Is it a word?

- biduous
- castrealm
- luezoid
- chirotony
- darg
- chesture
- educand
- feandra

- Yes
- No
- No
- Yes
- Yes
- No
- Yes
- No

Parameters and Return Types
Encapsulation
Accessors and Mutators

Luke Sathrum – CSCI 20

Today's Class

- Parameters
- Return Types
- Encapsulation
- Accessor Member Functions
- Mutator Member Functions

Today's Class

```
class DayOfYear {
 public:
   void Output();
   int month_;
   int day_;
};
```

Parameters and Return Types

Parameters

- Functions can take input
- They do this through their parameters
- These are defined inside the parentheses
- Member Function Declaration

```
void Input(int month, int day);
```

- The variables (called parameters) can be used inside the member function

Parameters - How to use

- We access members via the dot operator
- To use parameters we pass in arguments
  - Can be variables, literals, or expressions
- Example

```
DayOfYear today;
today.Input(10, 5);
```

Parameters - How to use

In **main()**

```
DayOfYear today;
today.Input(10, 5);
```

Below **main()**

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

Return Types

- Member Functions can
  - Return a value
  - Perform an action but not return a value
    - Called **`void`** member functions
- The first word of our member function definition is our return type
  - Can be any of our types
    - **`int`**
    - **`double`**
    - etc.

Return Types

- Inside our functions we use the keyword **`return`** to return a value from a member function
  - It must match the return type specified
  - We have already seen the keyword before
- Example
  - Assuming our function's return type is **`char`**…
  - `return 'Y';`

Sample Code

- Parameters and Return Types
  - `parameters_return.cpp`

Encapsulation

Encapsulation

- A programmer doesn’t need to know
  - how your class is implemented
  - the details of your functions or data
- Just need to know how to use/interface with your class

Encapsulation Example

- Let's say `DayOfYear` class has a function named **`OutputMonth()`**
- Do we need to know how the month is stored in the object?
  - Is it an integer?
    - Range 0 -11?
    - Range 1 - 12?
  - Is it a string?
    - Name of month?
- The answer is NO
- All we need to know is the name of the function

Encapsulation in C++

- We have two keywords to implement Encapsulation
  - `public`
    - Data and functions are accessible to the outside world
  - `private`
    - Data and functions can only be accessed from inside the object

Encapsulation in C++ - Example

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

Encapsulation in C++ - Example

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

- Private member names are not accessible from outside
- We create our object the same `DayOfYear today;`
- The following would be illegal

```
today.month_ = 2;
```

Encapsulation Notes

- Anything following a keyword will belong to that keyword
- You can have multiple `private:` and `public:` sections
  - This is bad practice but does work
- If you don’t specify a keyword then everything will default to `private`

Encapsulation Notes

- It is good practice to make all member names **`private`**
  - Remember these are your attributes
  - This allows you to protect them from unknown/bad data
- Most (if not all) member function definitions will be **`public`**

Summary

- The outside world doesn’t need to know the details
- Just need to know how to use
- We show and hide details via the keywords
  - `public`
  - `private`

Sample Code

- Encapsulation in Classes
  - `encapsulation.cpp`

Accessor and Mutator Member Functions

Introduction

- Good practice to make member variables private
- Your class would be useless if you couldn’t access those variables
- We write member functions in order to access our private member variables
- Called
  - Accessor Functions
    - Allow us to ‘Access’ our data
  - Mutator Functions
    - Allow us to change (‘Mutate’) our data

Accessor Functions

- Allow us to read the data
- Accessor functions are named for their member name
- So if we has a member name

```
int month_;
```

- Our accessor would be
  - `int month();`

Mutator Functions

- Allow you to change the member data
- A great place to do some error checking on the input
  - Remember, input is via Parameters!
- Allows you to control and filter the data that is going into your object
- We start our mutator functions with the word ‘set’
- Example

```
void set_month(int month);
```

Sample Code

- Accessors and Mutators
  - `accessor_mutator.cpp`

Review

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