Decomposition
Abstraction
Class Syntax

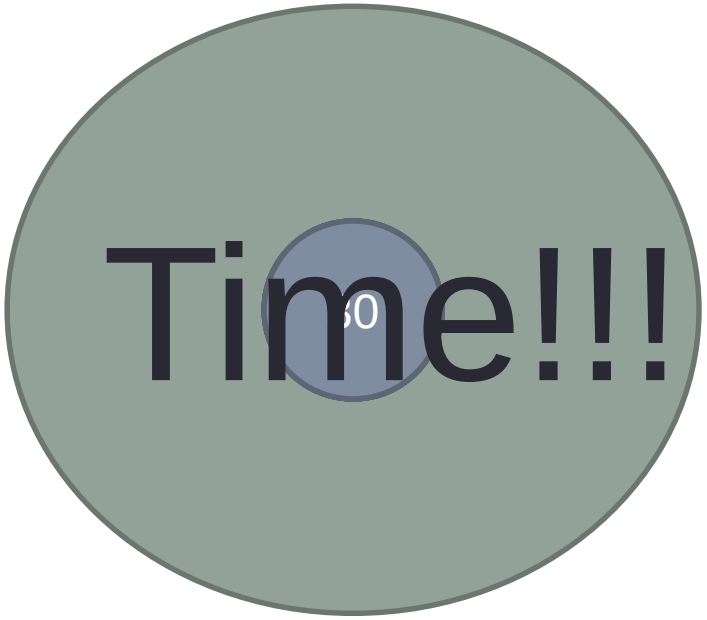
Luke Sathrum – CSCI 20

Today's Class

- Decomposition
- Abstraction
- Class Syntax
- Member Functions

Figure It Out

- Sum all the numbers between 1 & 200
  - Must do it in your head
  - And you get 30 seconds
- Go!!!



Decomposition

- The process of breaking a problem down into smaller pieces
- How can we break down the “Sum To” problem?
  - What if we just add the ends together?
  - 200 + 1 = ???
  - 199 + 2 = ???
  - 198 + 3 = ???
  - Pattern?
  - How many pairs?
    - Last one is 101 + 100

Abstraction

- Pulling out specific differences to make one solution work for multiple problems
- What if we wanted to work with any number?
- Remember
  - SumTo: 200
    - 100 * 201
  - SumTo: 10
    - ?????
  - SumTo: 2000
    - ?????
  - SumTo: n
    - ?????

In Code

- How would we write our problem in code?
- What if we wanted a different number?
- What if we wanted to run the algorithm on 2 different numbers at the same time?

Another Problem

- Let's figure out an algorithm to calculate exponents
  - i.e. 2^5, 3^{10}
- Go ahead and work on it with a partner or on your own
- What does this look like in code?
- Instead of copying and pasting can we abstract?
- We will do this abstraction via functions
- Something like
  - `SumTo(200)`
  - `RaiseToPower(2, 5)`

Why Define Functions?

- Readability
  - **`RaiseToPower()`** is easier to read than a bunch of copy and pasted code
- Maintainability
  - To change/fix an algorithm you just need to change it in one place
- Code Reuse
  - You (and others) can now use the function as much as you want

Class Syntax

Introduction to Class Syntax

- Class syntax is very similar to structure syntax
- Classes can have member variables
  - These are the attributes of the class
- In addition classes can have member functions
  - These are the behaviors of our class
- Don’t Forget
  - Classes are the blueprints for our objects
  - Each object is an instance of a class

Class Syntax

- Let's make a class to interact with our functions

```
class Math {
 public:
  void OutputCurrentValue();
  int value_;
  int extra_input_;
};
```

Class Syntax Notes

```
class Math {
 public:
  void OutputCurrentValue();
  int value_;
  int extra_input_;
};
```

- Class syntax ends with a semicolon
- The first letter of the class is uppercase
- Everything after **`public`** will be accessible to the outside
- **`OutputCurrentValue()`** is a member function declaration
- **`value_`** and **`extra_input_`** are member names

Accessing Class Members

- We use the dot operator to access
  - Member Variables
  - Member Functions
- Remember
  - An object is an instance of a class (a class variable)
- To create an object

```
Math math_object;
```

- To access class members

```
math_object.Output();
math_object.value_ = 5;
```

Summary

- Class syntax is similar to structure syntax
- The contain
  - Member Names
    - The Attributes
  - Member Function Declarations
    - The Behaviors
- We access members with the dot operator

Sample Code

- Our first Class
  - `class.cpp`

Member Functions

Member Functions

- Member Functions are the behaviors or our classes
- They can do lots of things
  - Obtain Input
  - Calculate Data
  - Display Data
- Each of these can be put into their own function
- These functions usually manipulate our attributes

Member Functions May

- Return a Value
  - Produce a value
- Perform an action without producing a value
  - Called a **`void`** member function

Member Functions

- Member functions belong to a particular class
- We need some way to relate the function back to its class
- This allows us to use the same function name for different classes
  - Makes sure we don’t overlap function names from class to class
- We will assume we still have our `Math` class

Member Function Syntax

```
void Math::OutputCurrentValue() {
  // Code goes here
}
```

- The first word is the return type
- Next we add the name of our class
- After that comes the two colons
  - `::` is called the Scope Resolution Operator
  - Similar to the dot operator
  - Only used with class definition
    - Use dot for class instance (object)
- Class name is called the type qualifier
- Then comes the function name
- And after that parentheses

Member Function Syntax

```
void Math::OutputCurrentValue() {
   // Code goes here
}
```

Member Function Syntax

- Member Functions are allowed to access other Members
  - Both variables and functions
- No need to re-declare member variables

Member Function Syntax

- Assume `value_` and `extra_input_` are both member names
- We can access from our member function

```
void DayOfYear::OutputCurrentValue() {
  cout << value_ << endl;
}
```

Summary

- Member functions are the behaviors of our objects
- They can return a value
  - Or not, called `void` member functions
- The syntax order is
  - Return type
  - Class Name
  - Scope Resolution Operator
  - Member Function Name
  - Parentheses

Sample Code

- Our first member function
  - `member_functions.cpp`

More Member Functions

Sample Code

- Let's write both **`SumTo()`** and **`RaiseToPower()`**
  - `more_member_functions.cpp`

Review

- CinReader
- Code Style
- Boolean Expressions