In C++, classes are essential when coding. Most importantly, the names utilized for classes must follow the right rules and criteria, just as the conventional naming of variables is important as well. For classes specifically, the name should be a noun and the name should be camelcase to separate the words from each other while the first letter of the class must be upper case. However, something to remember is that underscores are NOT allowed in class names.

While underscores are not allowed in class names, they are allowed in variable names as long as the rest of the variable name is valid in following their own rules. C++ asks that variable names begin with an alphabet and numbers are allowed, but only after the first letter is strictly an alphabet. Additionally, keywords are not allowed as variable names and any special characters, including whitespaces, are also not allowed as this language is case sensitive so the smallest details are important.

To furthermore break down the naming conventions, classes, structs and typedefs start with a capital letter while others such as functions and variables start with a lowercase letter. Pointer Variables require the keyword be followed by an asterisk (*) and the asterisk is followed by the variable name. Interfaces, which are functions an object must have to be classified as that object, are named with an *Interface* suffix, abstract base classes with an *Abstract* prefix, member variables with a trailing underscore. Global variables have a prefix of g_ while global constants typically are named with the prefix c_ . Static class variables are named with a prefix, s_ .

C++, a strongly typed programming language, is a statically-typed language meaning that data types are associated with variables rather than values. The variable types are determined at compile time so the naming conventions mentioned above are enforced by the compiler. It is both explicitly and implicitly typed. The difference between the two is that explicitly typed languages have variables declared by the users while the implicit type has the program inferring what the variable data types are. As for mutability, c++ objects are mutable by default but can use keywords such as const or constexpr to declare an object immutable.What these keywords do is prevent objects, methods() and variables from being modified.
https://www.codeproject.com/Articles/5315601/Python-vs-Cplusplus-Series-Mutable-Immutable-and-C#:~:text=C%2B%2B%20supports%20two%20notions%20of,objects%20are%20mutable%20by%20default
https://en.wikipedia.org/wiki/Immutable_object#C++

Binding is defined as converting identifiers into addresses. Identifier names and operator symbols are bound during their binding time in C++. This binding time occurs at the moment of its creation or declaration. The binding in C++ is static by default. It is known as early binding as the programmer is the one to explicitly make the call in the program.

## Naming Convention examples:

int myAge = 22;
Char name[5] = “Aang”; → the string variable is a char data type. It is an array known as C-strings that end with null characters so the length is the number of characters + 1
Floating-point number → float decimal = 2.5;
Boolean → bool  real = true;
Outputting a boolean returns either 1 or 0 depending whether a statement is true or false.
Array/list: → int score [5] = {10, 20, 15, 10, 30}; → Arrays must contain elements of one data type, you can not replace an int element with a string element in an array of ints.
Dictionary(map): map<int, string> colors= {{1, “Purple”,},
{2, “Black”, },
{3, “Orange”,}}
https://www.udacity.com/blog/2020/03/c-maps-explained.html

## Mixed Type Arithmetic

int/int = int → 9/2 = 4
float/int = float → 9.0/2 = 4.5
int/float = float → 9/2.0 = 4.5
float/float = float → 9.0/2.0 = 4.5

## Arithmetic operators:

For performing arithmetic operations on variables and data
+Addition
- Subtraction
* Multiplication
/Division
%Modulo Operation(Remainder after division)
++Increment
–Decrement

## Assignment Operators

= → a = b;
+= → a = a + b;
-= → a = a - b;
*= → a = a * b;
/= → a = a / b;
%= → a = a % b;

## Relational Operators

== Is Equal To
!= Not Equal To
> Greater Than
< Less Than
>= Greater Than or Equal To
<= Less Than or Equal To
Logical Operators
&& → Logical AND → True only if all expressions are true
|| → Logical OR → True if at least one expression is true
! → Logical NOT → True only if operand is false

## Bitwise Operators

Used to perform operations on individual bits but can only be used alongside char and int data types.

& Binary AND
| Binary OR
^ Binary XOR
~ Binary One's Complement
<< Binary Shift Left
>> Binary Shift Right

https://www.programiz.com/cpp-programming/operators

Concatenating an integer to a string object can be done by calling the std:: to_string function. For example if we have an int x = 10 and call the to_string function, that line of code will look like → std::string y = "Number " + std::to_string(x); The code would thus compile and run to output Number 10. However, expressions with strings and ints can also be outputted. An example is std::cout << "Half of " << 12 << " is " << (12/2) << "\n"; This would execute Half of 12 is 6. These examples apply to floating point numbers as well, just switch the examples from ints to floats and you'll be able to concatenate strings with floats and output floats with strings in expressions.
https://www.techiedelight.com/concatenate-integer-string-object-cpp/
https://www.oreilly.com/library/view/practical-c-programming/0596004192/ch04.html

However something such as declaring an integer as a string plus a number for example int x = "5" + 6; will not output. Instead an error is received stating "invalid conversion from 'const char' to 'int'.

Stacks and Heaps

When it comes to memory in c++, memory can be allocated and de-allocated using either stack or heap. There are a couple of advantages and disadvantages based on which is used. In stack, allocations are done by the compiler but with a heap, the allocations must be done by the programmer. Accessing data with stack memory is a lot faster than with heap memory. This is the result of their data structure. A concern with stack is memory shortage while with heap memory, a concern is fragmentation of allocated memory being repetitively released. While stack is automatically managed by the computer, heaps are more free range, not controlled neither by the user or the CPU. A big advantage to stack memory is that data is automatically cleared after function calls whereas heap memory requires the programmer to do the clearing by freeing data when required. Stack memory is associated with allowing access only to local variables and data methods, but heap is convenient to access global data and data present anywhere in the program. Overall, both stack memory and heap memory are utilized in the C++ language. Stack memory is responsible for local variables, arguments that are passed through a function and their return addresses, using LIFO(Last In First Out). Heap memory is responsible for storing data such as all the global variables, accessed through pointers that can reach anywhere in the memory. It is important to note that pointers are allocated in the stack but are utilized to access memory on the heap.

https://www.educba.com/c-stack-vs-heap/