Data Types

Important data types you will be working with,

`int`	Integers are numbers that represent whole positive or negative numbers. Holds 2, 4, or 8 bytes, depending on the compiler. Typically `int` holds 4 bytes. `int` is `signed`, meaning it holds *positive and negative* numbers. A 4 byte `int` has a value range from -2,147,483,648 to 2,147,483,647.
`uint`	Unsigned Integers are whole *positive* numbers. Size: 2, 4 or 8 bytes. Is unsigned, meaning only positive numbers. A 4 byte `uint` has a value range from 0 to 4,294,967,295. Another acceptable syntax for `uint` is `unsigned int`.
`short`	A `short` is the same as `int`, but holds 2 bytes. Size: 2 bytes. Is `signed`. Value range from -32,768 to 32,767. You may also declare an `unsigned short`.
`long`	A `long` is the same as `int`, but holds 4 bytes. Size: 4 bytes. Is signed. Value range from -2,147,483,648 to 2,147,483,647. You may also declare an `unsigned long`.
`float`	Floating-point numbers are very large and very small signed numbers with decimals to indicate degrees of accuracy. Size: 4 bytes. Is `signed`. Precision to 6 decimal places.
`double`	A `double` is the same as `float`, but holds 8 bytes. Size: 8 bytes. Is signed. Precision to 15 decimal places. You may also declare a `double float` which has a size of 10 bytes and precision to 19 decimal places.
`char`	Character variables are a fixed 1 byte representation of an integer as an ASCII character such as ‘Z’ (90) or ‘z’ (122). Size: 1 byte (8 bits). Is signed. The char type holds 256 values. Value range from -128 to 127.
`string`	Strings are not a “built-in” data type like the above data types (`int`, `float` `char`), and is part of the C++ standard library, requiring `#include <string>` to work. Strings represent sequences of characters, literally a “string” of 1 byte `char` values strung together.
`bool`	Boolean types are the simplest data type and are either `true` or `false`.

Integer Values

- The size of an `int` is 2, 4, or 8 bytes, this is compiler dependent. When processors were 16 bit, an `int` was 2 bytes. Currently `int` is most often 4 bytes on a 32 bits system, or 8 bytes on 64 bits system.
- `int` is `signed`, meaning it holds *positive and negative* numbers.
- `uint` is `unsigned` version of `int`. `uint` is shorthand for `unsigned int`.
- `short` and `long` are more specific versions of int. They may also be `unsigned`.

- Why `i`, `j`, `k` variable names? Mathematicians use i, j, k in summation notation, traditionally i for the first index, j for the second, etc. `i` is appropriate as index in loops when programming. In Fortran, one of the earliest widespread programming languages, integer variables had to start with the letters I through N and real variables started with the other letters. Habits developed by programmers using Fortran carried over to other languages.

cout [C++]

- C++ uses a convenient abstraction called *streams* to perform input and output operations to or from sequential "media" such as the screen, the keyboard, or a file. A stream is an entity that a program can either insert or extract characters to or from.
- C++ has a much simpler method than C's printf() called `cout`.
- You simply pass what you want to print as a stream into cout.

```
cout << "Hello " << endl << " C++" << endl;
```

- `endl` for `cout` is the same as `"\n"`

sizeof()

To get the exact size of a type or a variable in a particular compiler, you can use the `sizeof` operator. `sizeof(type)` yields the storage size of the type in bytes.

```
cout << "Storage size for int :" << sizeof(int) << endl;
```

Floating-point Values

- Floating-point numbers are very large and very small signed numbers with decimals to indicate degrees of accuracy. Typically floating point values are declared using `float`, which is a `signed` value of 4 bytes in size. A `float` has precision to 6 decimal places.
- If you need greater precision than that use a `double`, which is the same as a `float` but holds 8 bytes of information and will give precision to 15 decimal places.
- If you need greater precision still you can use a `double float` which has a size of 10 bytes and precision to 19 decimal places.

Constants

- Read-only variables, constants cannot change their value during program execution.
- A classic example of a constant is Pi. `const float PI = 3.14159;`
- Constants help readability of code. Try to avoid "magic numbers" in your code and instead use a constant describing what the number is or does.

For example, the header file float.h (more detail about header files later) defines macros that allow you to use various constants (and other details) about the binary representation of real numbers in your programs.

```
#include <float.h>
```

…

```
cout << "Storage size for float :" << sizeof(float) << endl;
cout << "Minimum float positive value:" << FLT_MIN << endl;
cout << "Maximum float positive value:" << FLT_MAX << endl;
cout << "Precision value:” << FLT_DIG << endl;
```

Note that by convention constants are CAPITALIZED. Such as PI, and, FLT_MIN and FLT_MAX that you see here.

Character Values

- A `char` is a representation of an `int` as an ASCII (American Standard Code for Information Interchange) *character code.* The character code 90 represents the uppercase latter ‘Z’.
- Note Lowercase ‘z’ has a character code of 122.
- ASCII was developed from telegraph code. Its first commercial use was as a seven-bit teleprinter code promoted by Bell data services in 1960.
- ASCII standard was developed by the American Standards Association (ASA), which ultimately became the American National Standards Institute (ANSI), who standardized C.
- For a full list of ASCII character codes see the handout “ASCII character codes”
- try adding a `char` to an `int`.
- Multiple characters (words) are stored as a character array or `string`, to be discussed later.

Naming Conventions

- Always begin variables with a lower-case letter.
- Constants should always use ALL CAPS.
- Try to avoid “magic numbers” in your code and instead use a clearly named constant to describe what the number or value represents, e.g., `const float PI = 3.14159;`
- When declaring a `char` use single quotes. `char c = 'z';`
- You should choose between using "Camel Case", e.g. `int daysInSeptember = 30;`
- or all "Lower Case", variable names. e.g. `int days_in_september = 30;`
- Just as you should choose between “cuddling”, or indenting your braces, it’s more important to be consistent with variable naming rather than which convention you choose.
- Some people like using "Hungarian Notation" when naming their variables. Hungarian notion was popular at Microsoft in the 80s and early 90s, though is now considered unnecessary. Hungarian Notation begins the name of a variable by indicating its data type, e.g. `int iDaysInSeptember = 30;` Here the `i` prefix denotes the variable is an integer.
- C++ Classes, which you will learn about later, should always start with a capitol letter.
- The C# language has reintroduced "Pascal Case" where the variable starts with a capitol, e.g. `int DaysInSeptember = 30;` This convention is acceptable, bot not recommended in C++ as it does not distinguish between variables and C++ Classes.