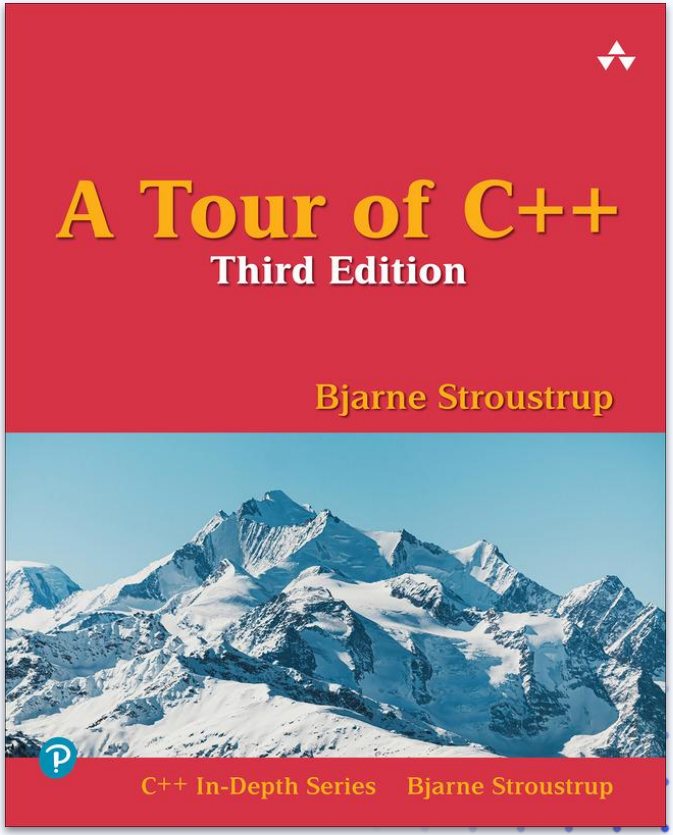
Types, Variables, and Arithmetic

CSE 232 - Dr. Josh Nahum

Reading:

Section 1.4



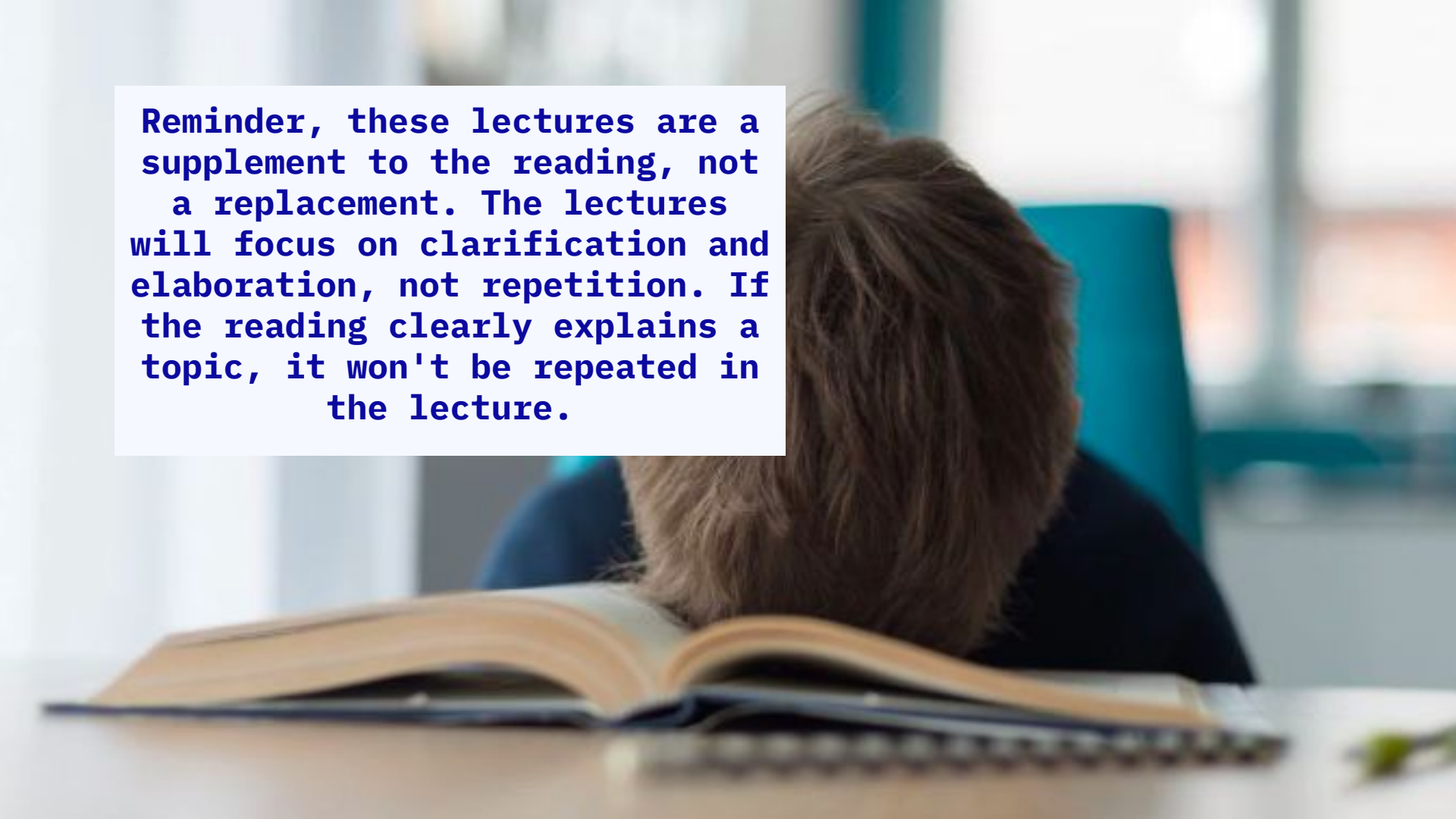

Reminder, these lectures are a supplement to the reading, not a replacement. The lectures will focus on clarification and elaboration, not repetition. If the reading clearly explains a topic, it won't be repeated in the lecture.

Table of contents

00

Types

```
int x = 42;
```

Vocabulary

Type

How the bit pattern is interpreted and what it can do, e.g. a 32 bit `int`.

Value

The set of bits, e.g. `42` or `0b101010`.

Object

Some amount of memory that holds a value of some type, e.g. the location of `x`.

Variable

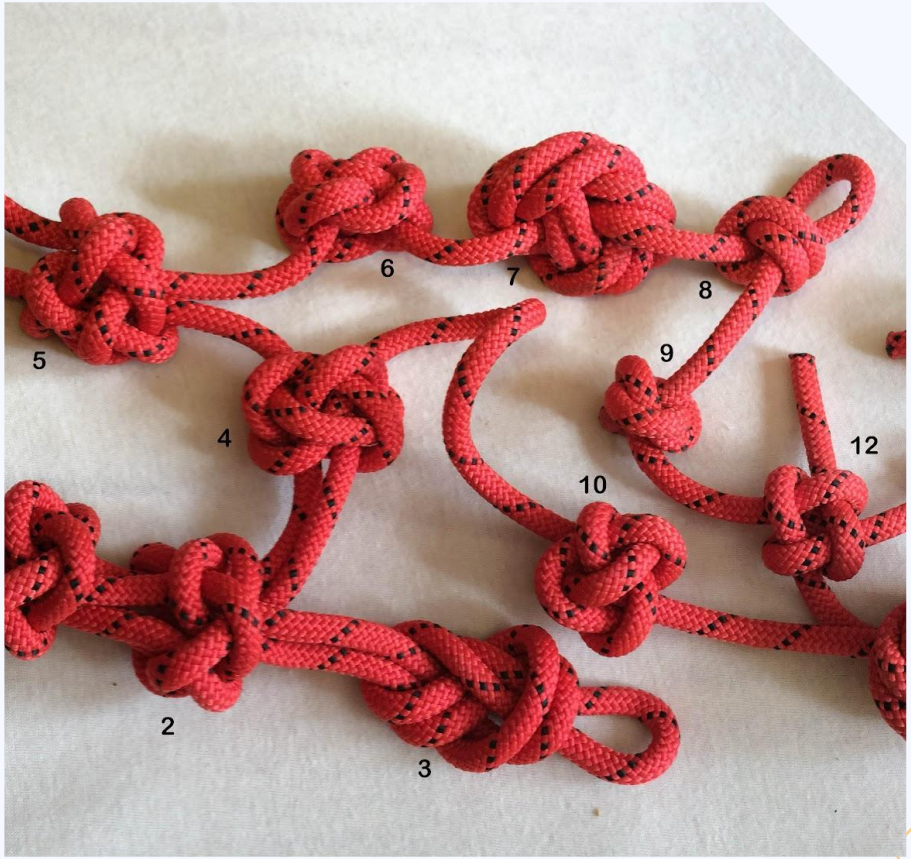
A named object, e.g. `x`.

Strings?

Text is generally represented in C++ in two different ways:

- arrays of `char`
- `std::string` in `<string>`

We will teach both later.



What about the other fundamental types?

As the reading indicates, only a few of the types on the right are in common use, for this course we only teach, and you should only use, the types that are most useful.

Type specifier	Equivalent type	Width in bits by data			
		C++ standard	LP32	ILP32	LLP
signed char	signed char	at least 8	8	8	8
unsigned char	unsigned char				
short	short int	at least 16	16	16	16
short int					
signed short					
signed short int					
unsigned short	unsigned short int				
unsigned short int					
int	int	at least 16	16	32	32
signed					
signed int					
unsigned	unsigned int				
unsigned int					
long	long int	at least 32	32	32	64
long int					
signed long					
signed long int					
unsigned long	unsigned long int				
unsigned long int					
long long	long long int (C++11)	at least 64	64	64	64
long long int					
signed long long					
signed long long int					
unsigned long long	unsigned long long int (C++11)				
unsigned long long int					

01

Arithmetic

Operators To Ignore

Bitwise Operators

This class doesn't teach bitwise operations, a.k.a. Bit Magic, so you won't need to know the following operators:

- `x & y`
- `x | y`
- `x ^ y`
- `~x`

Don't confuse logical operators `&&` and `||` for the bitwise operators `&` and `|`.

Postfix Increment/Decrement

c++ and c--

The reading skips over a common set of operators, the postfix increment and decrement.

They work identically to their prefix equivalents, but return the old value of the variable when used in a larger expression.

Postfix increment and decrement shouldn't be used unless this behavior is desired.

Example:

```
int a = 4;
int b = a++;
// a's value is 5, b's value is 4.

int x = 4;
int y = ++x;
// x's value is 5, y's value is 5.
```

02

Initialization

Most Vexing Parse

```
double x(int y);
```

Is the code above declaring a function named `x` that takes an `int` parameter named `y` and returns a `double`?

Or

It is an initialization for a variable `double` named `x` from the value of `y` converted to an `int`?

Solution

Use curly braces

```
double x{y};
```

Undefined Behavior

Division by 0

```
return x / 0;
```

Accessing Uninitialized Memory

```
int x;
std::cout << x;
```

Others

Dereferencing a null pointer, out of bounds memory access, signed overflow, ...

Undefined behavior is a way for C++ to give compilers permission to not check and not guarantee any observable behavior for code. It allows them to write more efficient compilers, at the expense of erroneous code. **Never write code with undefined behavior.**

Types Of Bad Code

Ill-Formed

Syntax errors that a compiler must identify

Unspecified Behavior

Varies between implementations, but the compiler isn't required to specify exact behavior

Implementation Defined Behavior

Code that does different things in different environments. Compiler must specify the exact behavior

Undefined Behavior

No restrictions on behavior, the code could do anything!

03

auto

When to use auto?

Rarely (while you are learning). Auto is a powerful tool for simplifying and clarifying your code. But it often makes debugging type errors more challenging because you aren't being explicit with your types.

Only use auto when you know the type it will infer. We strongly recommend avoiding its use until you are more confident in your C++ skills. We will require explicit types in a solution if you ask for help regarding compiler errors.

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**