#2: Classes and Reference Variables

January 21, 2022 · *G Carl Evans*

Creating New Types

In data structures, we will be learning and creating new types of structures to store data. We will start simply – by the end, we will have types we built being the building blocks for new types!

Big Idea: Encapsulation

Our First Class – Cube:

Cube.h

```
#pragma once

class Cube {
  public:
   double getVolume();





  private:




};
```

Cube.cpp

```
#include "Cube.h"

double Cube::getVolume() {


}
```

Public vs. Private:

Situation	Protection Level
`Cube` functionality provided to **client code**	
Variable containing data about the `Cube`	
Helper function used in `Cube`	

Hierarchy in C++:

There `Cube` class we're building might not be the only `Cube` class.
Large libraries in C++ are organized into ____________________.

Cube.h

```
#pragma once

namespace cs225 {
  class Cube {
  public:
    double getVolume();

```

Cube.cpp

```
#include "Cube.h"

namespace cs225 {
  double
    Cube::getVolume() {
      return length_ *
             length_ * length_;
  }
```

Default Constructor:

Every class in C++ has a constructor – even if you didn't define one!

- Automatic/Implicit Default Constructor:
- Custom Default Constructor:

Cube.h

```
class Cube {
  public:
   Cube();
   /* ... */
```

Cube.cpp

```
Cube::Cube()  {


}
```

Custom, Non-Default Constructors:

We can provide also create constructors that require parameters when initializing the variable:

Cube.h

```
class Cube {
  public:
   Cube(double length);
   /* ... */
```

Cube.cpp

```
Cube::Cube(double length) {


}
```

Our First Program:

main.cpp

```
#include "Cube.h"
#include <iostream>

int main() {
  cs225::Cube c;
  std::cout << "Volume: " << c.getVolume() << std::endl;
  return 0;
}
```

...run this yourself: run `make` and `./main` in the lecture source code.

However, our program is unreliable. **Why?**

Default Constructor:
Every class in C++ has a constructor – even if you didn't define one!

- Automatic/Implicit Default Constructor:

- Custom Default Constructor:

Cube.h

```
class Cube {
  public:
   Cube();
   /* ... */
```

Cube.cpp

```
Cube::Cube() {


}
```

Custom, Non-Default Constructors:
We can provide also create constructors that require parameters when initializing the variable:

Cube.h

```
class Cube {
  public:
   Cube(double length);
   /* ... */
```

Cube.cpp

```
Cube::Cube(double length) {


}
```

Puzzle #1: How do we fix our first program?

puzzle.cpp w/ above custom constructor

```
  cs225::Cube c;
  cout << "Volume: " << c.getVolume() << endl;
```

...run this yourself: run `make puzzle` and `./puzzle` in the lecture source code.

Solution #1:

Solution #2:

The beauty of programming is both solutions work! There's no one right answer, both have advantages and disadvantages!

Pointers and References
Often, we will have direct access to our object:

```
Cube s1;  // A variable of type Cube
```

Occasionally, we have a reference or pointer to our data:

```
Cube & r1 = s1; // A reference variable of type Cube
Cube * p1;  // A pointer that points to a Cube
```

Pointers
Unlike reference variables, which alias another variable's memory, pointers are variables with their own memory. Pointers store the memory address of the contents they're "pointing to".

Three things to remember on pointers:

1.

2.

3.

main.cpp

```
int main() {
  cs225::Cube c;
  std::cout << "Address storing `c`:" << &c << std::endl;

  cs225::Cube *ptr = &c;
  std::cout << "Addr. storing ptr: "<< &ptr << std::endl;
  std::cout << "Contents of ptr: "<< ptr << std::endl;

  return 0;
}
```

Indirection Operators:

`&v`

`*v`

`v->`

CS 225 – Things To Be Doing:

1. Setup your computer
2. Join us on Discord