Riddles

- What can run but never walks, has a mouth but never talks, has a head but never weeps, has a bed but never sleeps?
- You use a knife to slice my head and weep beside me when I am dead.
- I am weightless, but you can see me. Put me in a bucket, and I'll make it lighter.
- I'm light as a feather, yet the strongest man can't hold me for much more than a minute.
- I'm where yesterday follows today, and tomorrow's in the middle.

Review

- We get input into functions via?
- We get data out of functions via?
- Member Variables should be?
- We get data into member variables via?
- We get data out of member variables via?

```
class DayOfYear {
 public:
  void Output();
  int month();
  int day();
  void set_month(int month);
  void set_day(int day);
 private:
  int month_;
  int day_;
};
```

Constructors
Default Arguments

Luke Sathrum – CSCI 20

Today's Class

- Constructors
- Overloaded Constructors
- Default Arguments

Introduction to Constructors

Constructors

- It would be useful to initialize member variables when we create an object
  - Allows us to make sure our object has data that makes sense for it
- We have a special member function for this initialization
  - Called a Constructor
- It is used to
  - Initialize values of member variables
  - Do any other sort of initialization that may be needed

Constructors

- Defined the same way as other member functions except
  - A constructor MUST have the same name as the class
    - If your class name is `DayOfYear` then the name of the constructor is also **`DayOfYear`**
  - A constructor definition CANNOT return a value
    - Cannot use the **`void`** keyword either
- Think of a constructor as a function that only runs when you create your object

Constructor Syntax

```
class DayOfYear {
 public:
  //Constructor
  DayOfYear(int month, int day);
  //Rest of definition
};
```

- Notes
  - Name is same as the class
  - Does not include a return type
  - Place in the public section of your definition

Constructor Syntax - Definition

```
DayOfYear::DayOfYear(int month, int day) {
   month_ = month;
   day_ = day;
}
```

- Note
  - The first **DayOfYear** is the class name
  - The second **DayOfYear** is the constructor name
  - No return type is given

Constructor – How to Use

- To use the constructor for **DayOfYear**

```
DayOfYear date1(7, 4), date2(5, 5);
```

- How many objects created?
  - The created objects have also been
- You can think of it as being equivalent to the following

```
DayOfYear date1, date2;
date1.DayOfYear(7, 4);
date2.DayOfYear(5, 5);
```

Summary

- Constructors are called on Object creation
- Allow us to initialize our attributes
- Are named the same as the Class
- Written like a member function with no return type

Sample Code

- Creating Constructors
  - `constructor.cpp`

Overloaded Constructors

Overloading

- C++ allows you to give multiple definitions to the same function name
- This means you can reuse the name of a function for a function that is similar
- Called overloading

Overloading a Constructor

- Allows your object to be initialized in more than one way
- For **`DayOfYear`** we might want to allow the user to initialize both parameters, one parameter, or no parameters
- Allows us to initialize when the user doesn't specify any values

Constructors with No Parameters

- Perfectly legal to have a constructor with no parameters
- Called a Default Constructor
- This constructor can be either automatic or manual
- To call the default constructor you **DO NOT** use parentheses
- Syntax

```
//Calls the default constructor for both day1 and day2
DayOfYear day1, day2;
```

Automatic vs. Manual Constructors

- If you do not define any constructors then a default constructor is automatically created for you
  - This will create an uninitialized object
- If you create a constructor that has any number or parameters then a default constructor is not created
  - You will have to define it yourself
- Because of this problem we choose to always define the default constructor in our class

Example

- If I've created the following constructor
  - `DayOfYear(int month, int day);`
- I can do the following
  - `DayOfYear today(4, 29);`
- I cannot do the following
  - `DayOfYear today`
- Unless I've created the default constructor
  - `DayOfYear();`

Sample Code

- Initializing Constructors and Overloading
  - `constructor_overload.cpp`

Default Arguments

Default Arguments

- You can specify default arguments for one or more parameters
- If the corresponding argument is omitted then the function uses the default argument
- Example
- ```
  void SetVolume(int length, int width = 1, int height = 1);
  ```
  ```

Default Arguments - Notes

- You only declare the default argument value once
  - In the member function declaration (Inside the class)
- You may have more than one default argument
- Default arguments must start in the rightmost positions
- As you leave off arguments in the function call you do from right to left

Default Arguments - Notes

- Given

```
void SetVolume(int length, int width = 1, int height = 1);
```

- You can use the following calls

```
object.SetVolume(10, 5, 2);
object.SetVolume(10, 5);
object.SetVolume(10);
```

Sample Code

- Using default arguments
  - `default_arguments.cpp`

Summary

- _____ are called when an object is created
- Are named after the name of the class
- We often _____ constructors to allow for different initializations
- Default Arguments are used as placeholder when a user doesn't provide enough arguments