Week 1

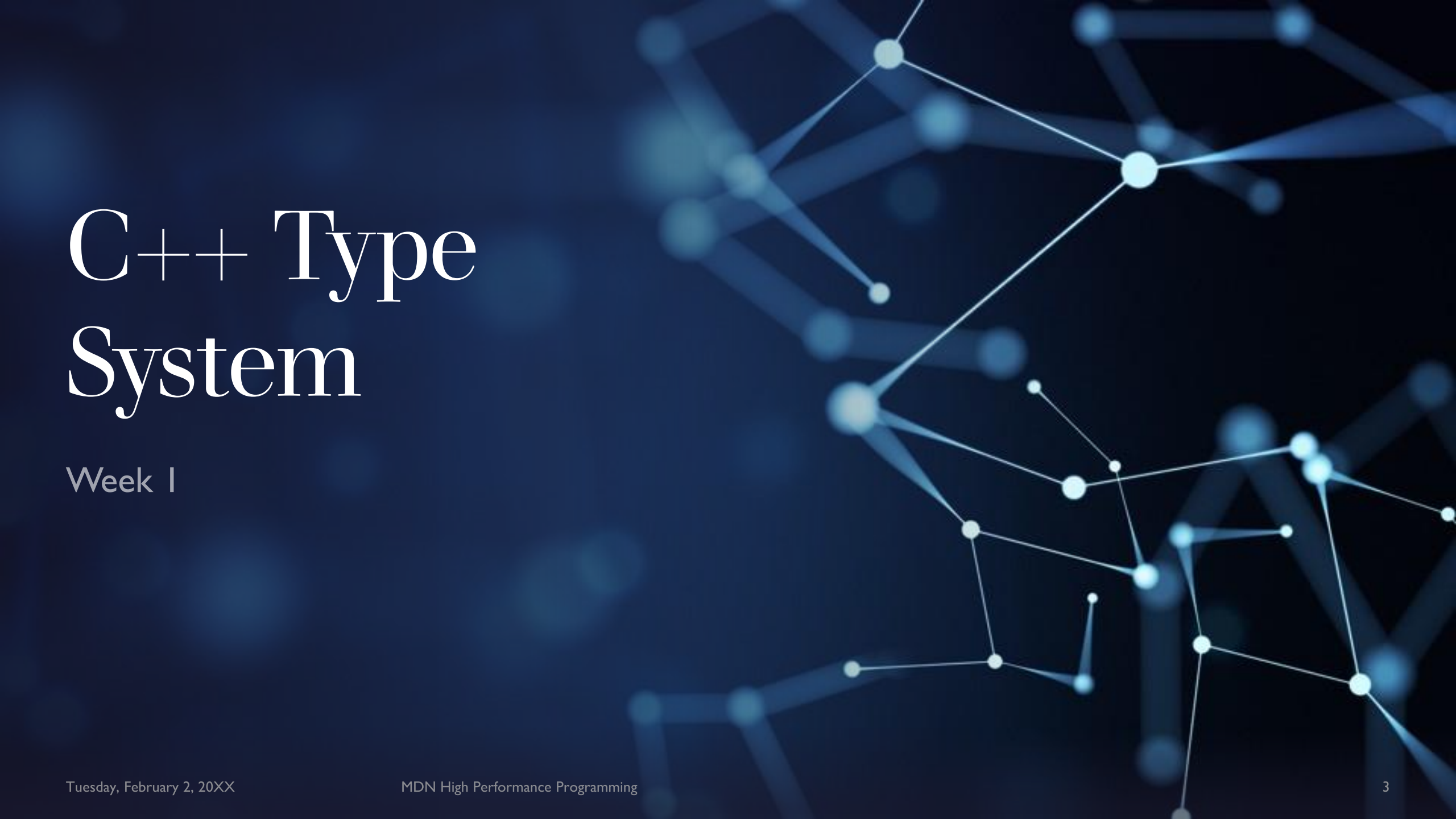
Tyler Swann

Agenda

- Recap
- C++ Type System
- Types
- Variables
- Operators
- Discussion

C++ Type System

Week 1

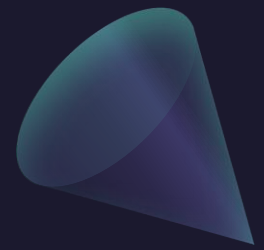


Type System

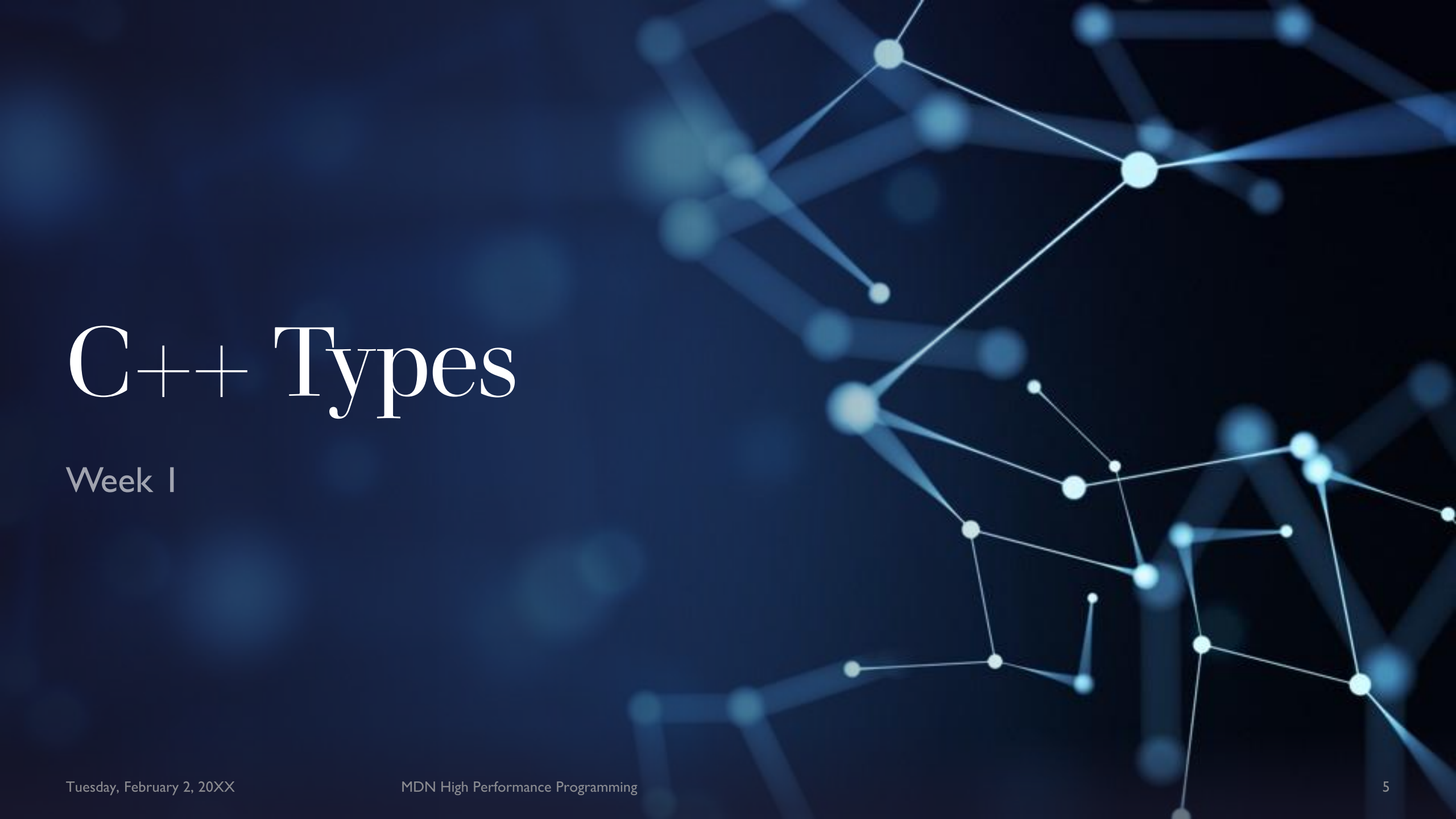
WHAT IS A TYPE SYSTEM

- A type system is a set of rules that govern the behavior and form the basis of the grammar of a language.
- How a programming languages dictate the notation if types and how types are assumed form the basis of its type system.

C++



- C++ has a strong type system
- C++ is statically typed
- C++ has a very rigorous definition of its type system and the various relationship between types
- C++ has the following type categories
  - Literals
  - Values
  - Types
  - Typeclasses



C++ Types

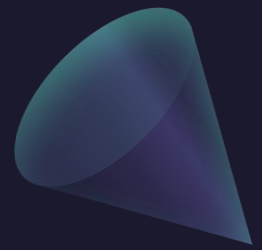
Week 1

Integral and Floating-Point Types

- `bool` – Boolean type – 8-bits – 1-byte
- `char` – character type – 8-bits – 1-byte
- `wchar_t` – wide character type – 16-bits or 32-bits – 2-bytes or 4-bytes
- `int` – integer type – 32-bits – 4-bytes
- `float` – single precision, floating-point type – binary32 format
- `double` – double precision, floating-point type – binary64 format

Other Types

- `void` – incomplete type – denotes no return.
- `nullptr` – literal for a pointer to nothing
- `std::nullptr_t` – type of `nullptr`
- `std::size_t` – Platform specific, maximum unsigned integer value
- `std::ptrdiff_t` – Type returned by the subtraction of two pointers
- `auto` – Automatic type (via deduction)



Variables

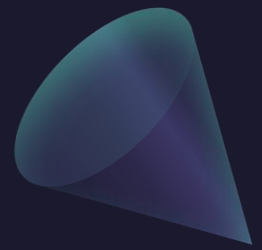
Week 1

Initialisation

WHAT ARE VARIABLES AND WHAT IS INITIALISATION

- A variable is an object or entity that has a single type and a single value.
- Variables store data for later use.
- Initialisation is the process of giving a variable a value of the variables type
- In C++, there are many ways to initialise a variable depending on the context.

KINDS OF INITIALISATIONS



- Default
- Value
- Copy
- Direct
- Aggregate
- List

Qualifiers

SIGNED-NESS AND SIZE

- `signed` – Makes integral signed
- `unsigned` – Make integral unsigned
- `short` - Integral with at least 16-bits (2-bytes)
- `long` - Integral with at least 32-bits (4-bytes)
- `long long` - Integral with at least 64-bits (8-bytes)
- `unsigned` can be used in combination with the size qualifiers increase the maximum possible value.

STORAGE AND MUTABILITY

- `static` - Declares static storage
- `inline` - In-lines a function call
- `const` - Data is immutable
- `constexpr` - Data may be evaluated at compile time
- `volatile` – Data is likely to change outside the compilers insight.

Automatic Type Deduction

- C++ allows for the elision of type declaration through the use of type deduction.
- Type deduction takes the surrounding context of an expression and is able to infer what type a variable should be.
- Automatic types are introduced using the `auto` keyword.
- The type on the right-hand-side must be obvious to the compiler.
- E.g. `auto a = {1};` Here it is clear that `a` is an `int`.

Value Categories

LVALUES

- Found on the left-hand-side of the assignment operator (=).
- Indicates copy semantics when used in the right-hand side of =.

RVALUES

- Found on the right-hand-side of the assignment operator (=).
- Indicates a temporary value.
- Indicates move semantics.

Operators

Week 1

Basic Arithmetic

OPERATORS

- C++ has the typical operators you would expect to do basic arithmetic with integral and floating-point types.
- `+` - Addition
- `-` - Subtraction
- `*` - Multiplication
- `/` - Division
- `%` - Modulus

ABOUT DIVISION AND MODULO

- Division of two integral types will perform integer division, where the remained will be discarded.
- You must cast an integral to a floating point type.

Basic Arithmetic

OPERATORS

- C++ has the typical operators you would expect to do basic arithmetic with integral and floating-point types.
- `+` - Addition and unary posigate
- `-` - Subtraction and unary negate
- `*` - Multiplication
- `/` - Division
- `%` - Modulus
- `++` - Increment (prefix and postfix)
- `--` - Decrement (prefix and postfix)

ABOUT DIVISION AND MODULO

- Division of two integral types will perform integer division, where the remained will be discarded.
- You must cast an integral to a floating point type.
- Modulo does not work on floating point types.

Casting

- Casting allows for conversion of the type from and expression to a new type
- `const_cast<T>(/* expr */);` - Changes cv-qualifications
- `static_cast<T>(/* expr */);` - Converts type
- `reinterpret_cast<T>(/* expr */);` - Reinterprets the underlying bits
- `dynamic_cast<T>(/* expr */);` - Allows fir casting up, down and across the class hierarchies
- `static_cast<T>()` is the one we will use most.

Bitwise Arithmetic

OPERATORS

- Bitwise operators allow for the manipulation of the underlying bits of a value in memory.
- & - And
- | - Or
- ^ - Xor
- << - Left Shift
- >> - Right Shift

FLOATING POINTS

- Bitwise operators only work for integral types.
- They do not work for floating point types.

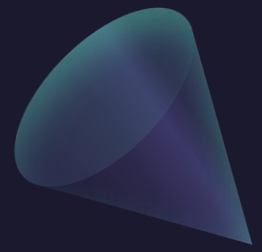
Arithmetic Assignment

- In C++, there are also assignment variants of all the arithmetic operators that perform the binary operation and then assign the result to the left point (argument).

- `+=` - Add assign
- `-=` - Sub assign
- `*=` - Multiply assign
- `/=` - Divide assign
- `%=` - Modulo assign
- `&=` - And assign
- `|=` - Or assign
- `^=` - Xor assign
- `<<=` - Left Shift assign
- `>>=` - Right shift assign

Size operator

- You can obtain the size of a type; in bytes using the `sizeof()` operator.
- This returns a `std::size_t` type.



Discussion

- Any questions?
- Need help?
- Open discussion.
- Concerns?

Next Week

1. Conditional Operators
2. Ordering
3. Logical Operators
4. Conditional Expressions
5. Loops
6. IO

Summary

This week you learnt about C++'s type system, what variables are and how to perform actions using operators.

Thank You

Tyler Swann

https://github.com/MonashDeepNeuron/HPP