

CSCI 200: Foundational Programming Concepts & Design

Lecture 18



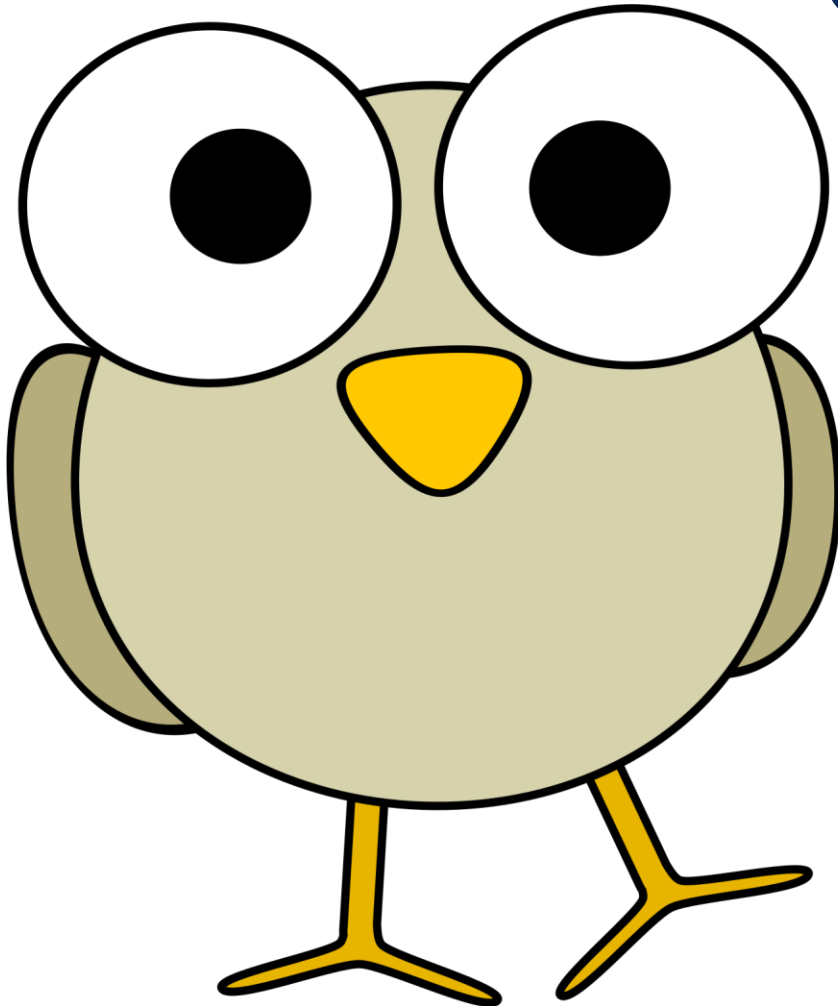
**Object-Oriented Programming:
Managing & Encapsulating State**

Previously in CSCI 200



- Vector implements dynamically sized list
- String implements dynamically sized character list
- Big O Notation describes runtime complexity of algorithm

Questions?



??

Learning Outcomes For Today



- Discuss the concept of encapsulation
- Draw a class diagram using UML to describe the structure of a class and its members
- Discuss the difference between a class and an object
- Create a class containing data members and member functions
- Compare and contrast Procedural Programming with Object-Oriented Programming

Learning Outcomes For Today



- Explain the following terms and how they are used
 - (1) dot operator / member access operator
 - (2) data member
 - (3) scope resolution operator
- Discuss the difference between
 - (1) a class and an object
 - (2) a class and a struct
- Explain and use the following terms
 - (1) constructors & destructors
 - (2) accessor modifiers
 - (3) accessor & mutator functions
- Discuss the concept of scope within and outside a class & struct

On Tap For Today



- Constructors
- Public & Private
 - Getters & Setters
- Practice

Class Declaration



- Place in its own header file ClassName.h

```
// inside Box.h

#ifndef BOX_H
#define BOX_H

class Box {
public:
    float height;
    float depth;
    float width;

};

#endif
```

Creating an Object



```
// inside main.cpp

#include "Box.h"
#include <iostream>
using namespace std;

int main() {
    Box smallBox;
    smallBox.height = 4;
    smallBox.width = 4;
    cout << "Enter the box length: ";
    cin >> smallBox.depth;
    cout << "The volume is: ";
    cout << smallBox.width * smallBox.height * smallBox.depth << endl;
    return 0;
}
```

Class Declaration



- Place in its own header file `ClassName.h`

```
// inside Box.h

#ifndef BOX_H
#define BOX_H

class Box {
public:
    float height;
    float depth;
    float width;
    float volume() ;
};

#endif
```

- Will get to implementation next

Class Definition



- Placed in a class implementation file
ClassName.cpp

```
// inside Box.cpp
#include "Box.h"

float Box::volume() {
    return height * depth * width;
}
```

- Functions have access to ALL data members of a class



- `::` is the **Scope Resolution operator**
 - Specifies which scope an identifier belongs to
- In our case, which class a function belongs to
 - `Box::volume()`
- Could have two classes with the same function name
 - `Box::volume()` `Tube::volume()`

Precedence Table

Category	Precedence	Operator	Associativity
Parenthesis	1	()	Innermost First
Scope Resolution	2	S::	Left to Right
Postfix Unary Operators	3	a++ a-- a. f()	
Prefix Unary Operators	4	++a --a +a -a !a ~a (type)a &a *p new delete	Right to Left
Binary Operators	5	a*b a/b a%b	Left to Right
	6	a+b a-b	
Shift Operators	7	a<<b a>>b	
Relational Operators	8	a<b a>b a<=b a>=b	
	9	a==b a!=b	
Bitwise Operators	10	a&b	
	11	a^b	
	12	a b	
Logical Operators	13	a&&b	
	14	a b	
Assignment	15	a=b a+=b a-=b a*=b a/=b a%=b a&=b a^=b a =b	Right to Left

Creating an Object



```
// inside main.cpp

#include "Box.h"
#include <iostream>
using namespace std;

int main() {
    Box smallBox;
    smallBox.height = 4;
    smallBox.width = 4;
    cout << "Enter the box length: ";
    cin >> smallBox.depth;
    cout << "The volume is: ";
    cout << smallBox.volume() << endl;
    return 0;
}
```

What happens if...



```
// inside main.cpp

#include "Box.h"
#include <iostream>
using namespace std;

int main() {
    Box smallBox;

    cout << "The volume is: ";

    cout << smallBox.volume() << endl; // what does it print?

    return 0;
}
```

On Tap For Today



- Constructors
- Public & Private
 - Getters & Setters
- Practice

Constructor



- A special function
- Named after the class name
- Has no return type
- Called automatically upon object creation
 - Used to setup/initialize/allocate object state

Creating a Constructor



```
// inside Box.h
```

```
class Box {  
public:  
    int height;  
    int width;  
    int depth;  
    Box();  
  
    int volume();  
};
```

```
// inside Box.cpp
```

```
#include "Box.h"  
  
Box::Box() {  
    height = 1;  
    width = 1;  
    depth = 1;  
}
```

Using a Constructor



```
// inside main.cpp

#include "Box.h"
#include <iostream>
using namespace std;

int main() {
    Box smallBox;                // use the default constructor implicitly
    Box squareBox = Box();       // use the default constructor explicitly
    Box myBox;                   // use the default constructor implicitly

    cout << smallBox.volume() << endl; // prints 1
    myBox.height = 4;
    myBox.width = 6;
    myBox.depth = 8;
    cout << myBox.volume() << endl;    // prints 192
    return 0;
}
```

Overloading a Constructor



```
// inside Box.h
```

```
class Box {  
public:  
    int height;  
    int width;  
    int depth;  
    Box();  
    Box(int h, int w, int d);  
    int volume();  
};
```

```
// inside Box.cpp
```

```
#include "Box.h"  
  
Box::Box() {  
    height = 1;  
    width = 1;  
    depth = 1;  
}  
  
Box::Box(const int H,  
        const int W,  
        const int D) {  
    height = H;  
    width = W;  
    depth = D;  
}
```

Using a Constructor



```
// inside main.cpp

#include "Box.h"
#include <iostream>
using namespace std;

int main() {
    Box smallBox;                // implicit default constructor
    Box myBox( 4, 6, 8 );        // implicit parameterized constructor
    Box squareBox1 = Box();      // explicit default constructor
    Box squareBox2 = Box(2, 2, 2); // explicit parameterized constructor
    cout << smallBox.volume() << endl; // prints 1
    cout << myBox.volume() << endl;    // prints 192
    return 0;
}
```

Box Rules



- All dimensions must be positive

Rule Enforced!



```
// inside Box.h
```

```
class Box {  
public:  
    int height;  
    int width;  
    int depth;  
    Box();  
    Box( int h, int w, int d );  
    int volume();  
};
```

```
// inside Box.cpp
```

```
#include "Box.h"
```

```
Box::Box( int h, int w, int d ) {  
    if(h > 0) height = h;  
    else      height = 1;  
  
    if(w > 0) width = w;  
    else      width = 1;  
  
    if(d > 0) depth = d;  
    else      depth = 1;  
}
```

```
// main.cpp
```

```
Box myBox(-5, -5, -5);  
cout << myBox.volume() << endl;    // 1
```

```
Box yourBox(5, 5, 5);  
cout << yourBox.volume() << endl; // 125
```

More Concerns



```
// inside Box.h
```

```
class Box {  
public:  
    int height;  
    int width;  
    int depth;  
    Box();  
    Box( int h, int w, int d );  
    int volume();  
};
```

```
// inside Box.cpp
```

```
#include "Box.h"
```

```
Box::Box( int h, int w, int d ) {  
    if(h > 0) height = h;  
    else      height = 1;  
  
    if(w > 0) width = w;  
    else      width = 1;  
  
    if(d > 0) depth = d;  
    else      depth = 1;  
}
```

```
// main.cpp
```

```
Box myBox(-5, -5, -5);  
cout << myBox.volume() << endl;    // 1
```

```
Box yourBox(5, 5, 5);  
cout << yourBox.volume() << endl; // 125  
yourBox.width = -5;  
cout << yourBox.volume() << endl; // -125
```

Box Rules



- All dimensions must be positive

Rule Violated!

On Tap For Today



- Constructors
- **Public & Private**
 - Getters & Setters
- Practice

Public Access



```
// inside Box.h
```

```
class Box {  
public:  
    int height;  
    int width;  
    int depth;  
    Box();  
    Box( int h, int w, int d );  
    int volume();  
};
```

```
// inside Box.cpp
```

```
#include "Box.h"
```

```
Box::Box( int h, int w, int d ) {  
    if(h > 0) height = h;  
    else      height = 1;  
  
    if(w > 0) width = w;  
    else      width = 1;  
  
    if(d > 0) depth = d;  
    else      depth = 1;  
}
```

```
// main.cpp
```

```
Box myBox(5, 5, 5);  
cout << myBox.volume() << endl; // 125  
  
myBox.width = -5;  
cout << myBox.volume() << endl; // -125  
  
myBox.height = -5;  
cout << myBox.volume() << endl; // 125
```

Private Access



// inside Box.h

```
class Box {
public:
    Box();
    Box( int h, int w, int d );
    int volume();
private:
    int height;
    int width;
    int depth;
};
```

// inside Box.cpp

```
#include "Box.h"

Box::Box( int h, int w, int d ) {
    if(h > 0) height = h;
    else      height = 1;

    if(w > 0) width = w;
    else      width = 1;

    if(d > 0) depth = d;
    else      depth = 1;
}
```

// main.cpp

```
Box myBox(5, 5, 5);
cout << myBox.volume() << endl; // 125
```

```
myBox.width = -5; // compiler error!
cout << myBox.volume() << endl;
```

```
myBox.height = -5; // compiler error!
cout << myBox.volume() << endl;
```

Private Access



// inside Box.h

```
class Box {  
public:  
    Box();  
    Box( int h, int w, int d );  
    int volume();  
private:  
    int _height;  
    int _width;  
    int _depth;  
};
```

// inside Box.cpp

```
#include "Box.h"  
  
Box::Box( int h, int w, int d ) {  
    if(h > 0) _height = h;  
    else     _height = 1;  
  
    if(w > 0) _width = w;  
    else     _width = 1;  
  
    if(d > 0) _depth = d;  
    else     _depth = 1;  
}
```

// main.cpp

```
Box myBox(5, 5, 5);  
cout << myBox.volume() << endl; // 125
```

```
myBox._width = -5; // compiler error!  
cout << myBox.volume() << endl;
```

```
myBox._height = -5; // compiler error!  
cout << myBox.volume() << endl;
```

Private Access



// inside Box.h

```
class Box {
public:
    Box();
    Box( int h, int w, int d );
    int volume();
private:
    int _height;
    int _width;
    int _depth;
};
```

// inside Box.cpp

```
#include "Box.h"

Box::Box( int h, int w, int d ) {
    if(h > 0) _height = h;
    else _height = 1;

    if(w > 0) _width = w;
    else _width = 1;

    if(d > 0) _depth = d;
    else _depth = 1;
}
```

// main.cpp

```
Box myBox(5, 5, 5);

cout << myBox.volume() << endl; // 125
cout << myBox._height << " " // compiler error!
    << myBox._width << " " // compiler error!
    << myBox._depth << endl; // compiler error!

myBox._height = 10; // compiler error!
```

On Tap For Today



- Constructors
- Public & Private
 - Getters & Setters
- Practice

Accessor Methods



- Aka “getters”
- A member function used to provide managed access to data members
- Allows a user to get (access) the value of a data member

Mutator Methods



- Aka “setters”
- A member function used to provide managed access to data members
- Allows a user to set (mutate) the value of a data member

Getters & Setters



```
// inside Box.h
class Box {
public:
    Box();
    Box( int h, int w, int d );
    int volume();
    int getHeight();
    void setHeight(const int H);
    // others for width & depth
private:
    int _height;
    int _width;
    int _depth;
};
```

```
// inside Box.cpp
#include "Box.h"

int Box::getHeight() {
    return _height;
}

void Box::setHeight(const int H) {
    if(H > 0) _height = H;
}

// others for width & depth
```

```
// main.cpp
Box myBox(5, 5, 5);
cout << myBox.volume() << endl; // 125

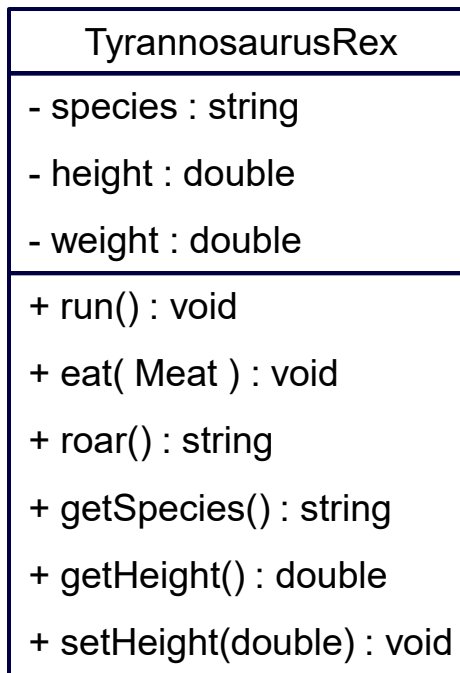
myBox.setWidth(-5);
cout << myBox.volume() << endl; // 125

myBox.setHeight(10);
cout << myBox.volume() << endl; // 250
```

UML Diagrams



- Use + - to denote public private



On Tap For Today



- Constructors
- Public & Private
 - Getters & Setters
- Practice

To Do For Next Time



- Today: complete Lecture 18 Post Class Survey
- Can formally start A3
 - Due Thu Oct 23