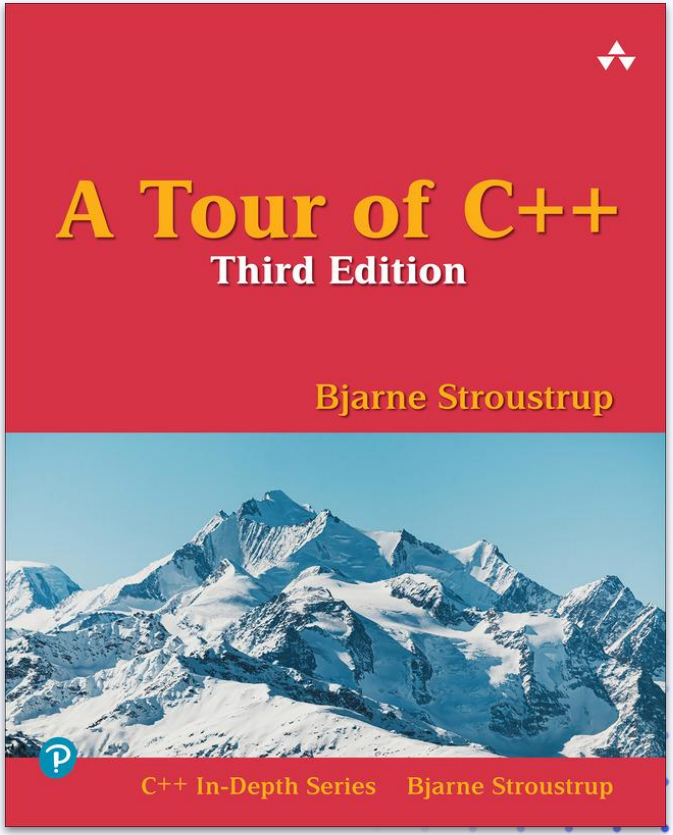
# Classes and Concrete Types

CSE 232 - Dr. Josh Nahum

# Reading:

Section 5.1 though Section 5.2.1



# Table of contents

# OO What's Not Taught

# Inheritance

The ability to build a new class out of an existing class, or implement an existing class in a different way is an essential part of object-oriented programming.

However, we will leave teaching that to future classes (notably CSE 335, "Object-Oriented Software"). You are welcome to read about those parts of the language (Section 5.3-5.5), but it is entirely optional.

# 01 Getters and Setters

# Const Member Functions

```
double real() const { return re; }
```

Note the in the **const** in the member function definition above. This is saying that the member function doesn't alter the object and hence can be called on const objects.

The **const** in const member functions comes **after** the list of parameters.

# Encapsulation

```
double real() const { return re; } // Getter
void real(double d) { re=d; } // Setter
double imag() const { return im; } // Getter
void imag(double d) { im=d; }  // Setter
```

Getter member functions provide read-only access to internal data members (like re and im). Setter member functions provide the ability to set internal data members. Together they provide a more controlled way to access and alter data members.

Note that getters are const member functions, so they can be used on const objects.

# 02

# Constructors

# Constructors

```
complex(double r, double i) :re{r}, im{i} {}
complex(double r) :re{r}, im{0} {}
complex() :re{0}, im{0} {} // Default Constructor
```

Constructors are how objects of a given class are created and initialized. The complex class has 4 overloaded constructors. Some constructors have special names because they are very commonly defined:

Default constructor: A constructor with no parameters.

# Copy Constructor

```
complex(complex z) :re{z.re}, im{z.im} {}
// From Book, WRONG!

complex(complex const & z) :re{z.re}, im{z.im} {} // Copy Constructor
// Correct
```

A "Copy Constructor" is a constructor that takes a reference to an object of the same class as its only parameter. It is usually used to make a **deep copy** of the object. A deep copy is a copy with no references or pointers to the original object. Changing a deep copy will never affect the original.

Note the book has a wrong implementation of a copy constructor.

# 03

# Operators

# Concepts

## this

The **this** keyword is a pointer to a member function's object.

## Returning a reference

```
complex & operator+=(complex z);
```

Some operators return a reference to allow chaining:

```
complex a{2, 3};
a += 2 += 3;
// Same as
(a += 2) += 3
// Which wouldn't be possible unless
// operator+= returned a reference.
```

# Example: operator<

```
complex a {1, 2};
complex b {3, 4};
std::cout << (a < b) << std::endl;
```

How do you compare complex numbers with each other?

- Only look at the real component?
- Add the real and imaginary components?
- Throw an exception?

- Compare the absolute value (the distance from the origin on the complex plane).

# Two ways to define most operators

As a member function (where the first argument is implicitly the object) or as a separate function.

Live Coding Demonstration!

# Attribution

## Please ask questions via Piazza

Dr. Joshua Nahum
www.nahum.us
EB 3504



**CREDITS:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**