Basic Terminology for Programming in C++

Reference: *The C++ Programming Language (3rd edition)* by B. Stroustrup.

bit— Binary Digit. That is, a variable quantity capable of holding exactly one of two possible values. By convention, these values are labeled 0 (zero) and 1 (one). A **byte** is 8 bits.

A **kilobyte** (KB) is 2^{10} bytes = 1024 bytes.
A **megabyte** (MB) is 2^{20} bytes $\approx 10^6$ bytes.
A **gigabyte** (GB) is 2^{30} bytes $\approx 10^9$ bytes.
A **terabyte** (TB) is 2^{40} bytes $\approx 10^{12}$ bytes.
A **petabyte** (PB) is 2^{50} bytes $\approx 10^{15}$ bytes.
An **exabyte** (EB) is 2^{60} bytes $\approx 10^{18}$ bytes.

We use the same prefixes to quantify processing speeds, e.g. GHz for gigahertz (2^{30} clock cycles/sec) or TFlops for teraflops (2^{40} floating-point operations per second).

Small-scale times and dimensions are referred to with metric prefixes:
milli (10^{-3}), *micro* (10^{-6}), *nano* (10^{-9}),
pico (10^{-12}), *femto* (10^{-15}).

type— A rule or set of rules for

1. the translation of certain symbols or symbol sequences into a specified number of bits
2. the syntax involving identifiers declared to have that type.

Remarks.

1.

(a) A type is either *exact* or *inexact*.

(b) Exact types are represented as *integers*. Inexact types are represented as rounded real numbers in *floating-point*.

(c) In C++, the basic built-in types are `bool`, `char`, `int`, and `float`. Of these, `float` is inexact; the others are exact.

Remarks on **type**, *continued*

2. From the basic built-in types, the following types are derived: enumerations, `void`, pointer types, array types, reference types, and programmer-defined types (`struct`s and `class`es).

3. Variations on `int` and `float`:
`double` – twice as many digits as a `float`.
`short, long, unsigned` – variations on `int`.

4. Definitions of built-in types may vary from one platform to another. Use the `sizeof` operator to determine how many bytes a particular type uses on a given platform.

5. Because individual bits in most machines are not actually addressable, a variable of type `bool` usually uses the same number of bits as a variable of type `char`.

object— A contiguous region of memory with a type.
First-class objects are endowed with all the "usual" operations, e.g., assignment, and can generally be handled just as the basic built-in types (`int, double, char, bool`) are handled.
Second-class objects (e.g., arrays) have special syntactical restrictions on their usage (e.g., assignment is not defined for arrays).

grammar— The formal definition of the language. The grammar specifies what combinations of symbols are legal and how these combinations are evaluated.

expression— A sequence of symbols that can be reduced to a single value of a specific type. Expressions are typically nested. Numbers and identifiers are examples of *primitive* expressions.

identifier— A programmer-specified name; a sequence of characters associated with and used to access a particular object, class or struct, or function. Every identifier has a *scope*.

scope— The source code extending from a declaration up until the end of the *block* which contains the declaration.

lifetime— The interval of run-time during which a particular object is reserved for use by the program.

The lifetime of an object whose memory is reserved at compile time (a **statically allocated** object) is simply the time required to execute the instructions for all statements in its scope. A statically allocated object "dies" when its identifier goes out of scope.

The lifetime of an object whose memory is reserved at run-time (a **dynamically allocated** object) extends from the time at which the memory is allocated to the time at which is deallocated. *For dynamically allocated objects, there is no connection between scope and lifetime.*

declaration— The introduction of a name (identifier) into a scope.

definition— The association of an identifier with a specific object (in memory).

Every definition is also a declaration, and *most* declarations are also definitions.

Not every declaration is a definition. For example, `extern int i;` introduces the identifier `i` into the current scope, but the `extern` specifier says that the associated object was already created elsewhere (e.g., another file) in the source code.

statement Any of the following:
1. A declaration.
2. An expression followed by a semi-colon.
3. A sequence of statements enclosed within curly braces: a **block**.
4. A selection-control structure (an `if` or `switch` block); a `case` or `default` block within a such a structure.
5. An iteration-control structure (a `for`, `while`, or `do while` loop).
6. Any unconditional transfer of control of the form `break;`, `continue;`, `return;`, or `goto` *expression* `;`.
7. A statement preceded by an identifying label and a colon.
8. An exception-handler block (a `try-catch` block).

Note: Many authors reserve the word *block* for sequences of statements that contain declarations.

parse— To break a program, statement, or expression into its component parts.

compiler— A program that translates high-level source code, e.g., C++, into low-level machine language. Some compilers output *assembly language* which is then converted to machine language by a separate *assembler*. A compiler must first separate the statements of the source code from one another before it can translate them. Each statement must then be translated by evaluating its expressions according to the grammar. Both of these steps use *parsing*.

variable— An *object* with an *identifier*. It follows that a variable has all the following attributes.

1. A contiguous region of storage
2. An address
3. A type
4. A value
5. A name
6. A scope
7. A lifetime