***Install CLion by JetBrains* (If students already have JetBrains account or have used IntelliJ Ultimate just sign-in or Link account)**

* **Use existing account for jet brain**
* **Download appropriate**
* **Register free student licensing using school email**
* **Some might need to install compiler and debugger using MingW64 (use same version as VScode for future purposes** [**https://code.visualstudio.com/docs/cpp/config-mingw**](https://code.visualstudio.com/docs/cpp/config-mingw) **windows must edit environment**
* **Activate licensing within CLion prompt**
* **Test run hello world program**
* **Download Jimmy’s C++ Code package (Most likely hosted on ACM website or Google Drive or ACM Club Discord)**

***Learning Objective and Syllabus***

**The sequence of learning C++ typically follows a structured path, starting with the basics and gradually moving on to more advanced topics. Here's a suggested sequence:**

**1. \*\*Basics:\*\***

**- Variables: Learn about different data types (integers, floating-point numbers, characters, etc.) and how to declare and use variables.**

**- Operators: Understand basic arithmetic, relational, and logical operators.**

**- Control Flow: Learn about if statements, loops (for, while, do-while), and switch statements.**

**2. \*\*Functions:\*\***

**- Define and use functions.**

**- Parameters and return values.**

**- Function overloading.**

**3. \*\*Arrays:\*\***

**- Declare and initialize arrays.**

**- Accessing elements and array manipulation.**

**- Multidimensional arrays.**

**4. \*\*Pointers:\*\***

**- Basics of pointers and addresses.**

**- Pointer arithmetic.**

**- Dynamic memory allocation and deallocation (new and delete operators).**

**5. \*\*Strings:\*\***

**- C-style strings (character arrays).**

**- `std::string` class and its methods.**

**6. \*\*Object-Oriented Programming (OOP):\*\***

**- Classes and objects.**

**- Constructors and destructors.**

**- Inheritance, polymorphism, encapsulation, and abstraction.**

**7. \*\*Standard Template Library (STL):\*\***

**- Vectors: Dynamic arrays with useful methods.**

**- Iterators: Generalized pointers for traversing containers.**

**- Algorithms: Use algorithms provided by the STL.**

**8. \*\*File I/O:\*\***

**- Reading from and writing to files.**

**9. \*\*Exception Handling:\*\***

**- Try, catch, throw mechanisms for handling exceptions.**

**10. \*\*Templates:\*\***

**- Function templates and class templates.**

**11. \*\*Smart Pointers:\*\***

**- `std::unique\_ptr`, `std::shared\_ptr`, and `std::weak\_ptr` for managing memory.**

**12. \*\*Concurrency (Optional):\*\***

**- Basics of multi-threading and concurrent programming.**

**As you progress through these topics, practice is crucial. Work on small projects and coding exercises to reinforce your understanding. Additionally, refer to C++ documentation and books for in-depth explanations of each concept.**

**Helpful Links:**

* [cplusplus.com/doc/ascii/](https://cplusplus.com/doc/ascii/) **ASCII table**

***Here are some of the essential keywords in C++:***

**The most used keywords in C++ can be categorized into several groups based on their functionalities.**

**Data Types:**

***int, char, float, double, bool*: Fundamental data types.**

**void: Represents lack of type.**

**Control Flow:**

***if, else, switch, case, default*: Conditional statements.**

**for, while, do: Looping constructs.**

**Functions:**

***return*: Returns a value from a function.**

***void*: Specifies that a function does not return a value.**

**Pointers and References:**

***\* (asterisk)*: Used for pointer declaration and dereferencing.**

***& (ampersand)*: Used for reference declaration.**

**Memory Management:**

***new, delete*: Dynamic memory allocation and deallocation.**

**Object-Oriented Programming (OOP):**

***class, struct*: Defines a class or structure.**

***public, private, protected*: Specifies access levels in a class.**

***this*: Pointer to the current instance of a class.**

***virtual:* Used in virtual functions and classes.**

***override*: Indicates that a function is intended to override a virtual function in a base class.**

**Modifiers:**

***const*: Indicates that a variable's value cannot be changed.**

***static*: Specifies that a variable or function is shared among all instances of a class.**

***volatile*: Indicates that a variable may be changed by external processes.**

**Control Statements:**

***break, continue*: Control flow within loops.**

***goto*: Unconditional jump to a labeled statement.**

**Error Handling:**

***try, catch, throw*: Used for exception handling.**

**Namespaces:**

***namespace:* Declares a named scope.**

***Templates*:**

***template*: Used for template metaprogramming.**

**Standard Library:**

***std::*: Standard namespace prefix for elements in the Standard Template Library (STL).**

**While many keywords in C++ are commonly used and essential for programming, some keywords are less frequently used but still important to know. These may be specialized or provide features that are used in specific situations. *Here are a few less commonly used keywords in C++ that are good to be aware of:***

***explicit*: Used in constructors to prevent implicit type conversions.**

***friend*: Grants non-member functions or other classes access to the private and protected members of a class.**

***mutable*: Specifies that a member of an object can be modified even if the object is declared as const.**

***typename*: Used in template programming to indicate that a dependent name is a type.**

***using*: Declares a using directive or introduces a typedef.**

***export*: Historically used for export of template definitions but has been removed in recent C++ standards.**

***decltyp*e: Returns the type of an expression.**

***constexpr*: Indicates that a variable or function can be evaluated at compile-time.**

***noexcept*: Specifies that a function does not throw exceptions.**

***aligna*s and *alignof*: Used for controlling alignment of variables.**

**C++ and Java are both powerful and widely-used programming languages, but they have some significant differences. *Here are some of the key distinctions between C++ and Java:***

**Paradigm:**

**C++: C++ is a multi-paradigm programming language, supporting procedural, object-oriented, and generic programming.**

**Java: Java is primarily an object-oriented programming language with some support for functional programming.**

**Memory Management:**

**C++: Developers have direct control over memory management using pointers and manual memory allocation/deallocation (new and delete operators).**

**Java: Memory management is automatic through garbage collection. Developers don't need to explicitly free memory, as the Java Virtual Machine (JVM) handles it.**

**Platform Dependency:**

**C++: Compiled into machine-specific code, making it more platform-dependent.**

**Java: Compiled into an intermediate bytecode, which is then executed by the Java Virtual Machine (JVM). This bytecode makes Java more platform-independent ("Write Once, Run Anywhere" principle).**

**Syntax:**

**C++: Syntax is similar to C and has pointers, multiple inheritance, and manual memory management.**

**Java: Syntax is similar to C++, but it lacks pointers and supports single inheritance (with interfaces), and memory management is handled by the JVM.**

**Object-Oriented Features:**

**C++: Supports multiple inheritance, operator overloading, and friend functions.**

**Java: Supports single inheritance through classes and multiple inheritance through interfaces. Operator overloading is not allowed.**

**Exception Handling:**

**C++: Uses both try-catch blocks for exception handling and also supports the use of the throw keyword.**

**Java: Uses try-catch blocks exclusively for exception handling and the throw keyword.**

**Threading:**

**C++: Supports multithreading through native threads and libraries like POSIX threads.**

**Java: Has built-in support for multithreading with the java.util.concurrent package.**

**Standard Libraries:**

**C++: Standard Template Library (STL) provides a rich set of generic algorithms and containers.**

**Java: Has its own standard libraries, including the Java Standard Edition (SE) libraries.**

**Compilation and Execution:**

**C++: Compiled directly into machine code using a compiler.**

**Java: Compiled into bytecode, which is then interpreted by the JVM or compiled Just-In-Time (JIT) into machine code for execution.**

**Pointers:**

**C++: Supports pointers, allowing for direct memory manipulation.**

**Java: Does not have explicit pointer support. References in Java are not as low-level as pointers in C++.**

***Pointer Similarity and Differences between C and C++***

**In C and C++, pointers, dereferencing, and passing pointers through functions are concepts closely related to memory manipulation and function parameter passing. Here's a brief overview of the differences between C and C++ in this context:**

**Pointers:**

**C:**

**Pointers are a fundamental concept in C.**

**Pointers can be used to store memory addresses and manipulate data indirectly.**

**Pointer declaration syntax: type \*variable\_name;**

**C++:**

**C++ also supports pointers and inherits the pointer features from C.**

**Additional features like smart pointers (std::shared\_ptr, std::unique\_ptr) are introduced in C++ for safer memory management.**

**Dereferencing:**

**C:**

**In C, dereferencing is done using the \* (asterisk) operator.**

**Example:**

**c**

**Copy code**

**int x = 10;**

**int \*ptr = &x; // Pointer declaration and initialization**

**int value = \*ptr; // Dereferencing the pointer to get the value at the memory address**

**C++:**

**C++ inherits the dereferencing syntax from C.**

**Example:**

**cpp**

**Copy code**

**int x = 10;**

**int \*ptr = &x; // Pointer declaration and initialization**

**int value = \*ptr; // Dereferencing the pointer to get the value at the memory address**

**Passing Pointers Through Functions:**

**C:**

**In C, when you pass a pointer to a function, the function receives a copy of the pointer.**

**This allows the function to modify the data at the memory location pointed to by the pointer.**

**Example:**

**c**

**Copy code**

**void modifyValue(int \*ptr) {**

**\*ptr = 20; // Modifying the value at the memory address pointed to by ptr**

**}**

**int main() {**

**int x = 10;**

**int \*ptr = &x;**

**modifyValue(ptr);**

**// Now, the value of x is 20**

**return 0;**

**}**

**C++:**

**C++ supports the same mechanism for passing pointers to functions as in C.**

**Additionally, C++ allows the use of references, providing an alternative syntax for passing and modifying data through functions.**

**Example:**

**cpp**

**Copy code**

**void modifyValue(int \*ptr) {**

**\*ptr = 20; // Modifying the value at the memory address pointed to by ptr**

**}**

**int main() {**

**int x = 10;**

**int \*ptr = &x;**

**modifyValue(ptr);**

**// Now, the value of x is 20**

**return 0;**

**}**

**In summary, the concepts of pointers, dereferencing, and passing pointers through functions are similar in C and C++. C++ introduces additional features such as smart pointers and references, providing more options for memory management and function parameter passing.**

***Understanding File Structure***

**When organizing a C++ project into folders, you typically structure it in a way that separates different concerns and components. Here's a common folder structure for a C++ project:**

**/project\_root**

**/inc**

**header1.h**

**header2.h**

**/src**

**main.cpp (Not like Java you can name your main() function anyway you like)**

**source1.cpp**

**source2.cpp**

**/lib**

**library1**

**library2**

**/bin**

**executable**

**/docs**

**/tests**

**Explanation of each folder:**

**inc: This folder contains header files (.h) that declare the interfaces of your classes and functions. These headers are meant to be included by other parts of your code or by external code using your library.**

**src: This folder contains the source files (.cpp) that implement the functionality declared in the header files. It's where you define the actual behavior of your classes and functions.**

**lib: This folder can contain third-party libraries or external dependencies that your project relies on. This can include precompiled libraries or source code that you compile along with your project.**

**bin: The binary folder holds the compiled executable files. After compiling your source code, the resulting executable (or binaries) are placed in this folder.**

**docs: This folder is for documentation. You might include a README file, code documentation generated by tools like Doxygen, or any other relevant documentation for your project.**

**tests: This folder is for your test suite. It can include unit tests, integration tests, or any other tests you use to verify the correctness of your code.**

***The content provided herein is intended solely for educational purposes. No part of this material may be reproduced, distributed, or sold without explicit permission. All rights to the content are reserved within the guidelines set forth by Eastern Washington University (EWU) and the ACM Club. Unauthorized use or distribution is strictly prohibited. Any requests for reproduction or distribution should be directed to the appropriate authorities at EWU or the ACM Club in accordance with their established guidelines and policies.***

**ACM club website (Join Us)** [**Computer Science ACM – Eastern Washington University (ewu.edu)**](https://inside.ewu.edu/csacm/)

**This document is produced by Yue “Jimmy” Yang** [**yyang13@ewu.edu**](mailto:yyang13@ewu.edu)

**\*TO-DO put a survey here maybe**

**Upcoming ACM events Looking forward for you to join us:**

* **April 13th 2024 “Code Fest” Link below**