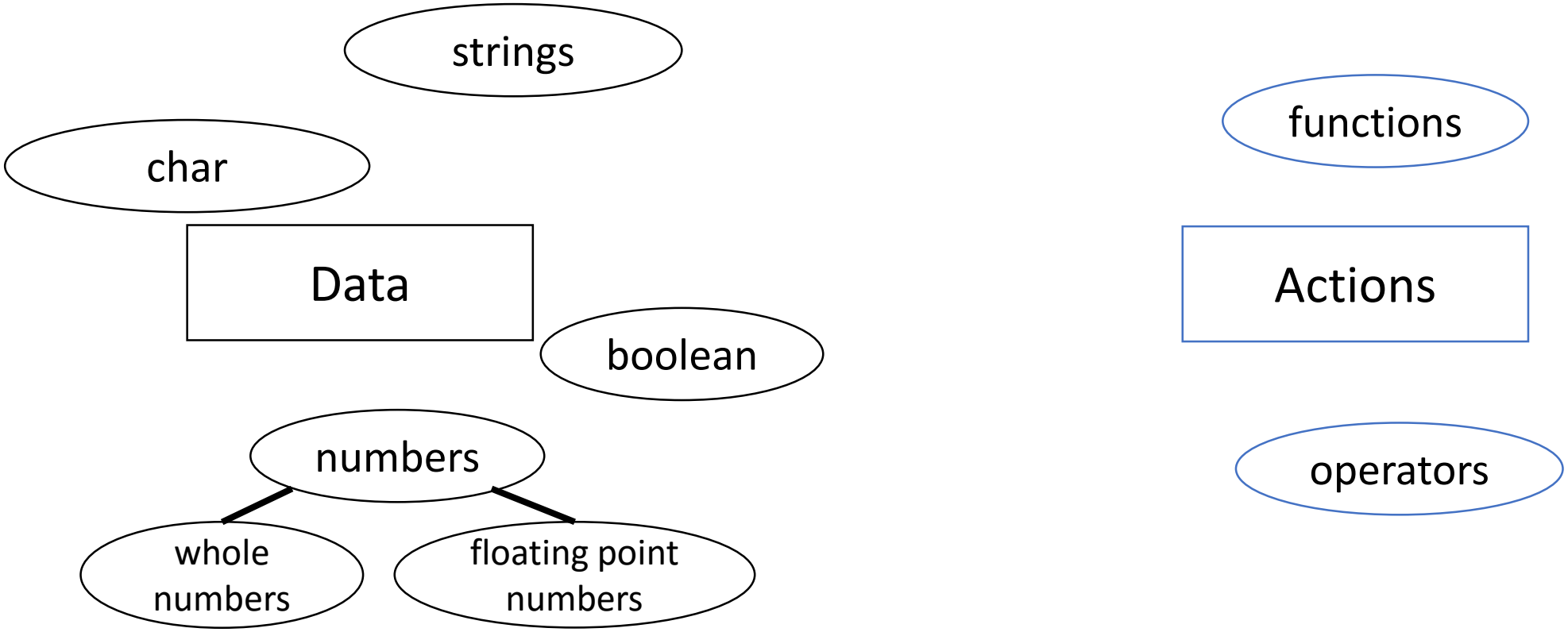
pointer: can be used to access dynamic memory

structure: a **new data type** that can hold different kinds of data

class: ???

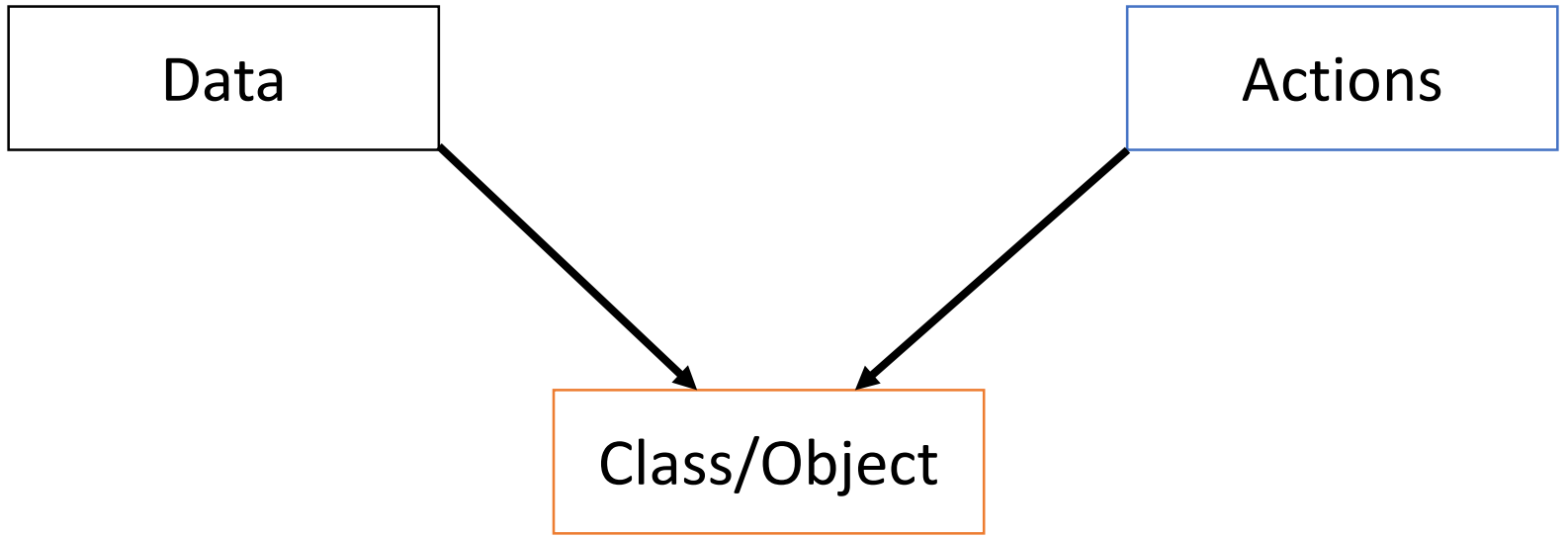
Building Blocks of programming

- So far, programs are made up of two main components:



Building Blocks of programming

- **Object Oriented Programming** brings these two main components together:



Quick Review

What is the advantage/purpose of the following:

array: hold many pieces of data of the same type

pointer: can be used to access dynamic memory

structure: a **new data type** that can hold different kinds of data

class: a **new data type** that can hold different kinds of data, and can have actions for that data (functions and operations)

Some Definitions that I want you to *memorize*

- **Object:** (general definition) a logical combination of data and actions
  - vector, string, Button

- **Class:** the definition of an object or the blue print of an object

- **Instance:** a variable that is created from a class (....sometimes called an object)

OOP ???

- Object Oriented Programming
  - towards
  - in relation to
  - concerning

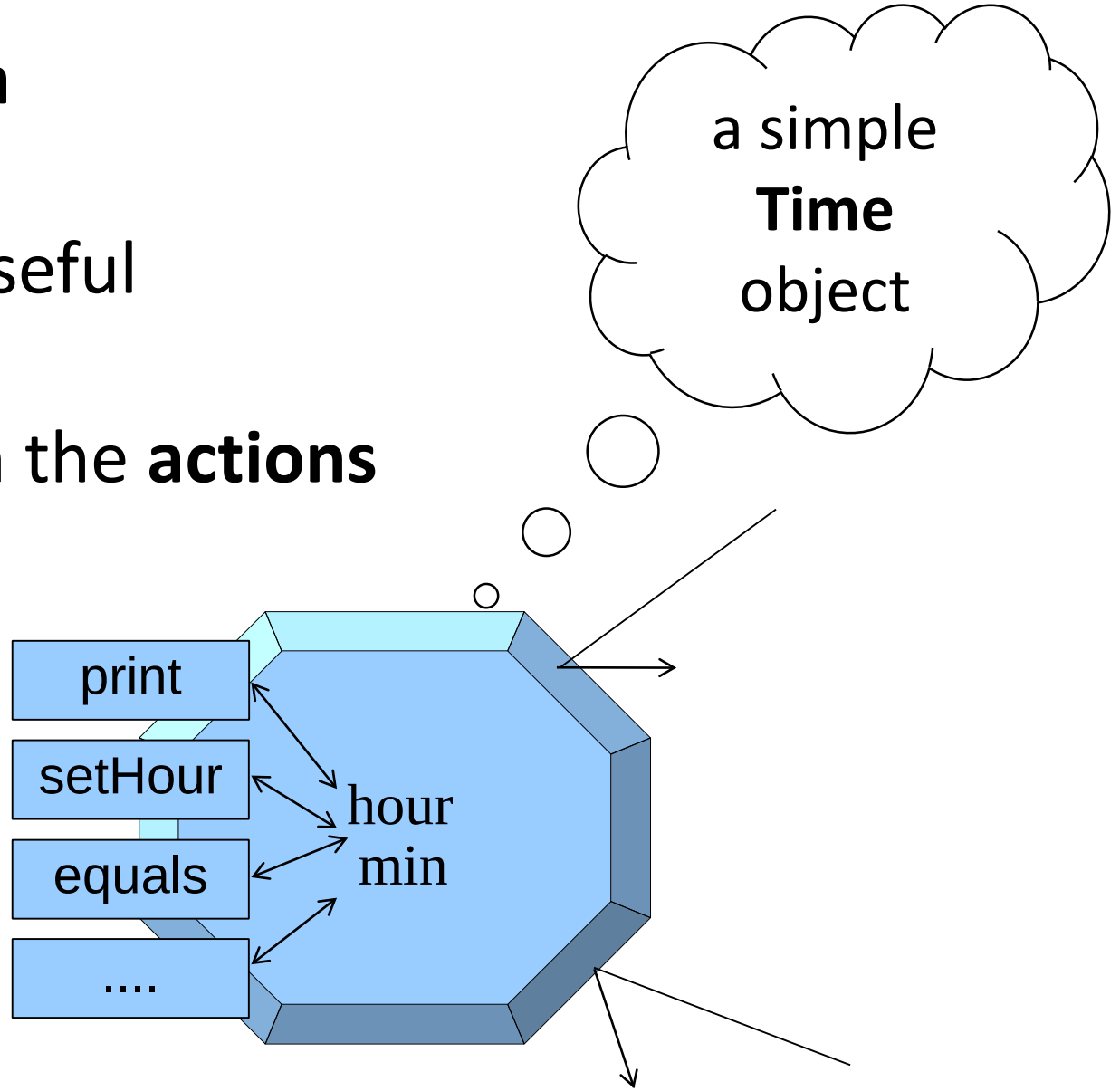
  or this could have been written as

- Programming using Objects

  So what are Objects.....?

An image I like for an object

- We can control access to the **data**
- Provide **actions** so the object is useful
- The **data** can be changed through the **actions**



Goals of an Object

- Protect the data
  - make sure it is valid (good)

- Example : Time object
  - control the access to the data

- Provide actions so the object is useful

Comparison

- Before OOP (Functional prog)
  - Changes to the data could be made through out the program
    - Hard to track down
- With OOP
  - Program is a collection of objects that work with each other

Comparison

- Before OOP
  - Changes to the data could be made through out the program
    - Hard to track down

  { for big projects }
  - Harder to write and test (for big projects)

- With OOP
  - Program is a collection of objects that work with each other

  - Easier to write and test
    - Write and test one object at a time

Comparison

- Before OOP
  - Changes to the data could be made through out the program
    - Hard to track down

  { for big projects }
  - Harder to write and test

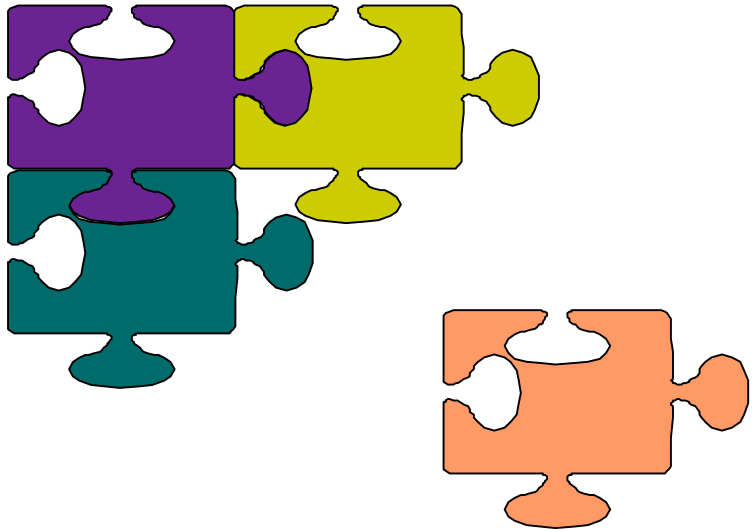
  - Harder to modify
    - Often easier to start over

- With OOP
  - Program is a collection of objects that work with each other

  - Easier to write and test
    - Write and test one object at a time

  - Easier to modify
    - The data and actions on the data are all in the same object

Comparison



- With OOP

– program is a collection of objects that work with each other

–Easier to write and test

- Write and test one object at a time

–Easier to modify

- can swap in and out different objects

(even at run time)

Something to think about

- *Object: logical combination of data and actions*
- What would a MP3 player object be like?

– Data:

– Actions:

Something to think about

- What would a MP3 player object be like?

– Data:
  - number of songs
  - current song
  - power/battery level
  - songs – could be objects

– Actions:
  - play
  - pause
  - rewind
  - forward
  - on/off

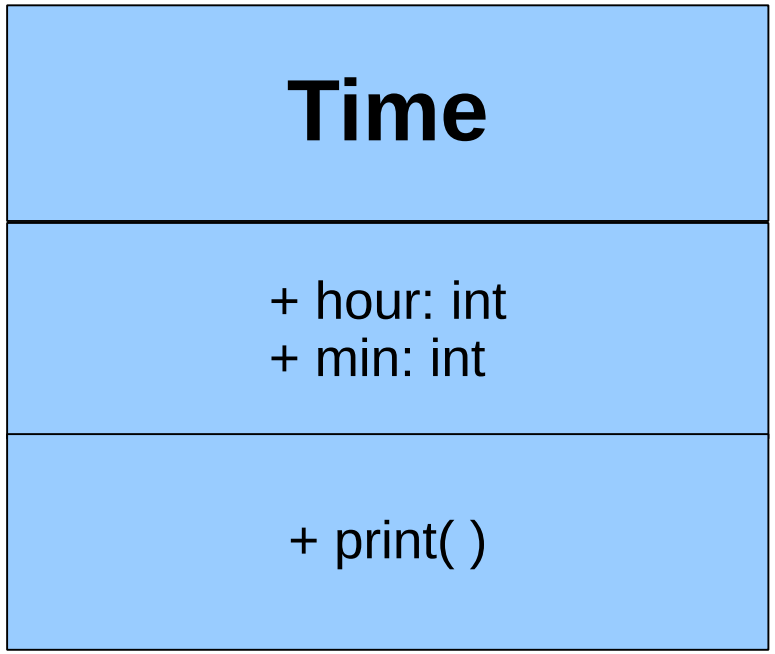
Example: Time Class

- Simple class to hold a time of the day
  - Display: HH:MM
  - Range: 00:00 – 23:59

- **Actions**:
  - **print:** show the time in HH:MM format
  - more actions will be added later....

UML: Unified Modeling Language

- A UML class diagram is often used show the information of a class in a language independent way.



Name of the class

Data for the class

Actions for the class

Here is a class definition

```
class Time {
  public:
    void print() {
       cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

Here are highlighted parts of a class definition

```
class Time {
   public:
      void print() {
         cout << hour << ":" << min;
      }

      int hour, min;
};  //  end of the Time class
```

- Keyword to start the definition of a class

Here are highlighted parts of a class definition

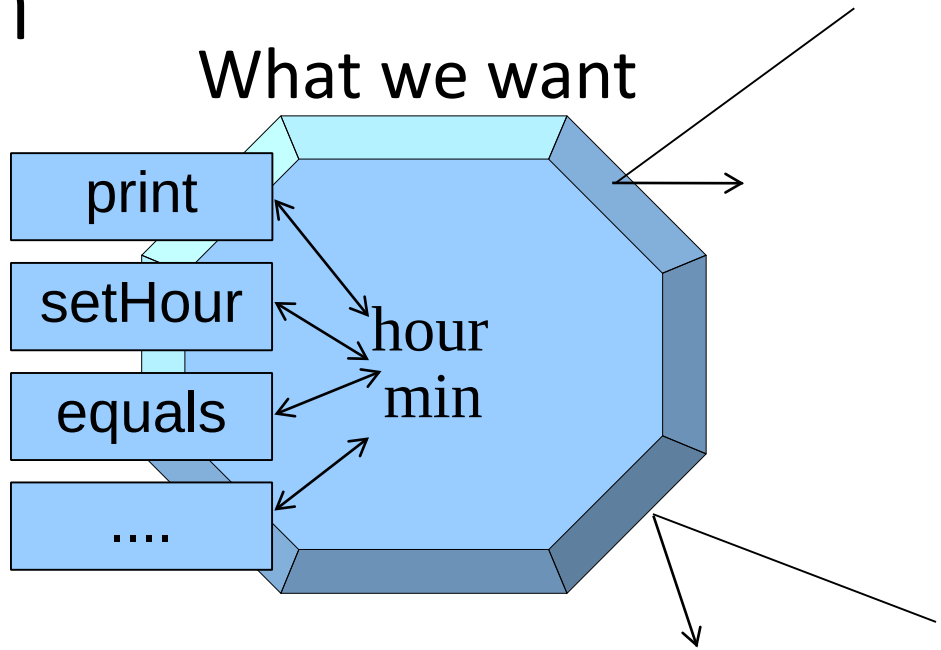
```
class Time {
  public:
    void print() {
        cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

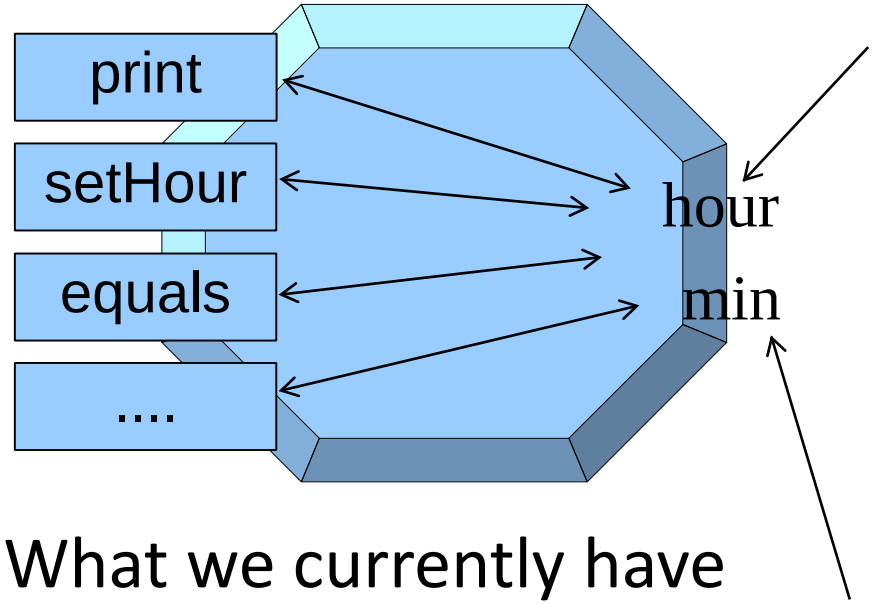
- Name of the new class (we will **always** start the name of a class with a capital letter)

Here is a class definition

```
class Time {
  public:
    void print() {
        cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

- Access specifier:
  - **public:** available *inside* and *outside* of the class
  - **private:** available only *inside* of the class





Here are highlighted parts of a class definition

```
class Time {
  public:
    void print() {
       cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

- Action for the class
  (also know as: a member function, a method or a procedure)

Here are highlighted parts of a class definition

```
class Time {
  public:
    void print() {
       cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

- Member data for the class

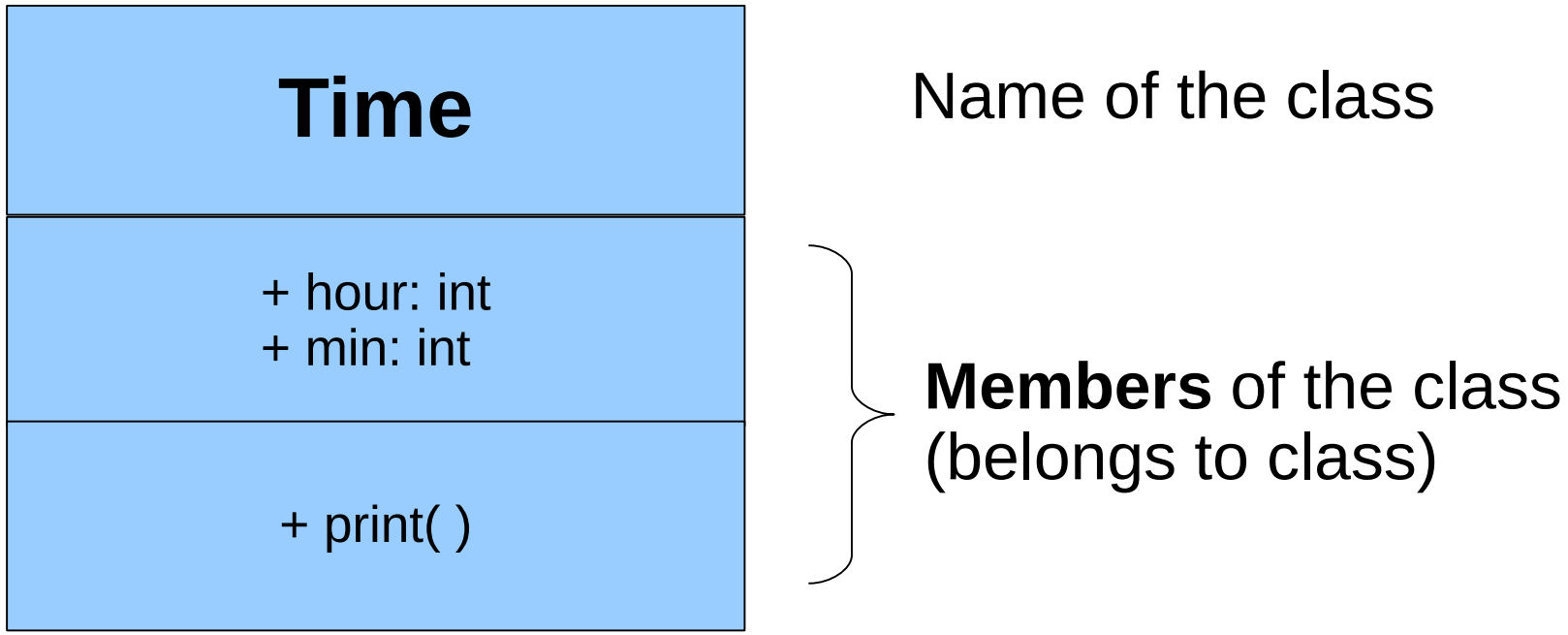
Here are highlighted parts of a class definition

```
class Time {
  public:
    void print() {
       cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

- A class definition ends with a ;
  - just like a structure

UML: Unified Modeling Language

- A UML class diagram is often used show the information of a class



Members

```
class Time {
  public:
    void print() {
       cout << hour << ":" << min;
    }

    int hour, min;
};  //  end of the Time class
```

Members of the class
(belongs to class)

- member functions
- member data

Creating and Using an instance

```
int main( ) {
   Time t, t2 = { 9, 50 };          // create two Time instances

   t.hour = 12;          // set the hour of t
   t.min = 30;
   t.print( );           // use the print action (function) of t
   cout << endl;

   t2.print( );           // use the print action (function) of t2
   cout << "\n\n";
   return 0;
}
```

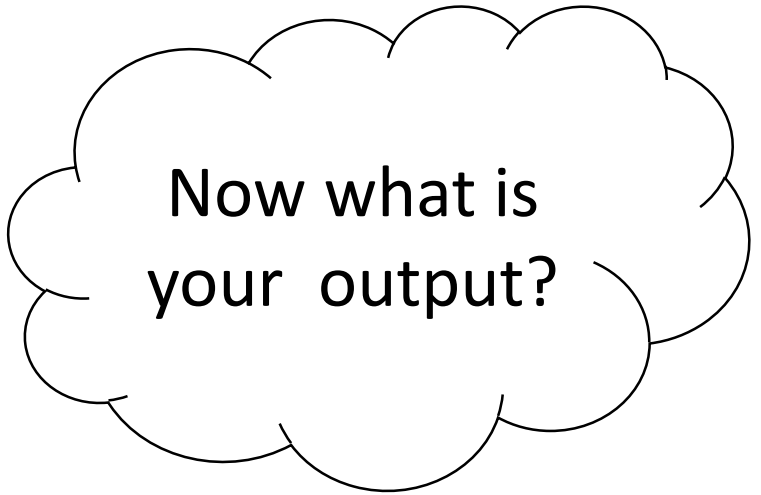
Class Activity: Enter & run the following

```
//  before main
class Time {
   public:
      void print() {
         cout << hour << ":" << min;
      }

      int hour, min;
};  //  end of the Time class
```

```
int main( ) {
   Time t, t2 = { 9, 50 };

   t.hour = 12;
   t.min = 30;
   t.print( );
   cout << endl;

   t2.print( );
   cout << "\n\n";
   system("pause");
}
```

Make the following changes

```
int main( ) {
    Time t, t2  = { 9, 50};

    t.hour = 12;
    t.min = 67;
    t.print( );
    cout << endl;

    t2.print( );
    cout << "\n\n";
    return 0;
}
```



Activity

Two Problems

```
int main( ) {
    Time t, t2;


    t.hour = 12;
    t.min = 67;          // can set a bad value
    t.print( );


    t2.print( );         // can print t2 before it has good info
    cout << "\n\n";
    return 0;
}
```

```
class Time {
 // public:
 void print(){
      cout << hour << ":" << min;
  }
  int hour, min;
};
```

Quick experiment

Let’s see what the default access for class members?

Comment out public and then run your code

Activity

Quick experiment

```
class Time {
 // public:
 void print(){
      cout << hour << ":" << min;
  }
  int hour, min;
};
```

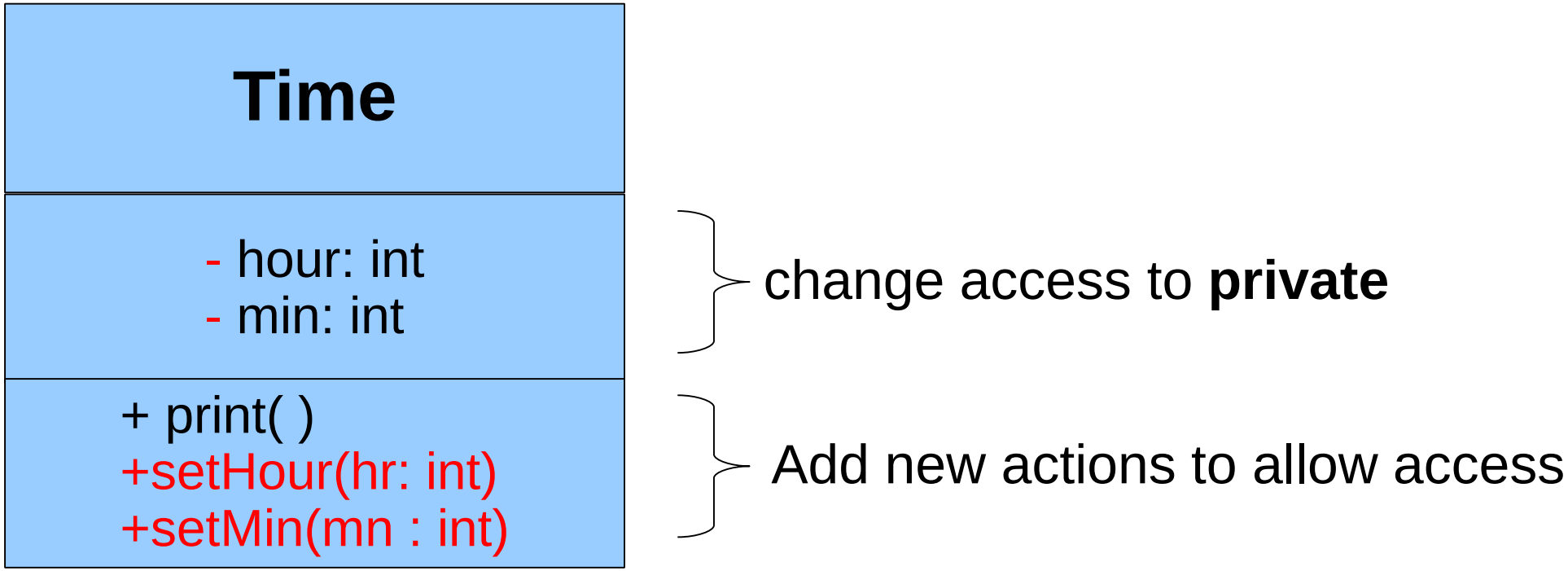
Let's see what the default access for class members?

Comment out public and then run your code

We get illegal access errors because the default access is **private**

Solving Problem #1

- We need to protect how the member variables are accessed



```
class Time {
  public:
   void print( ){
     cout << hour << ":" << min;
   }
   void setHour(int hr) {
      hour = 0;
      if (hr >= 0 && hr <= 23 ){
            hour = hr;
      }
   }// end of setHour
  private:
   int hour, min;
};
```

New version of the class definition

NOTE: setMin is very similar to setHour but checks for 0-59 and is not shown due to space

Add both accessor functions to your code

A few changes to use the instances

```
int main() {
  Time t, t2 = { 8, 15 } ;   // what happens here ???



  // t.hour = 12;     Can't access directly
  //                        hour is private
  t.setHour( 12 );      // set the hour of t
  t.setMin( 67 );
  t.print( );           // use the print function of t

  t2.print( );          // use the print function of t2
  cout << "\n\n";
  return 0;
}
```

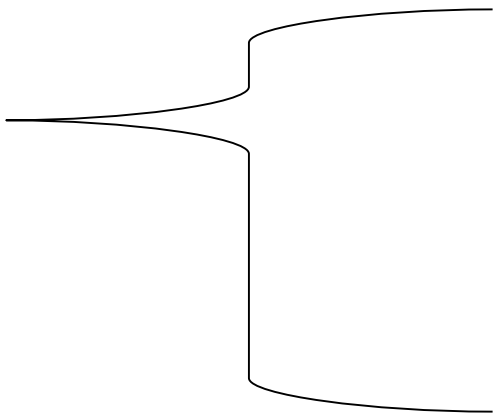
Activity

A few changes to use the instances

```
int main() {
  Time t, t2 ;   // = { 8, 15 } ;// can’t access due to being private



  // t.hour = 12;     Can't access directly
  //                          hour is private
  t.setHour( 12 );      // set the hour of t
  t.setMin( 67 );
  t.print( );           // use the print function of t

  t2.print( );          // use the print function of t2
  cout << "\n\n";
  return 0;
}
```

Calls the member function

```
int main() {
  Time t, t2;

  //  t.hour = 12;
  t.setHour( 12 );
  t.setMin( 67 );
  t.print( );

  t2.print( );
  cout << "\n\n";
  return 0;
}
```

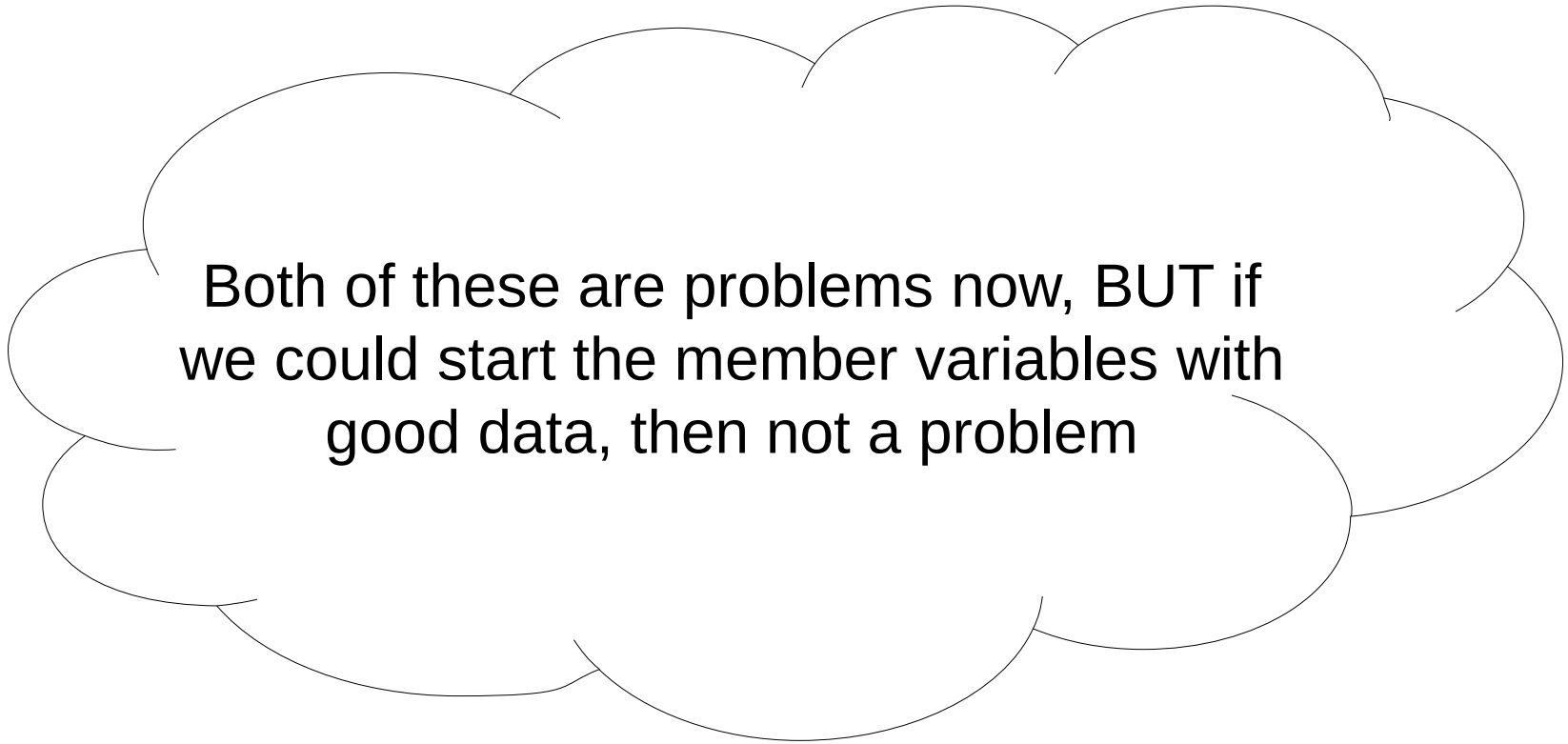


```
class Time {
  public:

    void setHour( int hr ) {
      hour = 0;
      if (hr >= 0  && hr <= 23 )
        hour = hr;
    }// end of setHour


  private:
    int hour, min;
};
```

12

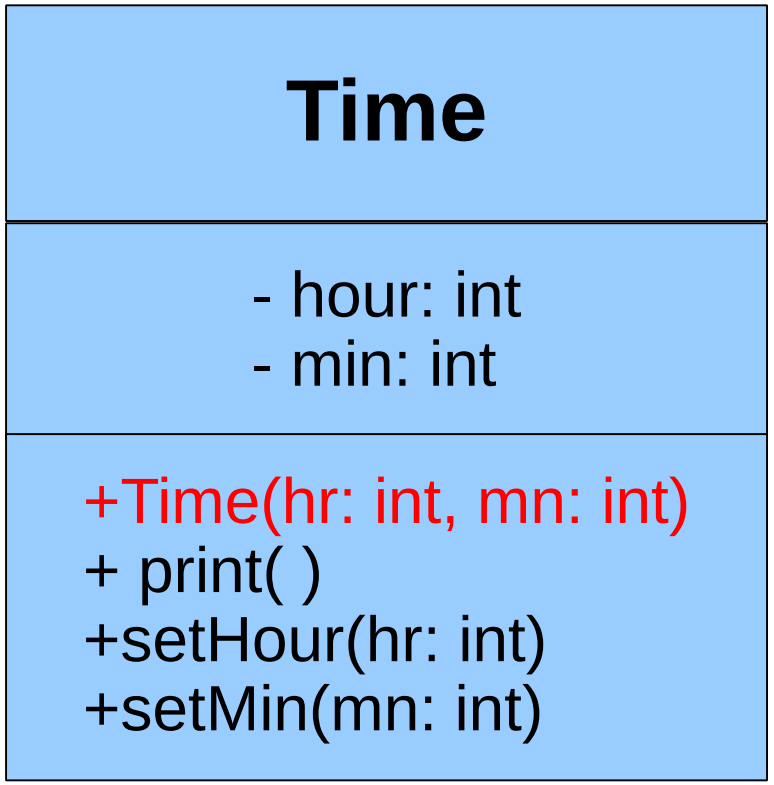
Using an instance

```
int main() {
  Time t, t2;


  t.setHour( 12 );
  t.setMin( 67 );
  t.print( );


  t2.print( );
  cout << "\n\n";
  return 0;
}
```



Solving Problem #2

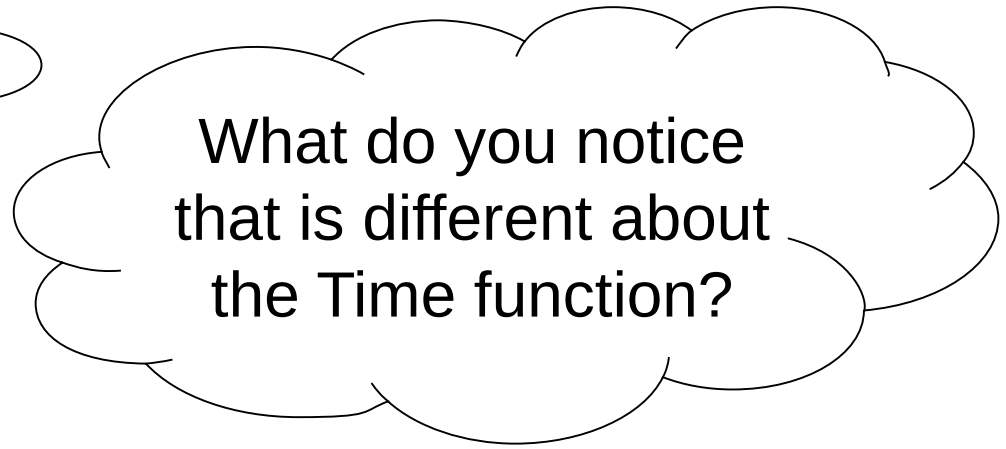
- We need to start the instance with “good” data



Add a new function. This is called the **constructor function**

Constructor Function

```
class Time {
  public:
    Time(int hr, int mn ) {
        hour = min = 0;
        setHour(hr);
        setMin(mn);
    }
    void print( ){
        cout << hour << ":" << min;
    }
    void setHour(int hr) {.....}
    void setMin(int mn) {.....}
  private:
    int hour, min;
};
```



Constructor Function

```
class Time {
  public:
    Time(int hr, int mn ) {
        hour = min = 0;
        setHour(hr);
        setMin(mn);
    }
    void print( ){
        cout << hour << ":" << min;
    }
    void setHour(int hr) {.....}
    void setMin(int mn) {.....}
  private:
    int hour, min;
};
```

What do you notice that is different about the Time function?

1. Same name as the class
2. No return type

Activity

Where do you think the Time function is being called (used)

```
int main() {
   Time  t( 12, 30 ) , t2(-1, 78) ;


   t.print( );
   cout << endl ;
   t2.print( );


   cout << endl << endl;
   return 0;
}
```

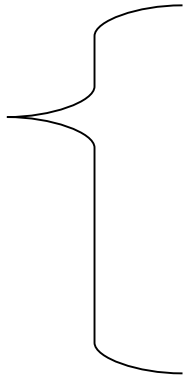
Activity

Where do you think the Time function is being called (used)

```
int main() {
    Time t( 12, 30 ), t2(-1, 78);

    t.print( );
    cout << endl ;
    t2.print( );

    cout << endl << endl;
    return 0;
}
```

Using the constructor

```
int main() {
  Time t(12, 30);
  Time t2(-1, 78);
  t.print( );
  cout << endl ;
  t2.print( );

  cout << endl << endl;
  return 0;
}
```



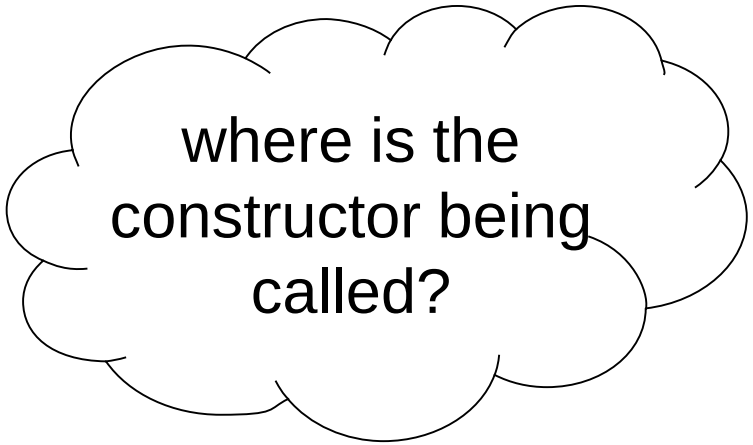
```
// in Time class definition
Time(int hr, int mn ){
    hour = min = 0;
    setHour(hr);
    setMin(mn);
}

void setHour(int hr) {
      if (hr >= 0 && hr <= 23 ){
          hour = hr;
       }
}   // end of setHour
```

Constructor Functions

- Special features of Constructor Function: (**understand** and **memorize**)
  - same names as the class
  - no return type
  - called when a instance is created
  - every class has one…*if the programmer doesn't create one*, then one is given by the compiler
    - the "free" constructor allows the creation of instances but does nothing else !!!
    - the "free" constructor takes no parameters

Old Example of Creating and Using an instance

```
int main( ) {
    Time t, t2;

    t.hour = 12;
    t.min = 30;
    t.print( );

    t2.print( );
    cout << "\n\n";
    return 0;
}
```



Old Example of Creating and Using an instance

```
int main( ) {
   Time t, t2;              // uses the “free” constructor

   t.hour = 12;
   t.min = 30;
   t.print( );

   t2.print( );
   cout << "\n\n";
   return 0;
}
```

Using the constructor function

```
int main() {
   Time  t(12, 30) , t2(-1, 78);
   Time  t3;

   t.print( );
   cout << endl ;
   t2.print( );

   cout << endl << endl;
   return 0;
}
```

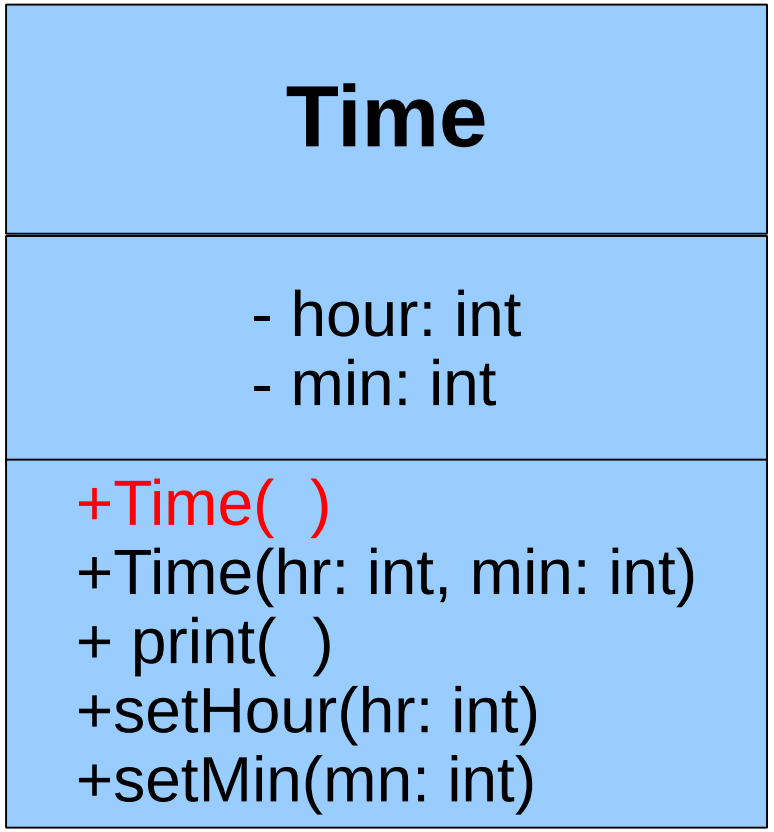
```
// in Time class
Time(int hr, int mn ){
   hour = min = 0;
   setHour(hr);
   setMin(mn);
}
```

NOTE:

- variable t3 would have used the "free" constructor, but since we created a constructor, we no longer get the free one

Default Constructor

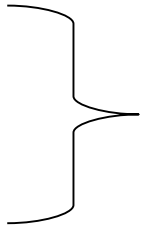
- **Default Constructor** is a constructor that does not have any parameters (can be called without any arguments)



Adding the default constructor function

Default Constructor

```
class Time {
  public:
    Time(int hr, int mn ) {
      hour = min =0;
      setHour(hr);
      setMin(mn);
    }
    Time(  ) {
      hour = min = 0;
    }
   void setHour(int hr) {..... }
 private:
   int hour, min;
};
```



The default constructor:
sets our member variables
to zero

Using the constructor functions

```
int main() {
   Time  t(12, 30);                    // uses 2 parameter constructor
   Time  t2(-1, 78);                   // uses 2 parameter constructor

   Time  t3;                           // uses default constructor

   t.print( );
   cout << endl ;
   t2.print( );

   cout << endl << endl;
   return 0;
}
```

Using the constructor functions

```
int main() {
   Time  t(12, 30);
   Time  t2(-1, 78);

   Time  t3;

   t.print( );
   cout << endl ;
   t2.print( );

   cout << endl << endl;
   return 0;
}
```

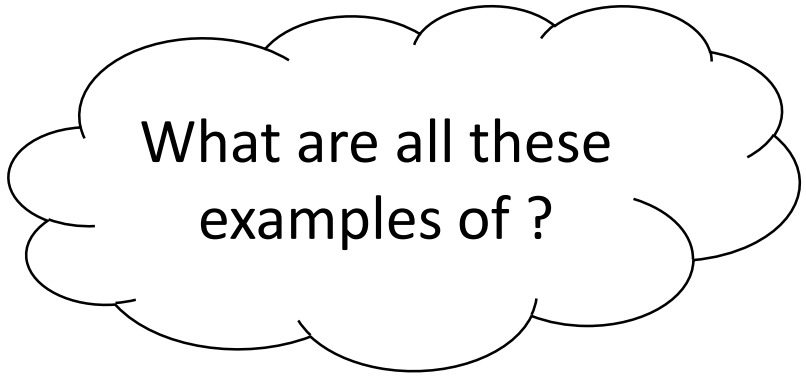
```
class Time {
  public:
    Time(int hr, int mn ) {
     cout << "2 param constructor\n";
      hour = min =0;
      setHour(hr);
      setMin(mn);
    }
    Time(  ) {
      cout << "default constructor\n";
      hour = min = 0;
    }
   // other stuff
```

Activity

Connecting some previous knowledge

- When we created a vector I gave you this example:

```
vector<int> v1 ( 4 );                    // v1: 0, 0, 0, 0
vector<double> v2(3, 5.4);               // v2: 5.4, 5.4, 5.4
```

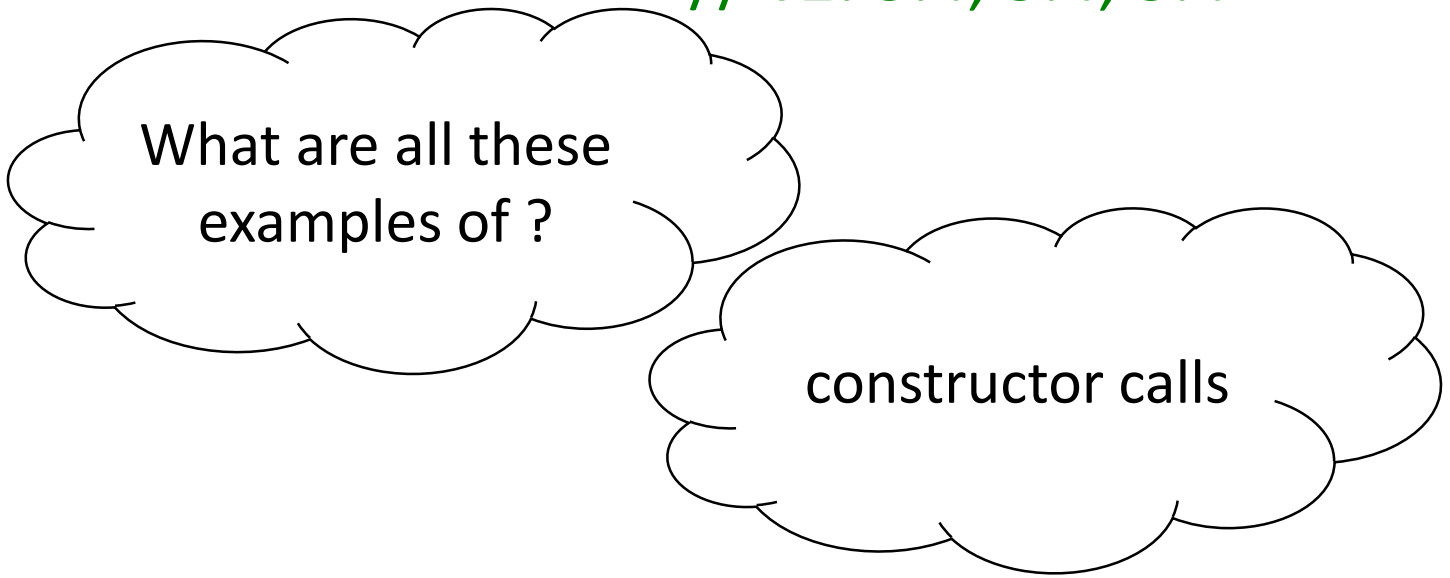



```
Time  t(12, 30);
Time  t2(-1, 78);
Time  t3;
```

Connecting some previous knowledge

- When we created a vector I gave you this example:

```
vector<int> v1 ( 4 );                          // v1: 0, 0, 0, 0
vector<double> v2(3, 5.4);                     // v2: 5.4, 5.4, 5.4
```



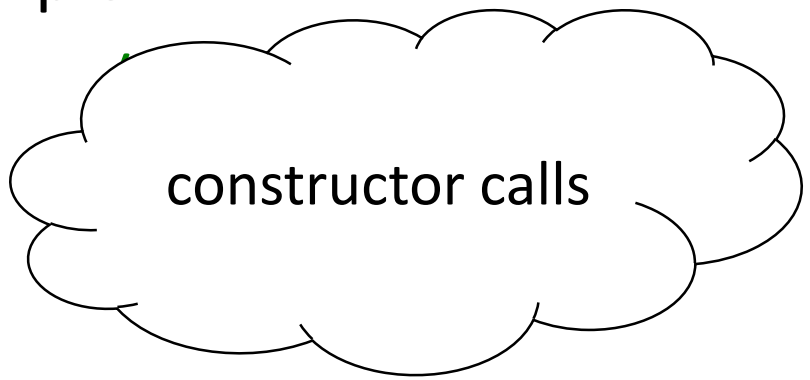

```
Time  t(12, 30);
Time  t2(-1, 78);
Time  t3;
```

```
vector< T > vectName( int size, T initVal = 0);        // prototype
```

Connecting some previous knowledge

- When we created a vector I gave you this example:

```
vector<int> v1 ( 4 );
vector<double> v2(3, 5.4);
```




```
Time  t(12, 30);
Time  t3;
```

Also I gave this as an example of the prototype for creating a vector

```
vector< T > vectName( int size, T initVal = 0);          // simplified prototype
```

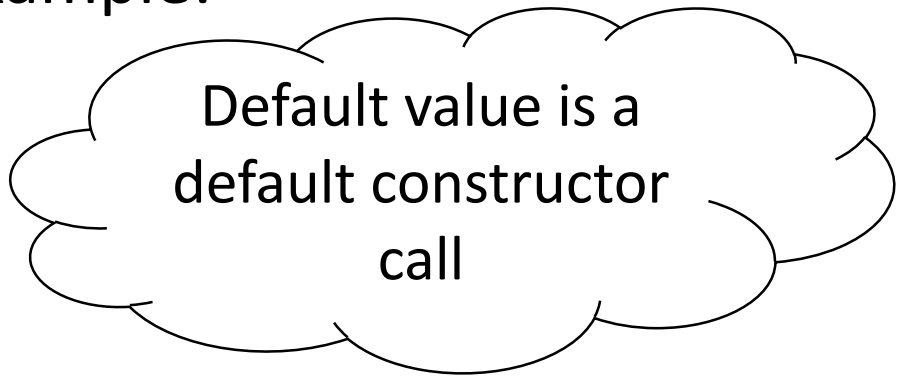
but here is how it really is:

```
vector< T > vectName( int size, T initVal =  T( )  );    // real prototype
```

Connecting some previous knowledge

- When we created a vector I gave you this example:

```
vector<int> v1 ( 4 );
vector<double> v2(3, 5.4);
```



Also I gave this as an example of the prototype for creating a vector

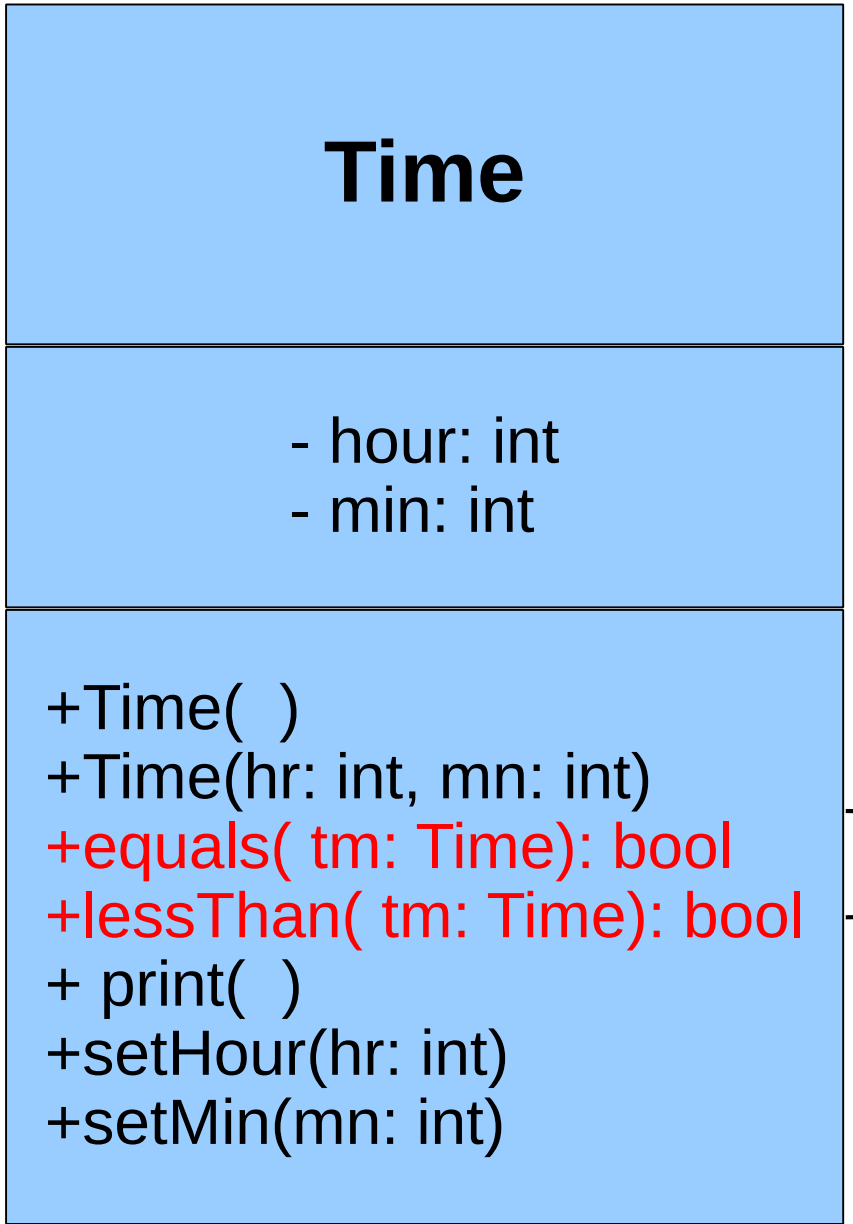
```
vector< T > vectName( int size, T initVal = 0);          // simplified prototype
```

but here is how it really is:

```
vector< T > vectName( int size, T initVal =  T( )  );    // real prototype
```

```
vector<Time> timeVect( 5 );  // creates 5 Time instances of 00:00
```

Adding comparison functions

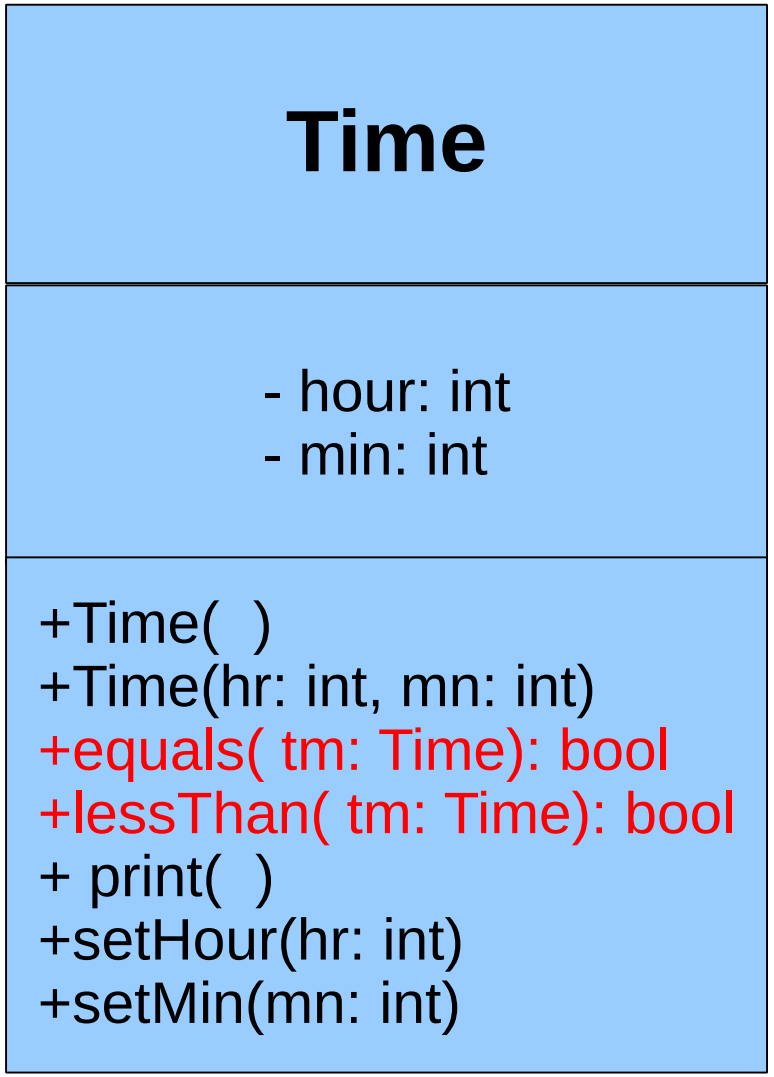


Adding a couple functions to allow us to compare Time instances

Using the *equals* function

```
// in main
cout << "Testing Time Class" << endl;
cout << endl << boolalpha;

t.print( );
cout << " == ";
t2.print( );
cout << " : " << t.equals( t2 ) << endl;

t2.print( );
cout << " == ";
t3.print( );
cout << " : " << t2.equals( t3 ) << endl;
cout << endl << endl;
```

Adding comparison functions

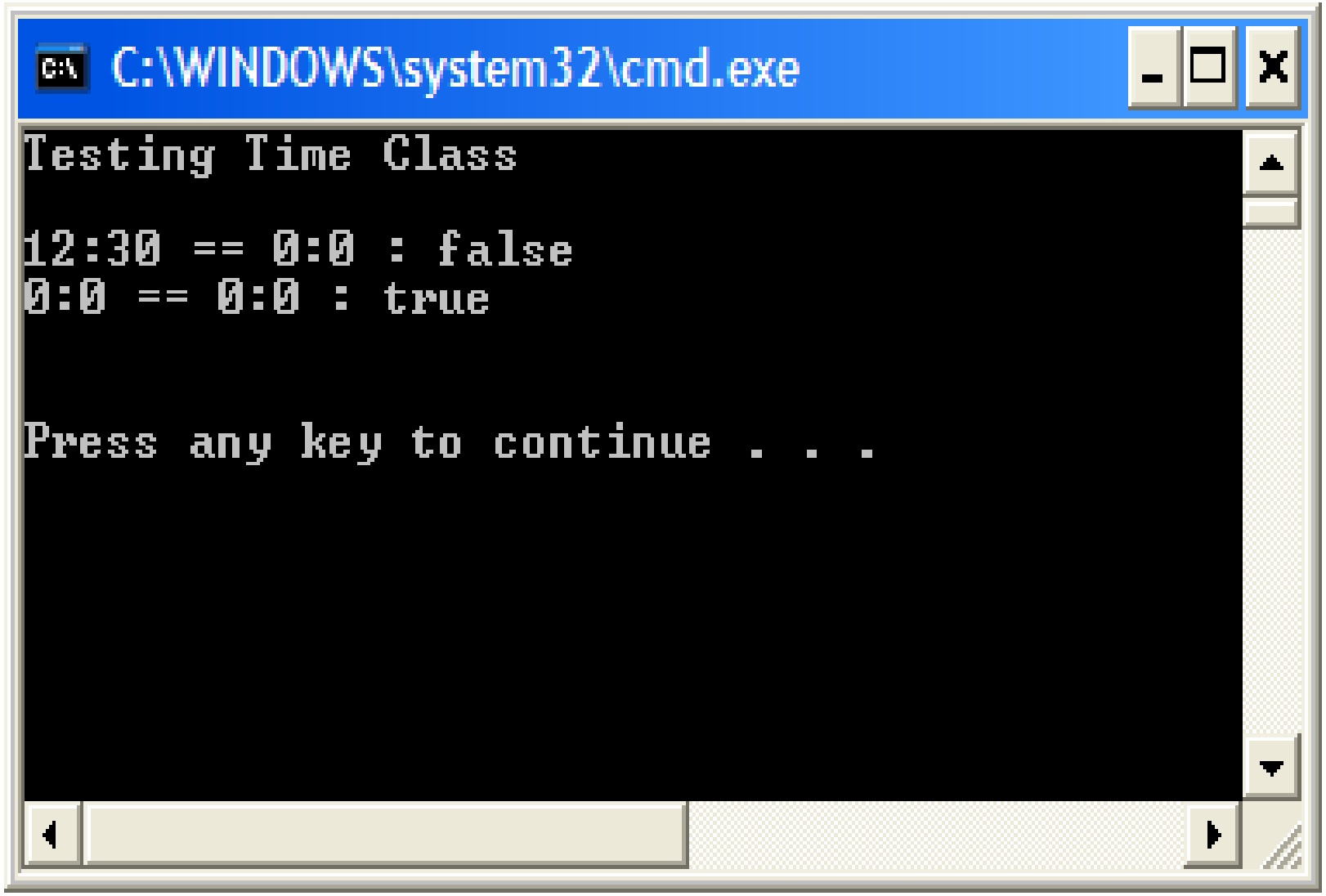



```
class Time {
 public:
   Time(int hr, int mn ){.....}

    bool equals( Time tm ) {
        bool ans = false;
        if (tm.hour == hour ) {
            if ( tm.min == min )
                ans = true;
      }
      return ans;
    }  // end of equals

// rest of Time definition
```

Using the *equals* function

```
// in main
cout << "Testing Time Class" << endl;
cout << endl << boolalpha;


t.print( );
cout << " == ";
t2.print( );
cout << " : " << t.equals( t2 ) << endl;


t2.print( );
cout << " == ";
t3.print( );
cout << " : " << t2.equals( t3 ) << endl;
cout << endl << endl;
```

Example of the output

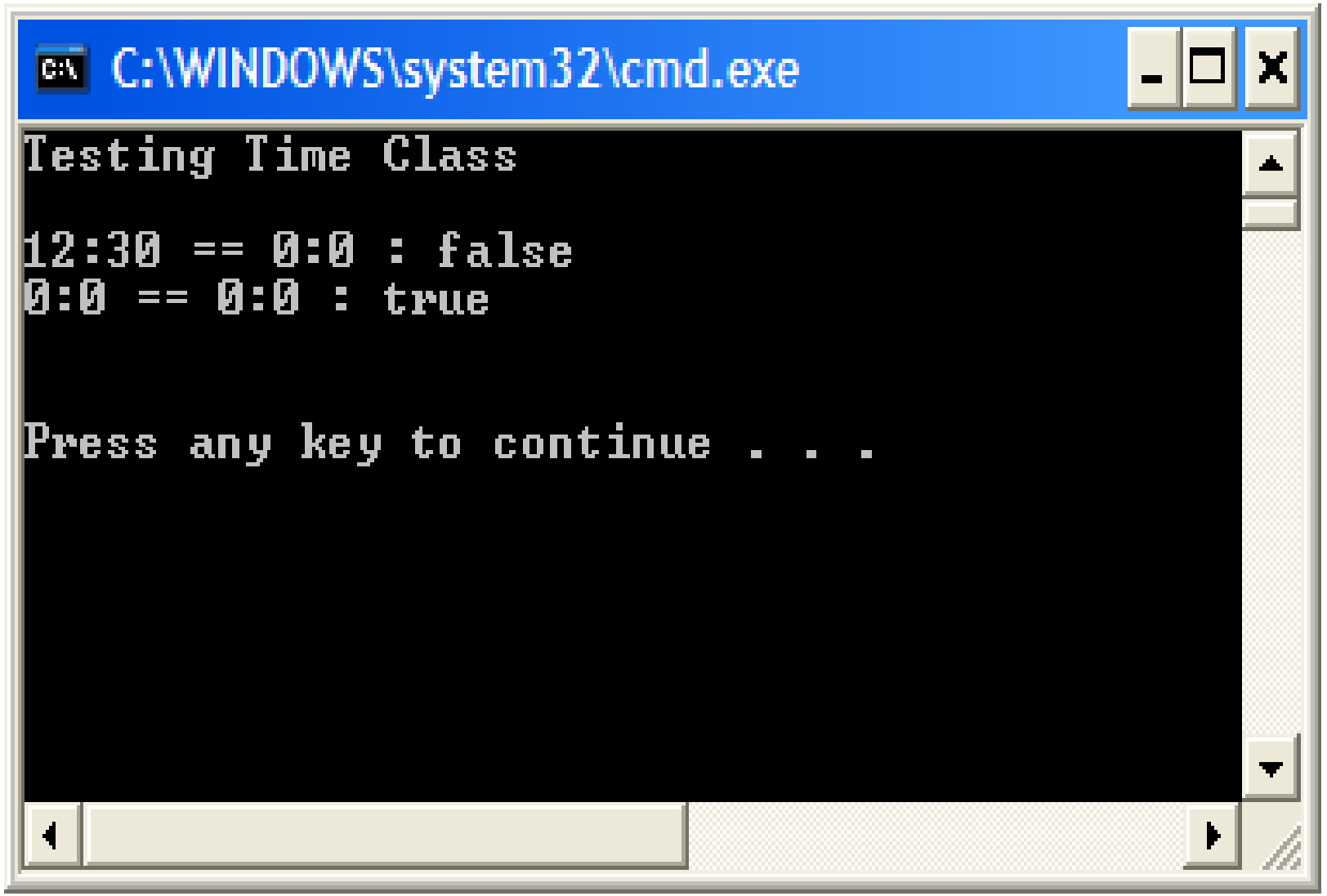


Using the *equals* function

```
// in main
cout << "Testing Time Class" << endl;
cout << endl << boolalpha;


t.print( );
cout << " == ";
t2.print( );
cout << " : " << t.equals( t2 ) << endl;
```

```
// above main
class Time {
  public:
    Time(int hr, int mn ){ .....  }

    bool equals( Time tm ) {
        bool ans = false;
        if (tm.hour == hour ) {
            if ( tm.min == min )
                ans = true;
        }
        return ans;
    }  // end of equals
```

Hmmm, the output could be better

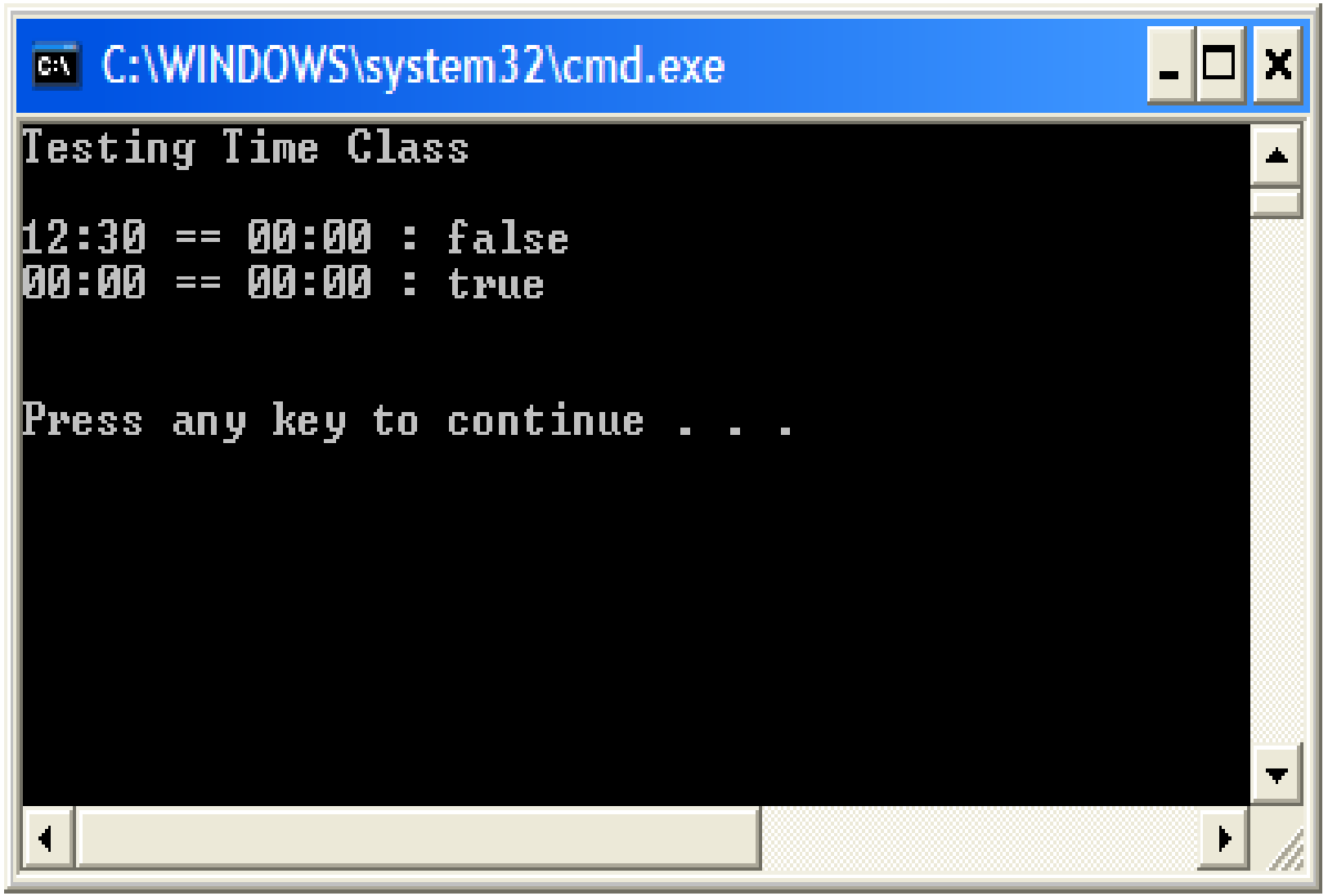


A simple change to print( )

```
class Time {
  public:
 ......
   // Display time in HH:MM format
   void print( ) {
      if( hour < 10 )
         cout << "0";
      cout << hour << ":";
      if (min < 10 )
         cout << "0";
      cout << min;
   }
```

Activity

Ahhh, that is much better



Adding and testing comparison functions

```
// in main
cout << "Testing Time Class" << endl;
cout << endl << boolalpha;


t.print( );
cout << " == ";
t2.print( );
cout << " : " << t.equals( t2 ) << endl;


t2.print( );
cout << " == ";
t3.print( );
cout << " : " << t2.equals( t3 ) << endl;
cout << endl << endl;
```

```
class Time {
 public:
   Time(int hr, int mn ){.....}

   bool equals( Time tm ) {
      bool ans = false;
      if (tm.hour == hour ) {
          if ( tm.min == min )
              ans = true;
      }
     return ans;
   }  // end of equals

// rest of Time definition
```

Activity

The whole program

```
#include <iostream>
using namespace std;

class Time {
    Time(int hr, int mn )  {.....}
  private:
    int hour, min;
};     // end of Time class

int main() {
   Time t(12, 30) , t2(-1, 67);
   t.print();
   cout << " == ";
   t2.print();
   cout << " : " << t.equals( t2 ) << endl;
   return 0;
}     // end of main
```

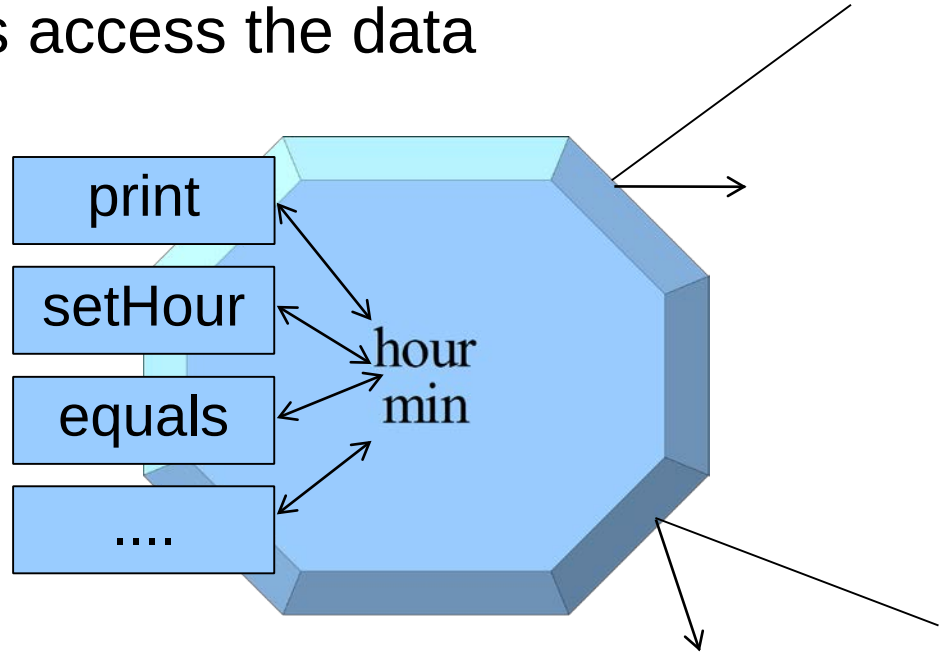
Time Class

main program uses Time

Lets take a rest and review

- We can control access to the data

–The member functions
  - are public (accessible)
  - are used to give outsiders access the data



Topics Summary

- Classes
  - creating and using
  - public, private access modifiers
  - member functions
  - constructor functions and how they are different from other member functions
  - can access private data of other instances from the same class (example: equals function)