Short Summary for [C++ Programming Course - Beginner to Advanced](https://www.youtube.com/watch?v=8jLOx1hD3\_o) by [Merlin](https://merlin.foyer.work/)

Learn C++ from beginner to advanced level with experienced developer Danielle Gaguaya. Covers fundamentals, procedures, and object-oriented programming. Uses Visual Studio Code.

[00:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=0) Learn C++ from beginner to advanced

[06:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=363) Install an IDE and a compiler depending on your operating system

[17:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1050) Install and configure different C++ compilers and environment variables in Windows

[22:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1374) Install Visual Studio Code and connect it to compilers

[34:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2044) Configure GCC with C++20 support in Visual Studio Code

[39:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2365) Connecting Visual Studio Code to GCC and Clang compilers

[50:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3000) Configure Visual Studio Code to use the MSVC compiler for C++ applications

[55:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3318) Setting up C++ development environment on Windows and Linux

[1:06:24](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3984) Configure Visual Studio Code to use gcc compiler

[05:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=304) Compile a C++ project on a Linux system

[1:22:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=4955) Install C++ compiler on Mac using Homebrew

[1:28:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=5280) Install Visual Studio Code on Mac and configure compilers

[1:38:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=5936) Compiler Explorer is a good tool to try out different compilers and share code

[1:44:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=6277) Setting up a C++ project in Visual Studio Code with GCC compiler

[1:56:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=6972) Learned about comments in C++

[2:01:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=7298) Errors and warnings in C++ programs

[2:12:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=7954) Statements and functions are key concepts in C++ programming

[2:18:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=8289) Functions are reusable pieces of code

[2:29:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=8959) Statements and functions are basic units in C++ programs

[2:35:10](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=9310) Learn about data input through streams in c++

[2:46:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=9981) You can grab data with spaces using std getline

[2:51:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=10304) Understanding the memory model is crucial for C++ programming.

[3:03:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=10999) Number systems allow us to transform data between convenient human-readable formats and binary.

[3:09:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=11345) Binary numbers can be grouped in different sizes of data to represent different ranges of values.

[3:20:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=12026) Understanding binary numbers and their representations in memory.

[05:59](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=359) Multiple ways to initialize variables in C++

[3:37:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=13056) Learned about integers and their initialization

[3:43:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=13410) Integer modifiers can be used to store positive and negative numbers, but only for integral data types.

[3:55:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=14103) Floating points in C++

[4:01:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=14465) Floating point numbers have limited precision

[4:12:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=15152) Learned about floating point numbers and their properties.

[4:18:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=15482) Booleans in C++ store two states, true or false, and take up one byte in memory.

[4:29:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=16152) Working with character data types and the auto keyword in C++

[4:35:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=16503) Auto type deduction in C++

[4:46:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=17204) Learn basic arithmetic operations in C++

[4:52:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=17545) Learned about basic arithmetic operations on integers in C++

[5:04:07](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=18247) Precedence table and parentheses can clarify order of operations

[5:10:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=18605) Learned about precedence and associativity

[5:21:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=19285) Learn about compound assignment operators.

[00:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26) C++ compound assignment operators and their usage

[5:39:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=20342) Learned about relational and logical operators in C++

[5:45:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=20705) Logical operators can be combined to create complex expressions

[5:57:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21426) Learn about manipulators such as stdendl, sdd flush, and std set w.

[6:03:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21792) Manipulate and display data in different bases and formats using C++ manipulators

[6:14:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=22489) Learn how to format data in C++ using set width and justification.

[6:20:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=22834) Modify boolean data display and output base system for terminal display

[00:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37) Documentation is crucial for learning.

[6:38:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=23889) Learned about the limits library and math functions in C++

[6:49:47](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=24587) Learned about various math functions in C

[6:55:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=24935) Integral types can have implicit conversions during arithmetic operations

[7:07:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=25620) Learn conditional programming using logical operators in C++

[7:12:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=25950) Nested if statements example

[7:23:28](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26608) Use else if statements to test for multiple conditions

[7:28:43](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26923) Learn about the switch statement in C++

[7:39:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=27566) Learned about switch statements and ternary operators

[7:45:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=27906) Learn how to use ternary operators in C++

[7:56:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=28582) Understanding how a loop works in C++

[8:01:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=28908) Loops in C++

[8:12:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=29549) Learn about for loops and iterator scope

[8:17:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=29875) Iterators can be declared inside or outside of a loop in C++

[8:28:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=30507) Using while loop to print 'I love C++' 10 times

[8:33:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=30818) Learned about the syntax of a while loop and do-while loop

[8:44:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=31448) Arrays are a way to group many variables together and manipulate them as a single unit in C++

[01:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86) Arrays in C++ can group variables under one entity

[9:00:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=32455) Writing and reading data in arrays using C++

[9:06:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=32778) Learn how to declare and initialize arrays in C++

[9:17:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=33434) Use std::size to get the size of an array at runtime

[9:22:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=33774) Use the std::size facility in the C++ standard library to get the count of elements in an array

[9:34:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=34443) Properly terminate C strings for correct output

[9:39:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=34788) Working with character arrays and string literals in C++

[9:51:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=35473) Don't go overbound, work within the legal boundaries of your array.

[9:56:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=35805) Pointers store addresses to variables and should be used carefully

[10:07:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=36465) Pointers and their usage in C

[10:13:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=36809) Learned how to declare, initialize, and use pointers in C

[10:24:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37498) C++ program memory map and virtual memory

[10:30:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37855) Memory Management Unit maps program memory to real RAM for efficient use.

[10:42:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=38534) Dynamic memory allocation in C++

[10:47:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=38869) Do not use or reference uninitialized memory in C++.

[10:58:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=39516) Dynamic heap memory pointers in C++

[11:03:57](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=39837) Avoid deleting memory twice in C++

[08:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=486) Avoid using deleted memory in pointers

[11:20:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=40821) Avoiding problems with dangling pointers

[11:31:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41473) Prevent program crashes with two options: exception mechanism and std no throw

[11:36:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41795) Learn about null pointer safety and how to handle it.

[11:47:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=42453) Avoid Memory Leaks

[11:53:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=42798) Avoid memory leaks in C++ programs

[12:04:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=43455) Exploring the differences between static and dynamic arrays

[01:17](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77) Dynamic arrays and references in C++

[12:20:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=44438) References and pointers have key differences in syntax and usage.

[12:26:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=44761) References and pointers can be used to read and write variables, but references cannot be reassigned to reference a different location in memory.

[12:36:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=45418) Const keyword creates a read-only reference

[12:42:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=45746) Simulating reference behavior with pointers

[12:53:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=46398) Manipulating character cases in C++

[12:58:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=46720) Try different compilers to learn about facilities from cctype

[13:09:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=47346) Learn to manipulate C strings using the C++ standard library

[13:14:20](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=47660) The sdrchr facility helps find characters in a string.

[13:25:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=48300) Function to compare strings based on specified number of characters.

[13:30:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=48614) Using std::strchr to search for a character in a string

[13:40:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=49213) Learn how to concatenate strings in C++

[00:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21) Learn how to concatenate and copy strings in C using strcat and strcpy functions.

[13:55:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=50158) New version of str cat allows specifying character count

[14:01:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=50483) std string is a high level type that simplifies working with string data in C++

[14:11:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=51110) Avoid wasting memory by using std strength

[14:17:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=51432) The one definition rule states that freestanding variables and functions cannot have multiple definitions.

[14:27:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=52060) Functions in C++

[14:33:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=52392) Functions in C++ can take parameters and return values

[14:43:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53034) Learn how to declare and call functions in C++.

[14:49:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53355) Demonstration of defining and calling functions in C++

[14:59:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53976) Functions can be split into declaration and definition to provide a summary of what the function does

[15:04:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=54298) Create a declaration and definition for the maximum function in C++

[15:15:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=54936) Program compilation model expanded to preprocessing, compilation, and linking

[15:21:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=55274) Splitting Functions Across Multiple Files in C++

[15:32:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=55946) Split functions across multiple files for organizational purposes.

[15:37:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=56276) Splitting functions across multiple files for project organization

[15:48:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=56909) Pass parameters by pointer to avoid copying

[15:54:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=57241) Passing parameters by pointer and reference in C++

[15:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=911) Using function parameters to output data from the function.

[16:10:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=58244) Learn how to use references and pointers to make changes in function stick and be visible on the outside

[16:21:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=58901) Returning by value creates copies of local variables.

[16:27:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=59229) Returning by value can lead to unexpected optimization by the compiler.

[16:38:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=59892) Functions with the same name can be set up to take different parameters.

[16:43:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=60220) You can set up all kinds of overloads for your functions in C++

[16:54:31](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=60871) Lambda functions can be declared with a capture list, parameter list, and body.

[19:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1182) Lambda functions in C++

[17:10:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=61808) How to set up a lambda function that returns something

[17:15:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=62130) Exploring lambda functions in C++

[17:26:10](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=62770) Learn how to capture variables in lambda functions.

[17:31:31](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=63091) Lambda functions can capture variables by value or reference

[17:42:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=63732) Function templates are just blueprints that get generated by the compiler based on the arguments you call your function with.

[17:47:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=64043) Function templates in C++ allow for flexible, efficient code.

[17:58:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=64706) Function templates are blueprints used to generate actual functions and can avoid multiple function overloads.

[18:03:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65016) Use compiler tools like cpp insights to understand errors in function templates.

[18:14:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65681) Function templates may not always behave as expected

[18:19:53](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65993) Template type deduction and explicit arguments

[18:30:46](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=66646) Using explicit template arguments allows passing different types for template arguments.

[18:36:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=66983) Passing template parameters by reference

[18:47:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=67631) Function template specialization allows bypassing the default mechanism for passing arguments.

[18:52:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=67952) Template specialization can be useful when working with pointers

[19:03:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=68598) Template specialization and concepts in C++

[19:08:53](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=68933) Using concepts to constrain function templates

[19:20:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=69606) C++ 20 provides multiple syntaxes for using concepts in function templates.

[19:25:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=69938) Create custom concepts in C++ using type traits.

[19:36:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=70616) Creating and using C++ concepts for type constraints

[19:42:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=70952) Learn about types of requirements in requires close

[19:53:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71586) Enforcing requirements in C++ using concepts.

[19:58:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71889) Combining concepts using logical operators

[20:08:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=72530) Enforcing concepts on variables and functions using the auto keyword

[20:13:57](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=72837) Learn about classes in C++ to build your own types and use them in your programs.

[20:25:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=73503) Create a class to compute the volume of a cylinder.

[20:30:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=73826) Declaring a class in C++

[20:41:28](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=74488) Creating a cylinder object using constructor with no parameter

[20:47:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=74821) Constructors are used by the compiler to build objects

[20:58:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=75486) Constructors must be public to be usable outside of the class

[13:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=809) Using getters and setters to modify member variables in a class

[21:14:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76443) Separate class definitions into header files for better organization

[21:19:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76749) Splitting code into multiple files and using include guards

[21:30:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77400) Manage class objects through pointers

[21:35:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77740) Managing objects through pointers in C++

[21:46:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=78404) Learn about constructors and destructors in C++

[21:52:07](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=78727) Learn about C++ destructors

[22:02:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=79357) Always explicitly release memory allocated through new operator to avoid memory leaks.

[22:07:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=79678) Learn about the this pointer in C++

[22:18:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=80335) Using the 'this' pointer to manipulate addresses in functions for a class

[22:24:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=80651) Using the 'this' pointer to set member variables

[22:34:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=81261) Structs have public member variables by default

[22:39:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=81551) Classes vs Structs

[22:49:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=82178) Size of an object is determined by the sum of sizes of its member variables.

[22:54:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=82489) Object oriented programming with inheritance in C++

[23:05:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=83134) Inheriting from person, the player class has a person part inside of it.

[01:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71) Understanding public inheritance in C++

[23:21:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=84095) Protected member variables allow for access and modification from derived classes in public inheritance

[23:27:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=84428) Using protected member variables in base class for easy data access in derived class constructors.

[23:38:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=85081) Private inheritance is the highest level of inheritance in C++

[23:43:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=85399) Creating a Player class that inherits from Person

[23:53:46](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86026) Protected inheritance makes public things private and protected things private in derived classes

[23:59:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86355) Engineer class inherits private variables from base class

[00:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1) Private inheritance makes all inherited member variables and functions private to the inheriting class.

[24:15:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=87340) Private inheritance strips down access to public and protected members in upstream class

[24:26:52](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=88012) Private inheritance in C++ strips down access to private level access in an inheritance class

[24:32:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=88357) Resurrected members take the access level of the section where they are resurrected

[24:43:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89013) Private inheritance can cause accessibility issues in downstream classes

[24:49:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89351) Always provide default constructors for your classes.

[24:59:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89982) Custom constructors are needed to forward information when building objects with inheritance

[05:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=322) Adding constructors to person and engineer classes

[25:16:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=90964) Custom constructors can be used to set up objects in layers

[25:21:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=91293) Custom constructors in inheritance hierarchy

[25:32:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=91923) Compiler strips off engineer information when initializing a person object

[25:37:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=92254) Implementing copy constructor for Engineer

[47:39](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2859) Demonstration of Custom Copy Constructor in C++

[25:53:17](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=93197) Inheriting constructors initializes only the base member variables

[13:24](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=804) Inheritance and destructors call structures in reverse order to constructors

[26:08:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=94122) Understanding the order of calling constructors and destructors

[26:19:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=94753) Polymorphism allows for managing derived objects using base class pointers or references.

[26:24:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=95073) Polymorphism allows us to store different types of objects in an array

[26:35:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=95736) Polymorphism in C++ solves looping and function problems

[00:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41) Classes Shape, Oval, and Circle were successfully implemented and integrated

[26:51:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=96687) Static binding in C++ inheritance hierarchy can lead to bad design

[26:56:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97010) Marking draw method as virtual enables dynamic binding

[27:07:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97655) Dynamic binding allows for easy management of different shapes in C++

[27:12:52](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97972) Achieving dynamic binding using virtual functions

[27:24:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=98652) Using base references or pointers is necessary for dynamic polymorphism.

[27:29:43](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=98983) Be careful when storing derived objects in collections

[11:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=666) Storing derived objects in an array designed for base class data leads to slicing

[00:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2) Use of 'override' keyword can help avoid polymorphic behavior issues

[27:57:59](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=100679) Understanding overloading and overriding in C++

[28:03:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=100994) Polymorphism must be declared at the top level class

[28:14:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=101662) Animal class hierarchy with virtual methods

[01:16](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76) Achieving animal polymorphism through inheritance hierarchy

[28:29:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=102596) Polymorphism and inheritance in C++

[28:34:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=102895) Understanding polymorphism with static member variables

[28:44:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=103488) Use of static member variables with inheritance and polymorphism

[28:49:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=103772) Restrict overriding and inheritance using final specifier

[29:00:16](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=104416) Final classes cannot be inherited in C++

[29:05:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=104721) Using default arguments with virtual functions can lead to unexpected results with polymorphism.

[29:15:51](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=105351) Polymorphism with default parameters in C++

[29:21:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=105661) Avoid using default arguments or default parameters with polymorphism

[29:31:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=106281) Mark destructors as virtual in inheritance hierarchy

[01:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=87) Transforming from a base pointer to a derived pointer using dynamic\_cast

[29:46:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=107205) Non-virtual function cannot be called through a base pointer

[29:51:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=107510) Transforming base pointer into derived pointer

[30:01:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108097) Dynamic casts work with polymorphic inheritance hierarchies

[30:06:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108401) Dynamic casts are meant to work only with pointers or references

[30:16:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108996) Polymorphism not working due to calling virtual function before derived object is constructed

[30:21:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=109309) Never call virtual functions from constructors or destructors

[30:32:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=109941) Abstract classes can't be instantiated

[30:37:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=110249) Demonstration of polymorphism in C++

[30:48:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=110916) Interfaces in C++ are super powerful.

[30:53:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=111234) Implement stream insertable interface to enable polymorphism in printing objects

[31:04:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=111854) Introduction to interfaces in C++

---------------------------------

Detailed Summary for [C++ Programming Course - Beginner to Advanced](https://www.youtube.com/watch?v=8jLOx1hD3\_o) by [Merlin](https://merlin.foyer.work/)

Learn C++ from beginner to advanced level with experienced developer Danielle Gaguaya. Covers fundamentals, procedures, and object-oriented programming. Uses Visual Studio Code.

[00:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=0) Learn C++ from beginner to advanced

- Course covers fundamentals of C++ programming language

- Course teaches object-oriented programming with inheritance and polymorphism

[06:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=363) Install an IDE and a compiler depending on your operating system

- Visual Studio Code is recommended as the IDE on all operating systems

- Install different compilers on Windows, Linux, and Mac for better portability

- GCC is the main compiler used in the course

[17:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1050) Install and configure different C++ compilers and environment variables in Windows

- Download and install GCC and Clang compilers and add them to environment variables

- Install Microsoft Visual Studio with the underlying compiler for C++ development on Windows

[22:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1374) Install Visual Studio Code and connect it to compilers

- Download Visual Studio Code from official website

- Install C/C++ extension to enable C/C++ development in Visual Studio Code

- Connect Visual Studio Code to installed compilers to compile projects

[34:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2044) Configure GCC with C++20 support in Visual Studio Code

- Find and copy the flag for C++20 mode from Stack Overflow

- Update the configuration file and build the program using the terminal

[39:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2365) Connecting Visual Studio Code to GCC and Clang compilers

- Configure tasks.json file to use GCC and Clang compilers

- Specify compiler options and output file name in tasks.json file

- Build project using selected compiler through terminal

- Remove binary and rebuild project with different compiler through terminal

[50:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3000) Configure Visual Studio Code to use the MSVC compiler for C++ applications

- Add a new entry in the tasks.json file for building with MSVC

- Configure Visual Studio Code for intellisense and autocomplete

- Configure Visual Studio Code to use the C++20 standard

[55:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3318) Setting up C++ development environment on Windows and Linux

- Install MSVC compiler and configure IntelliSense in Visual Studio Code on Windows

- Install GCC, G++, and GDB on Ubuntu 2004 and install Clang on Linux

[1:06:24](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=3984) Configure Visual Studio Code to use gcc compiler

- Install the C++ extension from Microsoft

- Configure task to use gcc compiler

[05:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=304) Compile a C++ project on a Linux system

- Set up an entry in tasks.json file to build and run the program

- Use specific instructions to compile with C++20 mode

- Compile every single CPP file in the directory

- Specify the output file name as 'rooster'

- Configure a task to use clang compiler

[1:22:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=4955) Install C++ compiler on Mac using Homebrew

- Homebrew is a package manager for Mac that makes installing GCC and Clang super easy.

- Install Homebrew using the command on the website provided.

- Use Homebrew to install GCC and Clang by running 'brew install gcc' and 'brew install llvm'.

- After installation, binaries for the compilers can be found in the user local directory.

[1:28:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=5280) Install Visual Studio Code on Mac and configure compilers

- Download and install Visual Studio Code from the website

- Install the C++ extension and configure it to use the compilers installed on the system

[1:38:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=5936) Compiler Explorer is a good tool to try out different compilers and share code

- It allows you to see assembly code generated for C++ code

- You can try modifying code and testing with different compilers

[1:44:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=6277) Setting up a C++ project in Visual Studio Code with GCC compiler

- Open a new folder and select the location of the project

- Include iostream to print messages to the console

- Use the main function as the starting point of the program

[1:56:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=6972) Learned about comments in C++

- Single line comments are done using // and can be placed anywhere in the code

- Block comments are done using /\* \*/ to comment out a block of text

- Nesting block comments is not allowed and will result in errors

[2:01:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=7298) Errors and warnings in C++ programs

- Compile time errors occur when code does not meet the requirements of the compiler

- Runtime errors occur when the program does not function as intended

- Warnings are issues that are not severe enough to halt compilation

[2:12:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=7954) Statements and functions are key concepts in C++ programming

- Statements are executed in order and must end with a semicolon

- Functions are like machines that take input and produce output

[2:18:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=8289) Functions are reusable pieces of code

- Functions group together a bunch of statements to do a task

- Functions allow for code reuse and reduce the need for rewriting code

- Variables can store data for use in a program

- Data stored in variables can be changed without manually changing code

- Functions can take input and return output

- Functions can be called multiple times without rewriting code

[2:29:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=8959) Statements and functions are basic units in C++ programs

- Statements are executed in order from top to bottom of the program

- Functions have a return value, a name, parameters, and a body within curly braces

[2:35:10](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=9310) Learn about data input through streams in c++

- Use cin to get data from the user and store it in variables

- Use std::getline to read data with spaces

- Understand that data flows from the program to the output stream

[2:46:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=9981) You can grab data with spaces using std getline

- Use std getline to take input with spaces

- std getline takes an input stream and stores data in a variable

[2:51:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=10304) Understanding the memory model is crucial for C++ programming.

- The CPU allocates memory and runs programs statement by statement.

- Knowing the memory model is necessary to optimize code and use C++ features.

[3:03:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=10999) Number systems allow us to transform data between convenient human-readable formats and binary.

- Binary numbers are made up of ones and zeros, and can be easily converted to decimal using basic math.

- Different number systems include binary, octal, and hexadecimal, each with their own base and range of representable numbers.

[3:09:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=11345) Binary numbers can be grouped in different sizes of data to represent different ranges of values.

- Grouping in groups of 3, 4, 5, 8, 16, 32 or 64 bits can represent different ranges of values.

- A mathematical formula can be used to determine the range of values represented by a certain number of digits.

- Hexadecimal number system can shorten the length of representing binary numbers in memory.

[3:20:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=12026) Understanding binary numbers and their representations in memory.

- Different representations like binary and hexadecimal can be used depending on convenience.

- Variables are named pieces of memory used to store specific types of data in binary format.

[05:59](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=359) Multiple ways to initialize variables in C++

- Curly braces and functional notation can be used for initialization

- Assignment initialization is another option

- Rules for variable naming in C++

- Querying the size of a variable in memory is possible

- Example code and demonstration in Visual Studio Code

[3:37:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=13056) Learned about integers and their initialization

- Integers occupy 4 bytes in memory and can be initialized using braces, parentheses, or assignment notation

- Size of can be used to check the size of integer variables in memory

- Integers can be signed or unsigned to store positive or negative numbers

- Range of values that can be stored in an integer depends on whether it is signed or unsigned

- Integers can also be flagged as short or long to modify their size in memory

[3:43:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=13410) Integer modifiers can be used to store positive and negative numbers, but only for integral data types.

- The 'short' modifier shortens the original size in memory for the data type it is applied on.

- The 'signed' modifier means that both positive and negative numbers can be stored.

- The 'unsigned' modifier means that only positive numbers can be stored.

- Using a modifier that is incompatible with the data type will result in a compiler error.

- The 'long' modifier increases the original size in memory for the data type it is applied on.

[3:55:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=14103) Floating points in C++

- Double has a precision of 15 and can handle larger numbers than float

- Scientific notation can be used to represent floating point numbers

- Floating point numbers are ultimately represented as ones and zeros in memory

[4:01:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=14465) Floating point numbers have limited precision

- The size of float is 4, double is 8, and long double is 16

- NaN stands for 'naughty number' and should be avoided in computations

- Suffixes should be used when initializing floating point numbers

- The precision of floating point numbers is limited, with floats having 7 digits and doubles having around 15

[4:12:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=15152) Learned about floating point numbers and their properties.

- Scientific notation used to make sense of really huge numbers.

- Playing with infinity and NaN.

- Introduction to booleans and their usage in decision making.

[4:18:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=15482) Booleans in C++ store two states, true or false, and take up one byte in memory.

- Booleans can be used to make decisions in C++ code.

- Printing boolean values can be confusing, but true is represented as one and false is represented as zero.

[4:29:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=16152) Working with character data types and the auto keyword in C++

- Characters occupy 1 byte in memory and can only have 256 different values

- Auto keyword is used to let the compiler deduce the type of a variable

- Examples of using auto with integers and suffixes to specify unsigned or long types

[4:35:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=16503) Auto type deduction in C++

- Visual Studio Code can deduce variable types before compiling

- Assignments can be made to auto-deducted variables, but be careful of illegal values

[4:46:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=17204) Learn basic arithmetic operations in C++

- Perform addition, subtraction, multiplication, and division on variables

- Understand integer division and the modulus operator

- Use the C++ standard library to format output

[4:52:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=17545) Learned about basic arithmetic operations on integers in C++

- Addition, subtraction, multiplication, division, and modulus operations were covered

- Modulus operator captures the remainder of division

- Valid only for integral types and not for floating points

- Precedence and associativity is an important concept to learn for operations in C++

[5:04:07](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=18247) Precedence table and parentheses can clarify order of operations

- Precedence table helps determine order of operations

- Parentheses can override precedence and clarify order of operations

[5:10:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=18605) Learned about precedence and associativity

- Use parentheses to make operations clear

- Don't rely too much on precedence tables

- Readability of code is important

- Learned about prefix and postfix increment and decrement

[5:21:25](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=19285) Learn about compound assignment operators.

- Compound assignment operators allow you to do an arithmetic operation and assign the result to a value in one go.

- The addition, subtraction, multiplication, division, and modulus operators all have a compound assignment version.

- Examples of compound assignment operators include value += 5, value -= 5, value \*= 2, value /= 30, and value %= 11.

[00:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26) C++ compound assignment operators and their usage

- Compound assignment operators allow for compact and efficient code

- Relational operators like less than, greater than, equal to, etc. can be used to compare numbers in C++

[5:39:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=20342) Learned about relational and logical operators in C++

- Relational operators can be stored in variables and used later in programs

- Logical operators include AND, OR, and NOT

- AND operator evaluates to true only if all operands are true

- OR operator evaluates to true if any operand is true

- NOT operator negates the operand

[5:45:05](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=20705) Logical operators can be combined to create complex expressions

- Use parentheses to ensure proper evaluation order

- Both relational and logical operators can be used together

- AND and OR operators can be used to combine expressions

[5:57:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21426) Learn about manipulators such as stdendl, sdd flush, and std set w.

- stdendl and \n both print out a new line character.

- sdd flush sends the output buffer to the terminal.

- std set w specifies a width for printed text.

- std set w can also be used for justification and field characters.

- std bool alpha forces boolean output to be true/false instead of 1/0.

- std showpos shows a positive sign for positive numbers.

[6:03:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21792) Manipulate and display data in different bases and formats using C++ manipulators

- Use std::showbase to show the base of the output

- Use std::uppercase to show the output in uppercase

- Use std::scientific and std::fixed to control how floating point data is shown

- Use std::setprecision to set the precision with which data is displayed

[6:14:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=22489) Learn how to format data in C++ using set width and justification.

- Data can be right justified, left justified, or internally justified.

- The default justification is right justified.

- You can specify a field character to fill empty spaces.

[6:20:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=22834) Modify boolean data display and output base system for terminal display

- Use std bull alpha to force output display in true/false format

- Set output base system using std deck, std hex or std act for different number systems

[00:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37) Documentation is crucial for learning.

- Numeric limits library helps understand types.

- Minimum, maximum, and lowest functions help understand ranges.

- Short, unsigned, long, double, and long double types are explored in the code example.

[6:38:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=23889) Learned about the limits library and math functions in C++

- Limits library provides a standardized way to carry various properties of arithmetic types

- Math functions include rounding, absolute value, trigonometry, and exponential functions

[6:49:47](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=24587) Learned about various math functions in C

- Explored the value of e and logarithmic functions

- Played with square root and rounding

[6:55:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=24935) Integral types can have implicit conversions during arithmetic operations

- Short and char variables occupy different amounts of memory

- Adding variables of different integral types can result in unexpected behavior

[7:07:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=25620) Learn conditional programming using logical operators in C++

- Use logical operators to achieve conditional programming

- Combine logical operators with conditional programming techniques for powerful results

[7:12:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=25950) Nested if statements example

- Demonstrating how nesting if statements work in C++.

- Example includes a traffic light simulation with the possibility of a police stop.

[7:23:28](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26608) Use else if statements to test for multiple conditions

- Initialize variables to store tools

- Set up a chain of if and else if statements to test for each tool

- Code after the last else if block will always execute

- Remember to use double equal signs to test for equality

[7:28:43](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=26923) Learn about the switch statement in C++

- The switch statement is a more compact way to test for conditions compared to else if clauses.

- The condition inside the parenthesis of switch statement can only be an integer or an enum.

[7:39:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=27566) Learned about switch statements and ternary operators

- Switch statements are used to catch different cases and handle them with one block

- Ternary operators are an alternative way to do tests with the if statement and make the code shorter

[7:45:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=27906) Learn how to use ternary operators in C++

- Initialize boolean variable 'fast' to use later in the code

- Use ternary operator to replace regular if statement for finding maximum value

[7:56:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=28582) Understanding how a loop works in C++

- Initialize iterator, run test, execute loop body, increment iterator, repeat until test fails

- Test controls how many times loop will start, can start from anywhere, iterators are common in C++ with type size\_t

[8:01:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=28908) Loops in C++

- Size t is used to represent sizes of things that can never be negative

- Hard coding values in loops is bad design and should be avoided

[8:12:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=29549) Learn about for loops and iterator scope

- For loops can be used to execute a block of code multiple times

- Iterator scope is limited to the body of the loop

[8:17:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=29875) Iterators can be declared inside or outside of a loop in C++

- Declaring an iterator inside the loop limits its scope to the body of the loop

- Declaring an iterator outside the loop makes it usable outside the loop body

- It is recommended to store loop values in a variable instead of hard coding

[8:28:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=30507) Using while loop to print 'I love C++' 10 times

- Initialize iterator outside loop

- Use iterator as test condition

- Print message and increment iterator inside loop

[8:33:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=30818) Learned about the syntax of a while loop and do-while loop

- While loop has 5 parts: iterator, starting point, test, incrementation, and loop body

- Do-while loop runs the body first and then does the test

- Validated loop syntax with a sample code

- Changed count to 100 and ran the loop successfully in rooster

[8:44:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=31448) Arrays are a way to group many variables together and manipulate them as a single unit in C++

- Arrays are constructed in C++ to group variables together and give them a name

- Arrays can be indexed using angle brackets and start at 0

[01:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86) Arrays in C++ can group variables under one entity

- The angle bracket syntax is used to read and write data in the array

- Arrays have clear boundaries and reading outside of them can cause issues

[9:00:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=32455) Writing and reading data in arrays using C++

- Data can be manually written into an array using the index and array name

- The loop can be used to write data into an array, making it easier to input data

[9:06:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=32778) Learn how to declare and initialize arrays in C++

- You can declare and initialize an array in place

- If you don't initialize all elements, the compiler initializes the rest to zero

- You can omit the size when declaring an array and the compiler will deduce it from the elements

[9:17:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=33434) Use std::size to get the size of an array at runtime

- Before C++17, we had to use a hack to get the size of an array at runtime

- Using std::size helps avoid hard-to-debug problems and makes the code more flexible

[9:22:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=33774) Use the std::size facility in the C++ standard library to get the count of elements in an array

- You can also divide the size of the array with the size of a single element to get the count

- Ranged-based for loops are convenient to work with arrays

- Arrays of characters can be printed to the console directly, but need a null termination string to indicate the end of the string

[9:34:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=34443) Properly terminate C strings for correct output

- A null terminator must be added to the end of the string

- C++ supports printing character arrays but without null terminator, output may not be as expected

- Two ways to set up a proper C string: manually adding null terminator or let the compiler add it

[9:39:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=34788) Working with character arrays and string literals in C++

- Character arrays can be initialized and printed directly with stdc out, but be careful about null termination

- String literals surrounded by double quotes are a more convenient way to store strings in a program

[9:51:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=35473) Don't go overbound, work within the legal boundaries of your array.

- Pointers are a special kind of variable that store addresses of other variables.

- Pointers and variables can work hand in hand, and we'll see how in the next few lectures.

[9:56:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=35805) Pointers store addresses to variables and should be used carefully

- Pointers can only store addresses to variables of the same type

- Pointers should not contain null pointers and all pointers on a system are of the same size

- Pointer variables can be initialized with the address of another variable and used to store values

[10:07:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=36465) Pointers and their usage in C

- Pointers can store data and addresses

- Cross-assigning between pointers of different types is not allowed

- Referencing is the act of reading through a pointer

[10:13:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=36809) Learned how to declare, initialize, and use pointers in C

- Pointers allow access to data stored in memory addresses

- Character pointers are special and can be used to treat strings as pointers

[10:24:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37498) C++ program memory map and virtual memory

- Programs are loaded into memory and stored in a special section called program area

- Virtual memory is used to trick programs into thinking they own all memory on the computer

[10:30:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=37855) Memory Management Unit maps program memory to real RAM for efficient use.

- The CPU only loads parts of the program that are likely to be used soon.

- The memory map is divided into sections such as stack and heap.

- Stack stores local variables and heap stores additional memory that can be used if the program runs out of stack memory.

- The memory map is defined by the operating system, and all programs on an operating system must conform to it.

[10:42:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=38534) Dynamic memory allocation in C++

- Memory allocated using new is stored on the heap and is not limited to a specific scope.

- Use delete to return memory to the operating system and reset the pointer to no ptr.

- Calling delete twice on a pointer can cause a crash.

- It is important to initialize pointers and reset them to no ptr after releasing memory.

[10:47:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=38869) Do not use or reference uninitialized memory in C++.

- Trying to write into uninitialized memory can cause crashes or other bad things to happen to your program, even before it completes running all the code you have designed.

- Allocating dynamic memory beforehand and then terminating the program before releasing that memory to the operating system can also cause issues.

[10:58:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=39516) Dynamic heap memory pointers in C++

- Pointers can be dynamically initialized at declaration

- Unused pointers should be reset to no pointer and not reused

- Calling delete twice on a pointer is bad practice

[11:03:57](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=39837) Avoid deleting memory twice in C++

- Initializing pointers when declared can prevent uninitialized pointers and null pointers can be used as placeholders

- Reset pointers to null after deleting memory to avoid using deleted memory

- Having multiple pointers pointing to the same memory location can cause problems if one of them deletes the memory

[08:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=486) Avoid using deleted memory in pointers

- Initializing pointers to null pointer can prevent crashes

- Always check if a pointer is null before using it

[11:20:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=40821) Avoiding problems with dangling pointers

- We can avoid dangling pointers by resetting deleted pointers to no ptr

- We can also avoid problems with multiple pointers pointing to the same address by designating one as the master pointer

[11:31:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41473) Prevent program crashes with two options: exception mechanism and std no throw

- Use try and catch blocks to handle potentially offending code

- Pass std no throw parameter to new operator and check for null pointer

[11:36:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41795) Learn about null pointer safety and how to handle it.

- Null pointer can cause program to crash, so we need to check if a pointer contains valid address before using it.

- We can use try-catch blocks or std no throw option to handle null pointer, and it is safe to call delete on a pointer containing null pointer.

[11:47:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=42453) Avoid Memory Leaks

- Memory leaks lead to loss of access to memory and can cause programs to crash.

- Be careful with dynamic memory allocation and make sure to release memory when it is no longer in use.

[11:53:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=42798) Avoid memory leaks in C++ programs

- Leaked memory can lead to program errors and inefficiencies

- Dynamic memory allocated on the heap can be deleted using the 'delete' keyword

- Dynamically allocated arrays can be created using the 'new' operator

[12:04:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=43455) Exploring the differences between static and dynamic arrays

- Static arrays are allocated on the stack while dynamic arrays are allocated on the heap

- Static arrays can be looped through using a range-based for loop, while dynamic arrays cannot be looped through in the same way

[01:17](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77) Dynamic arrays and references in C++

- Dynamic arrays can be stored on the heap and have some limitations compared to static arrays.

- References are aliases to variables and can be used like the original variable in C++ code.

[12:20:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=44438) References and pointers have key differences in syntax and usage.

- References allow direct access to original variables without dereferencing.

- Pointers can be changed to point to different variables, while references cannot.

[12:26:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=44761) References and pointers can be used to read and write variables, but references cannot be reassigned to reference a different location in memory.

- Syntaxes for reading and writing variables through references and pointers are different, but both can achieve the same result.

- References have benefits over pointers, but some people may find their syntax to be ugly.

- Attempting to reassign a reference to reference a different location in memory will result in an error.

[12:36:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=45418) Const keyword creates a read-only reference

- References can be modified without const keyword

- Const keyword applies to the reference variable name

[12:42:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=45746) Simulating reference behavior with pointers

- Const pointers can be used to simulate reference behavior in C++

- Modifying data through a const pointer will result in a compiler error

[12:53:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=46398) Manipulating character cases in C++

- Using std::toupper() to convert characters to uppercase

- Using std::tolower() to convert characters to lowercase

- Using std::isalnum() to check if a character is alphanumeric

- Using std::isalpha() to check if a character is alphabetic

[12:58:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=46720) Try different compilers to learn about facilities from cctype

- Remove rooster.exe and run task to build using msvc

- Example to count blank characters and check if character is uppercase or lowercase

[13:09:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=47346) Learn to manipulate C strings using the C++ standard library

- Use strlan to check the length of a string

- Use strcmp to compare strings, which returns negative value if the first string comes before the second and vice versa

- Use another version of strcmp to specify the number of characters to compare

[13:14:20](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=47660) The sdrchr facility helps find characters in a string.

- The facility returns a pointer to the first occurrence of a character in a string if found, and a null pointer if not found.

- It can also help find the last occurrence of a character in a string.

[13:25:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=48300) Function to compare strings based on specified number of characters.

- Cannot change where an array points.

- Example code provided for comparing strings with specified number of characters.

[13:30:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=48614) Using std::strchr to search for a character in a string

- std::strchr returns a pointer to the first occurrence of the target character in the string

- If the target character is not found, std::strchr returns a null pointer

[13:40:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=49213) Learn how to concatenate strings in C++

- Use stdstrcat to join two static arrays

- Check the length of your destination string before concatenation

[00:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=21) Learn how to concatenate and copy strings in C using strcat and strcpy functions.

- The strcat function is used to concatenate two strings, while the strcpy function is used to copy one string to another.

- Make sure the destination string is large enough to contain the concatenated or copied string, and ensure that both strings are null-terminated.

- In Visual Studio Code, include the cstring header file and use strcat and strcpy functions to concatenate and copy strings.

[13:55:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=50158) New version of str cat allows specifying character count

- Documentation for stdstrcat can be found on cppreference

- sdr cpy function copies from source to destination

- std str cpy with n parameter allows specifying character count

[14:01:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=50483) std string is a high level type that simplifies working with string data in C++

- std string hides details like keeping track of array bounds and null characters

- std string can be declared and initialized just like any other variable in C++

[14:11:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=51110) Avoid wasting memory by using std strength

- Initial memory allocated to store planet is returned to operand system

- New memory is allocated to contain new thing

[14:17:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=51432) The one definition rule states that freestanding variables and functions cannot have multiple definitions.

- Multiple definitions of a freestanding variable or function will result in a compiler error.

- In the context of classes, multiple definitions are allowed as long as they are in different translation units.

- Static member variables in a class can also cause issues with the one definition rule.

[14:27:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=52060) Functions in C++

- Functions are reusable pieces of code that can take input, do some processing, and give output.

- Functions in C++ are declared with a return type, function name, and parameter list, and are called by specifying the function name and passing in arguments.

[14:33:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=52392) Functions in C++ can take parameters and return values

- Parameters are input values passed into the function

- Return values are the output values returned by the function

[14:43:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53034) Learn how to declare and call functions in C++.

- Functions are identified by their name and parameters.

- Functions can be called multiple times and can have return values.

[14:49:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53355) Demonstration of defining and calling functions in C++

- Example of defining a maximum function with arguments and return value

- Example of defining a hello function with no arguments and no return value

- Example of defining a lucky number function with no arguments and return value

[14:59:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=53976) Functions can be split into declaration and definition to provide a summary of what the function does

- The function header or prototype is made up of the return type, function name, and parameters, followed by a semicolon

- The function definition contains the details of how the function works and is enclosed in curly braces

- The function prototype needs to come before wherever it is called

- A function definition can also double as a function declaration

[15:04:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=54298) Create a declaration and definition for the maximum function in C++

- The declaration includes the return type, name, and parameter list

- The definition includes the body of the function

[15:15:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=54936) Program compilation model expanded to preprocessing, compilation, and linking

- Preprocessing stage includes looking for include statements and replacing them with the content of the included file

- Compilation generates object files, one for each translation unit

- Linking stitches object files together to form a binary executable

- Program can be split across multiple files using header and implementation files

[15:21:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=55274) Splitting Functions Across Multiple Files in C++

- Functions defined in a header file should be declared in a .cpp file.

- The linker parses the entire project to find function definitions.

- Splitting functions across multiple files requires a specific structure in C++.

- Visual Studio Code can be used to create and organize multiple files for a project.

[15:32:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=55946) Split functions across multiple files for organizational purposes.

- Header files contain declarations while cpp files contain definitions.

- Definitions can be located in any cpp file as per project requirement.

- Function 'anchorMalt' can be split into a declaration and definition in separate files 'operations.h' and 'operations.cpp'.

[15:37:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=56276) Splitting functions across multiple files for project organization

- Functions can be split across multiple files to keep main file clean

- Header files can be moved to separate file for easier management

- Passing parameters to functions by value is a common practice

[15:48:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=56909) Pass parameters by pointer to avoid copying

- Passing by pointer allows modifying original variable

- Declare functions and definitions with pointer parameter syntax

[15:54:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=57241) Passing parameters by pointer and reference in C++

- Passing by pointer requires passing the address of the variable and using the referencing operator to modify the value

- Passing by reference avoids copying and allows modification of the value directly

[15:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=911) Using function parameters to output data from the function.

- Output parameters should be passed in a way that you can modify the arguments from inside the functions.

- We can use either references or pointers to achieve this.

- Input parameters shouldn't be modified; they are meant to get data in the function.

- The function is set up to compute the maximum of two input strings.

- The output parameter is used to get output outside the function.

- The changes made inside the function are visible on the outside because we are working on the original variable through the reference which is really a true alias.

[16:10:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=58244) Learn how to use references and pointers to make changes in function stick and be visible on the outside

- Passing the output parameter as a reference allows changes to be visible in outside variables

- Using pointers in functions allows changes to be made and visible on the outside

[16:21:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=58901) Returning by value creates copies of local variables.

- The compiler does some magic to avoid unnecessary copies and return by reference.

- However, relying on returning by value can lead to unexpected behavior.

[16:27:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=59229) Returning by value can lead to unexpected optimization by the compiler.

- Compiler can optimize return by value to return by reference, reusing memory addresses.

- Don't rely on returning by value and making copies in your code.

[16:38:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=59892) Functions with the same name can be set up to take different parameters.

- The return type alone does not make two functions different.

- Parameters can be different in type or order to create valid overloads.

[16:43:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=60220) You can set up all kinds of overloads for your functions in C++

- Overloads can be based on differences in parameter types, order of parameters, and number of parameters

- Implicit conversions can be used to call functions with different parameter types

- Valid overloads require parameters to be different in some way

[16:54:31](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=60871) Lambda functions can be declared with a capture list, parameter list, and body.

- The body of a lambda function can contain any valid C++ code.

- Lambda functions can also have a return type specified with the -> symbol.

[19:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1182) Lambda functions in C++

- Lambda functions can be assigned to a variable using auto type deduction

- Lambda functions can be called directly without assigning to a variable

- Lambda functions can take parameters and perform operations on them

[17:10:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=61808) How to set up a lambda function that returns something

- The lambda function can be named and called repeatedly

- The return type of the lambda function can be specified explicitly

- The function can take parameters and return a specific type

[17:15:30](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=62130) Exploring lambda functions in C++

- Lambda functions can be declared and called directly using syntax

- Parameters can be specified and used in lambda functions

- Return type can be specified explicitly or deduced automatically

- Capture lists in lambda functions can be used for useful tasks

[17:26:10](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=62770) Learn how to capture variables in lambda functions.

- Variables used in lambda functions must be captured in the capture list.

- Variables can be captured by value or reference.

[17:31:31](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=63091) Lambda functions can capture variables by value or reference

- To capture by value, list the variables directly in the capture list

- To capture by reference, add an ampersand symbol in front of the variable in the capture list

[17:42:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=63732) Function templates are just blueprints that get generated by the compiler based on the arguments you call your function with.

- The blueprint is set up using a syntax that includes a template type name and a placeholder for the types used in the function.

- The compiler generates actual functions based on what you call in your C++ code.

- Function templates help to avoid several overloads for the same function, but only the overloads that are called in your code are generated by the compiler.

- Once a function template instance is generated, the compiler can reuse it if it is needed again.

[17:47:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=64043) Function templates in C++ allow for flexible, efficient code.

- Templates allow for a single point of control, generating actual C++ functions based on arguments.

- Using function templates eliminates the need for copying and pasting code for different argument types.

[17:58:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=64706) Function templates are blueprints used to generate actual functions and can avoid multiple function overloads.

- The logic inside a function template must be supported by the types passed to the function.

- Multiplying two strings does not make sense and will result in a compiler error.

[18:03:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65016) Use compiler tools like cpp insights to understand errors in function templates.

- Compiler tools can help in understanding errors in function templates.

- Make sure that the template instances generated by the compiler support the operations in the function template.

[18:14:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65681) Function templates may not always behave as expected

- Template instances can produce unexpected results if not used carefully

- Arguments passed to a function template must support the operations used

- Tools like cpp insights can help identify issues

- Next lecture will cover template type deduction and explicit arguments

[18:19:53](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=65993) Template type deduction and explicit arguments

- The compiler deduces the type it would use to set up a template instance from the arguments passed to the function.

- Automatic template type deduction can have problems when passing parameters of different types.

- Explicit template arguments can be used to force the compiler to use a specific type to generate the template instance.

[18:30:46](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=66646) Using explicit template arguments allows passing different types for template arguments.

- Compiler generates template instance of specified type.

- Implicit conversions are done for arguments passed.

[18:36:23](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=66983) Passing template parameters by reference

- Function templates can be separated into declaration and definition

- Passing by reference can cause confusion and compiler errors

- Addresses of variables inside function templates are different from the outside

[18:47:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=67631) Function template specialization allows bypassing the default mechanism for passing arguments.

- Passing pointers to the string can cause issues with the default mechanism.

- Template specialization is achieved by using the template keyword and specifying an explicit template argument.

- A specialized function can use a built-in function from the C++ standard library.

[18:52:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=67952) Template specialization can be useful when working with pointers

- The maximum function template returns whichever value is greater between two parameters

- The function can be specialized to work with different data types

- When working with c-strings, template specialization is necessary to avoid comparing pointers instead of string data

[19:03:18](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=68598) Template specialization and concepts in C++

- Template specialization allows for custom implementations of function templates for specific types

- Concepts enforce constraints on template parameters for safer function templates

[19:08:53](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=68933) Using concepts to constrain function templates

- Concepts can be enforced using the 'requires' clause after template declaration

- Concepts can also be specified directly in template declaration using the concept keyword

- Auto keyword can also be used to specify concepts for function templates

- Multiple syntaxes provide flexibility in using concepts in C++ code

[19:20:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=69606) C++ 20 provides multiple syntaxes for using concepts in function templates.

- Syntax 1: Use a requires-clause to specify the constraints on the function template.

- Syntax 2: Use the auto keyword to deduce the return type and constraints on the function template.

- Syntax 3: Use the concept keyword to specify the constraints on the function template.

- The integral and floating-point concepts are some of the concepts provided by C++ 20.

- You can also create your own concepts in C++ 20.

[19:25:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=69938) Create custom concepts in C++ using type traits.

- Use the 'concept' keyword, followed by a name and requirements to constrain template parameters.

- Use a 'requires' clause for multiple statements in the concept.

[19:36:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=70616) Creating and using C++ concepts for type constraints

- Syntax for creating concepts using type traits or requirements

- Concepts only check for syntax, not value

- Examples of using concepts to constrain function templates

[19:42:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=70952) Learn about types of requirements in requires close

- Simple requirement checks syntax of an expression

- Nested requirement enforces expression to be true

- Compound requirement checks return type of an expression

[19:53:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71586) Enforcing requirements in C++ using concepts.

- Nested requirements can be used to enforce expression requirements.

- Compound requirements can be used to enforce expression and value requirements.

[19:58:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71889) Combining concepts using logical operators

- Concepts can be combined using the OR and AND operators

- A function template can have multiple concepts applied to it

- Examples of combining concepts using OR and AND operators

[20:08:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=72530) Enforcing concepts on variables and functions using the auto keyword

- Auto keyword with concepts can be used to constrain variables to a specific type

- Functions can also be constrained using auto keyword with concepts

[20:13:57](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=72837) Learn about classes in C++ to build your own types and use them in your programs.

- Classes are a mechanism in C++ to build our own types.

- Classes have member variables that model properties and behaviors/functions that do things on the class.

- You can use classes to model real-world objects and use them in your programs.

- Classes can be defined using the keyword 'class', the name of the class, and a pair of curly braces.

- Inside the class, member variables can be defined to model properties of the class.

- Behaviors or functions can also be defined inside the class to do things on the class.

- A cylinder can be modeled using a class in C++, with its base radius and height as member variables.

- The area and volume of the cylinder can be computed using the member variables and formulas.

- Classes can come in handy when modeling complex objects and keeping track of their properties and behaviors.

[20:25:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=73503) Create a class to compute the volume of a cylinder.

- Declare and initialize variables for base radius and height.

- Define a function 'volume' to compute the volume using the formula pi \* radius^2 \* height.

- Make the 'volume' function public to access it outside the class.

- Build and run the program to verify the output.

[20:30:26](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=73826) Declaring a class in C++

- Member variables are private by default, but can be made public using the public keyword

- Functions or methods are behaviors of the class and can be used to make objects do things

[20:41:28](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=74488) Creating a cylinder object using constructor with no parameter

- Member variables should be private to avoid exposing them to main function

- Constructor with no parameter can be used to initialize member variables

- Using debugging tools like gcc can help in understanding how constructor works

[20:47:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=74821) Constructors are used by the compiler to build objects

- Constructors can be used to set up parameters for objects

- The compiler generates an empty constructor if one is not defined

[20:58:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=75486) Constructors must be public to be usable outside of the class

- Private constructors cannot be called outside of the class, resulting in build errors

- Getters and setters are methods used to read and modify member variables from outside the class

- Getter and setter methods must be public to be accessible from outside the class

[13:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=809) Using getters and setters to modify member variables in a class

- Getters allow access to private member variables by returning a copy

- Setters allow modification of private member variables through function calls

[21:14:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76443) Separate class definitions into header files for better organization

- Create a separate header file for class definitions

- Include the header file in the main file for use in the program

[21:19:09](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76749) Splitting code into multiple files and using include guards

- Use include guards to prevent duplicate code inclusion

- Splitting code into header and implementation files makes it easier to manage

- Header files should contain class definitions, while implementation files should contain implementation details

- Include necessary header files in implementation files

[21:30:00](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77400) Manage class objects through pointers

- Use pointers to access class objects directly

- Allocate objects on heap using new operator and release memory using delete operator

[21:35:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=77740) Managing objects through pointers in C++

- Using the dereference operator and star operator to access objects

- Creating objects on the heap using the new operator and managing memory with delete

[21:46:44](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=78404) Learn about constructors and destructors in C++

- Use Visual Studio Code to play with C++ code

- Constructors allocate memory and initialize variables

- Destructors release memory when objects die

[21:52:07](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=78727) Learn about C++ destructors

- Constructors and destructors are used to manage memory in C++

- The destructor releases memory allocated by the constructor

- Destructors are always called when objects are destroyed

[22:02:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=79357) Always explicitly release memory allocated through new operator to avoid memory leaks.

- Use delete keyword to release memory.

- Destructor is a special method that is called when an object dies, is passed by value, or goes out of scope.

[22:07:58](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=79678) Learn about the this pointer in C++

- The this pointer is a special pointer maintained by C++ to manipulate the current object

- One use case is to print the memory address of the object using the this pointer

- Another use case is to resolve conflicts between member variables and function parameters

[22:18:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=80335) Using the 'this' pointer to manipulate addresses in functions for a class

- Using parameters with the same name as member variables can cause confusion

- Compiler assigns the value of the variable to itself using the syntax 'name=name'

- Assigning values using the 'this' pointer is a more reliable method

- Encouraged to play with code and test different methods

[22:24:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=80651) Using the 'this' pointer to set member variables

- Setters must go through the 'this' pointer to affect member variables

- Chained calls can be made using pointers returned by setters

[22:34:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=81261) Structs have public member variables by default

- Classes have private member variables by default

- We can change the defaults by defining public and private sections inside the class

[22:39:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=81551) Classes vs Structs

- Classes and structs are the same, except for their defaults

- Structs are useful for classes with only public member variables

[22:49:38](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=82178) Size of an object is determined by the sum of sizes of its member variables.

- Pointers are accounted for in the size of an object, not what they are pointing to.

- C++ inheritance allows building hierarchies of classes that depend on other classes.

[22:54:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=82489) Object oriented programming with inheritance in C++

- Classes and objects form the basic defining feature

- Inheritance allows building new classes in terms of pre-existing classes

- Inheritance creates an inheritance tree from most fundamental to derived class

- Example: Building a player class derived from a person class

[23:05:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=83134) Inheriting from person, the player class has a person part inside of it.

- Player objects are built with a string parameter to initialize the member variable.

- Player objects inherit first name and last name from person, but private members of person cannot be accessed from player objects.

- Public methods can be used to access private parts of the person class from player objects.

- The stream output operator in player uses getters to access the names of the player object.

- A string view parameter is used in the stream output operator to avoid passing a temporary object by reference.

- The program is built with GCC and the default destructor generated by the compiler is used.

- The default data for first name and last name in person is 'mysterious' and 'person', respectively.

[01:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=71) Understanding public inheritance in C++

- Private parts of the base class are not accessible in the derived class

- Using getter member functions from the base class is the correct way to access member variables

[23:21:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=84095) Protected member variables allow for access and modification from derived classes in public inheritance

- Changing the access specifier to protected in the base class enables this behavior

- Protected members are still not accessible from outside the class

- This can be useful in setting up constructors and modifying member variables in derived classes

[23:27:08](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=84428) Using protected member variables in base class for easy data access in derived class constructors.

- By setting member variables as protected in the base class, they become accessible and usable in any derived class that does public inheritance.

- However, they are not accessible from outside the class.

- We can use this mechanism to easily pass data from derived class constructors back to the base class.

- In the next lecture, we will learn about base class access specifiers and how they relate to inheritance.

[23:38:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=85081) Private inheritance is the highest level of inheritance in C++

- Anything that was private in the base class is private in the derived class

- Private inheritance makes all inherited members private to the derived class

[23:43:19](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=85399) Creating a Player class that inherits from Person

- Player class inherits publicly from Person class

- Public members in Person class stay public in Player class, protected members stay protected, and private members stay private

- Accessing private data from Person object is not possible in Player class

- We can access public and protected data from Person object using getters in Player object

[23:53:46](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86026) Protected inheritance makes public things private and protected things private in derived classes

- Inherited protected members can be accessed from within the derived class

- Private members of the base class cannot be accessed from the derived class

- Attempting to access private or protected members from outside the class or derived class will result in a compiler error

[23:59:15](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=86355) Engineer class inherits private variables from base class

- Output stream operator still works as a friend of the Engineer class

- Attempting to access private variables from outside the Engineer class will result in compiler errors

[00:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=1) Private inheritance makes all inherited member variables and functions private to the inheriting class.

- In this setup, even if a civil engineer inherits from a public inheritance, they won't have access to the private member variables.

- All member variables inherited from the parent class will become private to the child class, except for those already private in the parent class.

[24:15:40](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=87340) Private inheritance strips down access to public and protected members in upstream class

- Compiler errors occur when trying to access private members from a subclass

- Output stream operator can't access private members even if it's a friend of the subclass

[24:26:52](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=88012) Private inheritance in C++ strips down access to private level access in an inheritance class

- Inheriting classes won't have access to private member variables. Using the 'using' keyword can resurrect them to a more relaxed access level.

- Resurrected members cannot be something that is already private to an upstream class.

- Using private inheritance can make code harder to read and understand. It's better to use public or protected access levels instead.

[24:32:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=88357) Resurrected members take the access level of the section where they are resurrected

- Using the 'using' feature, we can resurrect members to have protected or public access levels in downstream classes

- Overloads with the same name for a member function can be shared by many overloads

- Privately inheriting from a class will strip its members down to private level access in the derived class

[24:43:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89013) Private inheritance can cause accessibility issues in downstream classes

- Attempting to access private members from a base class in a derived class will result in compiler errors

- Constructors are automatically called for base classes, and can be customized for derived classes

[24:49:11](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89351) Always provide default constructors for your classes.

- The compiler may call these default constructors in unexpected ways, especially if your class is part of an inheritance hierarchy.

- The most base part of your class is going to be built first.

[24:59:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=89982) Custom constructors are needed to forward information when building objects with inheritance

- Default constructors cannot initialize member variables that belong to base classes

- Initializer lists can be used with constructors to forward information from derived classes to base classes

[05:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=322) Adding constructors to person and engineer classes

- Created a constructor for the engineer class that initializes member variables and calls the constructor for the person class

- Added a constructor for the civil engineer class that also initializes member variables and calls the constructor for the engineer class

[25:16:04](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=90964) Custom constructors can be used to set up objects in layers

- Person, Engineer, and CivilEngineer custom constructors are called in that order when setting up a CivilEngineer object

- Initializing member variables in the body of a class can cause errors and should be avoided

[25:21:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=91293) Custom constructors in inheritance hierarchy

- Setting up custom constructors can prevent weird compiler errors

- Calling base constructors in the correct order is important

- Debugging can help verify the information in the object

[25:32:03](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=91923) Compiler strips off engineer information when initializing a person object

- Copy constructors are automatically generated but may need to be implemented for dynamic memory allocation

- Custom copy constructors can be set up using the syntax const className& source

[25:37:34](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=92254) Implementing copy constructor for Engineer

- Copy constructor for Person is not being called, resulting in default initialization of data

- Alternative approach is to call Person's copy constructor directly and pass the source object

[47:39](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2859) Demonstration of Custom Copy Constructor in C++

- Custom copy constructor allows for reusing code in inheritance hierarchy

- Copy constructor can be called from base class to avoid temporary copies

- Inheriting base constructors is possible in C++ to set up derived class objects

[25:53:17](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=93197) Inheriting constructors initializes only the base member variables

- Copy constructors are not inheritable and need to be set up explicitly if required

- Access specifier of the inherited constructor is based on the base class

- Setting up own constructors on top of the inherited constructors is possible

- Inheriting constructors can make the code confusing and is not recommended

[13:24](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=804) Inheritance and destructors call structures in reverse order to constructors

- Constructors are called in order from base to most specialized, while destructors are called in reverse

- Most specialized destructor is called first, followed by destruction of base parts

[26:08:42](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=94122) Understanding the order of calling constructors and destructors

- Constructors and destructors are called in reverse order starting from the most specialized class to the base class

- Reuse of names in inheritance hierarchy is possible, but the method called depends on the object type

[26:19:13](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=94753) Polymorphism allows for managing derived objects using base class pointers or references.

- This can simplify code and make it more flexible.

- Polymorphism is useful for tasks such as drawing objects in an inheritance hierarchy.

[26:24:33](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=95073) Polymorphism allows us to store different types of objects in an array

- An array can only store objects of the same type

- Polymorphism allows us to store base class pointers in an array and use them to store objects of different derived classes

- Polymorphism allows a base class pointer to take multiple forms and manage multiple kinds of objects in our program

- We can use polymorphism to manage objects in our inheritance hierarchy, such as circles and novels

- Polymorphism is a powerful feature in C++ and is not achieved by default

[26:35:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=95736) Polymorphism in C++ solves looping and function problems

- Polymorphism allows us to set up one draw method for all shapes

- Polymorphism allows us to set up one collection to manage all shapes

[00:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=41) Classes Shape, Oval, and Circle were successfully implemented and integrated

- The implementation of the classes was done correctly and the program was successfully compiled using GCC

- The program successfully created objects of the Shape, Oval, and Circle classes and printed their information using the draw method

- The program also demonstrated the use of a Shape pointer to manage data through base pointers or base references

[26:51:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=96687) Static binding in C++ inheritance hierarchy can lead to bad design

- Setting up multiple drawing functions for each shape is impractical

- Setting up different collections for each shape type is also bad design

- Dynamic binding can be achieved by marking the methods as virtual

- Use of virtual keyword in front of functions allows for correct method resolution

- Virtual functions should be added to all relevant classes

[26:56:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97010) Marking draw method as virtual enables dynamic binding

- By making draw method virtual, compiler knows to use dynamic binding instead of static binding

- Dynamic binding allows compiler to look at actual object type instead of just the base pointer type

- This enables calling the correct most specific method based on the actual object type

[27:07:35](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97655) Dynamic binding allows for easy management of different shapes in C++

- Using references and virtual functions, we can call methods polymorphically

- However, non-virtual methods will result in static binding and compiler errors

- Using a base pointer, we can set up an array to handle any shape in our inheritance hierarchy

[27:12:52](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=97972) Achieving dynamic binding using virtual functions

- Virtual functions allow for polymorphism and late binding

- Dynamic binding has a memory cost due to virtual tables

- Object slicing occurs when assigning a derived object to a base object

[27:24:12](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=98652) Using base references or pointers is necessary for dynamic polymorphism.

- Without base references or pointers, the compiler will slice off derived class information.

- Storing derived objects in spots designed for base objects will also result in slicing.

- Slicing results in only base class information being stored, so calling specific derived class methods will not work.

[27:29:43](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=98983) Be careful when storing derived objects in collections

- Storing derived objects in collections designed for base class data will slice off polymorphic data permanently

- Storing references in collections is not allowed due to the left assignability rule

- Storing pointers, whether raw or smart, is allowed and works with polymorphism

[11:06](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=666) Storing derived objects in an array designed for base class data leads to slicing

- References cannot be stored in collections like arrays

- Polymorphism through virtual functions works with base pointers managing derived objects

- Using smart pointers like unique or shared pointers will work

- The override mechanism in C++ helps avoid errors in inheritance hierarchies

[00:02](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=2) Use of 'override' keyword can help avoid polymorphic behavior issues

- Without 'override', typos in method setup can lead to separate methods being created

- Adding 'override' specification enforces method overriding from parent class

[27:57:59](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=100679) Understanding overloading and overriding in C++

- In C++, overloading and overriding can affect the availability of inherited functions

- It is important to carefully consider the overloads and overrides in inheritance hierarchy to ensure availability in downstream classes

[28:03:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=100994) Polymorphism must be declared at the top level class

- Methods must be known at the shape level for shape polymorphism to work

- Polymorphism can be set up at different levels in the inheritance hierarchy

[28:14:22](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=101662) Animal class hierarchy with virtual methods

- Animal class is the top level class with virtual breathe method

- Feline, Dog and Cat classes inherit from Animal and have their own unique methods

[01:16](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=76) Achieving animal polymorphism through inheritance hierarchy

- Inheriting bird class from animal class and changing parameter types to std string view

- Implementing animal polymorphism by calling most specific breathe method through base animal pointer

[28:29:56](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=102596) Polymorphism and inheritance in C++

- Polymorphism can be set up at different levels in the inheritance hierarchy

- Static members can be inherited and accessed from derived classes

[28:34:55](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=102895) Understanding polymorphism with static member variables

- Static variables are shared among all instances of a class

- Inheriting a static variable means all derived classes share the same variable

- To maintain separate static variables for each class, redefine the variable in each class

- Polymorphism can be used to call the most specific method and print the correct count

[28:44:48](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=103488) Use of static member variables with inheritance and polymorphism

- Setting up member variable at the ellipse level to solve the problem

- Incrementing to get the correct count

- Initializing the static variable at the ellipse level

- Using shape polymorphism to call the correct getcount method

- Learning about the final specifier used in inheritance hierarchies

[28:49:32](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=103772) Restrict overriding and inheritance using final specifier

- Use final specifier to mark a virtual method to restrict override in downstream classes

- Use final specifier to mark a class to restrict inheritance altogether

- Conflicting ideas: introducing virtual method in a final class

- Overriding in a final class is allowed

[29:00:16](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=104416) Final classes cannot be inherited in C++

- Marking a class as final prevents inheritance and leads to compiler errors

- Virtual methods can be marked as final, preventing further specialization or overriding

[29:05:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=104721) Using default arguments with virtual functions can lead to unexpected results with polymorphism.

- Default arguments are handled at compile time, while virtual functions are called at runtime with polymorphism.

- Static binding is used to decide which default parameters to pass to the function, but the actual function called is decided by dynamic binding.

- It is recommended to avoid using default arguments with virtual functions to make the code easier to follow and understand.

[29:15:51](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=105351) Polymorphism with default parameters in C++

- When going through a base reference, the most specialized virtual function override is called but it uses default parameters from the base class.

- When manipulating a derived object directly, the derived version of the function is called and it uses default arguments from the derived class.

- Assigning a derived object to a base object directly causes slicing and the base version of the function is called with default parameters from the base class.

[29:21:01](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=105661) Avoid using default arguments or default parameters with polymorphism

- The compiler uses static binding and does not do dynamic binding with default parameters

- The order of destructors called is from specialized to base implementation

- Using a base pointer to manage a derived object may result in incorrect destructor calls

[29:31:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=106281) Mark destructors as virtual in inheritance hierarchy

- Undefined behavior if destructor not called for all levels of inheritance

- Virtual functions and virtual destructors ensure proper memory release

- Dynamic casts can facilitate downstream transformations between polymorphic types

[01:27](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=87) Transforming from a base pointer to a derived pointer using dynamic\_cast

- Dynamic\_cast allows us to transform from a base class pointer or reference to a derived class pointer or reference at runtime.

- This transformation enables us to call non-polymorphic functions on derived pointers or references.

- The syntax for dynamic\_cast is straightforward: specify the input and output types within angle brackets.

- The transformation can fail if the object being pointed at does not have the desired derived class information.

[29:46:45](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=107205) Non-virtual function cannot be called through a base pointer

- Function needs to be virtual and polymorphic for this to work

- Dynamic cast can be used to transform base pointer to derived object

[29:51:50](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=107510) Transforming base pointer into derived pointer

- Dynamic casts can fail and we need to do some checks before calling methods.

- We can use a pointer transformation to check for valid data and avoid runtime crashes.

[30:01:37](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108097) Dynamic casts work with polymorphic inheritance hierarchies

- Dynamic casts can transform from a base pointer to a derived pointer or to transform from a base reference to a derived reference

- Transforming to a derived reference has a drawback in that you don't have a way to check and see if the transformation was successful

- Dynamic casts are only going to work with polymorphic inheritance hierarchies

[30:06:41](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108401) Dynamic casts are meant to work only with pointers or references

- Attempting to cast between pointers and references using dynamic casts will result in undefined behavior or compiler errors

- Virtual functions should never be called from constructors or destructors, and we will explore why in the next lecture

- The order in which constructors and destructors are called is important when dealing with inheritance and virtual functions

[30:16:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=108996) Polymorphism not working due to calling virtual function before derived object is constructed

- Calling a virtual function from a constructor or destructor calls the best version of the function before the derived object is constructed or after it is destroyed

- The best version of the function is determined at compile time resulting in static binding behavior instead of polymorphic behavior

[30:21:49](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=109309) Never call virtual functions from constructors or destructors

- Doing so can result in static binding results instead of dynamic binding behavior

- To achieve dynamic binding behavior, call setup and cleanup functions after the object has been properly constructed or before it gets destroyed

[30:32:21](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=109941) Abstract classes can't be instantiated

- Classes with pure virtual functions are abstract

- Base pointers can be used to manage derived objects

- Dynamic type of a base pointer can be determined using type id

[30:37:29](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=110249) Demonstration of polymorphism in C++

- Base pointers can manage objects of derived classes polymorphically

- Pure virtual functions in a base class make it an abstract class and must be implemented by downstream inheritance classes

[30:48:36](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=110916) Interfaces in C++ are super powerful.

- By attaching an interface to a type, we can inherit all the features of the interface.

- We can even attach the interface to a full inheritance hierarchy.

[30:53:54](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=111234) Implement stream insertable interface to enable polymorphism in printing objects

- Polymorphism allows calling stream insert method on different objects to print them out in their desired format

- Attaching stream insertable interface to point class enables printing point objects using output stream operator

[31:04:14](https://www.youtube.com/watch?v=8jLOx1hD3\_o&t=111854) Introduction to interfaces in C++

- Interfaces can be powerful in C++ design

- Attaching an interface to a type gives it the powers that come with that interface

- Deriving classes need to implement virtual functions of the interface to use it polymorphically