Glossary

"Often, a few well-chosen words are worth a thousand pictures."

—Anonymous

A *glossary* is a brief explanation of words used in a text. This is a rather short glossary of the terms we thought most essential, especially in the earlier stages of learning programming. The index and the "Terms" sections of the chapters might also help. A more extensive glossary, relating specifically to C++, can be found at www.stroustrup.com/glossary.html, and there is an incredible variety of specialized glossaries (of greatly varying quality) available on the web. Please note that a term can have several related meanings (so we occasionally list some) and that most terms we list have (often weakly) related meanings in other contexts; for example, we don't define *abstract* as it relates to modern painting, legal practice, or philosophy.

abstract class a class that cannot be directly used to create objects; often used to define an interface to derived classes. A class is made abstract by having a pure virtual function or a protected constructor.

abstraction a description of something that selectively and deliberately ignores (hides) details (e.g., implementation details); selective ignorance.

address a value that allows us to find an object in a computer's memory.

algorithm a procedure or formula for solving a problem; a finite series of computational steps to produce a result.

alias an alternative way of referring to an object; often a name, pointer, or reference.

application a program or a collection of programs that is considered an entity by its users.

approximation something (e.g., a value or a design) that is close to the perfect or ideal (value or design). Often an approximation is a result of trade-offs among ideals.

argument a value passed to a function or a template, in which it is accessed through a parameter.

array a homogeneous sequence of elements, usually numbered, e.g., [0:max).

assertion a statement inserted into a program to state (assert) that something must always be true at this point in the program.

base class a class used as the base of a class hierarchy. Typically a base class has one or more virtual functions.

bit the basic unit of information in a computer. A bit can have the value 0 or the value 1.

bug an error in a program.

byte the basic unit of addressing in most computers. Typically, a byte holds 8 bits.

class a user-defined type that may contain data members, function members, and member types.

code a program or a part of a program; ambiguously used for both source code and object code.

compiler a program that turns source code into object code.

complexity a hard-to-precisely-define notion or measure of the difficulty of constructing a solution to a problem or of the solution itself. Sometimes *complexity* is used to (simply) mean an estimate of the number of operations needed to execute an algorithm.

computation the execution of some code, usually taking some input and producing some output.

concept (1) a notion, an idea; (2) a set of requirements, usually for a template argument.

concrete class a class for which objects can be created.

constant a value that cannot be changed (in a given scope); not mutable.

constructor an operation that initializes ("constructs") an object. Typically a constructor establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).

container an object that holds elements (other objects).

copy an operation that makes two objects have values that compare equal. See also **move**.

correctness a program or a piece of a program is correct if it meets its specification. Unfortunately, a specification can be incomplete or inconsistent, or can fail to meet users' reasonable expectations. Thus, to produce acceptable code, we sometimes have to do more than just follow the formal specification.

cost the expense (e.g., in programmer time, run time, or space) of producing a program or of executing it. Ideally, cost should be a function of complexity.
data values used in a computation.
debugging the act of searching for and removing errors from a program; usually far less systematic than testing.
declaration the specification of a name with its type in a program.
definition a declaration of an entity that supplies all information necessary to complete a program using the entity. Simplified definition: a declaration that allocates memory.
derived class a class derived from one or more base classes.
design an overall description of how a piece of software should operate to meet its specification.
destructor an operation that is implicitly invoked (called) when an object is destroyed (e.g., at the end of a scope). Often, it releases resources.
encapsulation protecting something meant to be private (e.g., implementation details) from unauthorized access.
error a mismatch between reasonable expectations of program behavior (often expressed as a requirement or a users' guide) and what a program actually does.
executable a program ready to be run (executed) on a computer.
feature creep a tendency to add excess functionality to a program "just in case."
file a container of permanent information in a computer.
floating-point number a computer's approximation of a real number, such as 7.93 and 10.78e–3.
function a named unit of code that can be invoked (called) from different parts of a program; a logical unit of computation.
generic programming a style of programming focused on the design and efficient implementation of algorithms. A generic algorithm will work for all argument types that meet its requirements. In C++, generic programming typically uses templates.
handle a class that allows access to another through a member pointer or reference. See also **copy**, **move**, **resource**.
header a file containing declarations used to share interfaces between parts of a program.
hiding the act of preventing a piece of information from being directly seen or accessed. For example, a name from a nested (inner) scope can prevent that same name from an outer (enclosing) scope from being directly used.
ideal the perfect version of something we are striving for. Usually we have to make trade-offs and settle for an approximation.
implementation (1) the act of writing and testing code; (2) the code that implements a program.
infinite loop a loop where the termination condition never becomes true. See **iteration**.

infinite recursion a recursion that doesn't end until the machine runs out of memory to hold the calls. In reality, such recursion is never infinite but is terminated by some hardware error.

information hiding the act of separating interface and implementation, thus hiding implementation details not meant for the user's attention and providing an abstraction.

initialize giving an object its first (initial) value.

input values used by a computation (e.g., function arguments and characters typed on a keyboard).

integer a whole number, such as 42 and –99.

interface a declaration or a set of declarations specifying how a piece of code (such as a function or a class) can be called.

invariant something that must be always true at a given point (or points) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement.

iteration the act of repeatedly executing a piece of code; see **recursion**.

iterator an object that identifies an element of a sequence.

library a collection of types, functions, classes, etc. implementing a set of facilities (abstractions) meant to be potentially used as part of more than one program.

lifetime the time from the initialization of an object until it becomes unusable (goes out of scope, is deleted, or the program terminates).

linker a program that combines object code files and libraries into an executable program.

literal a notation that directly specifies a value, such as 12 specifying the integer value "twelve."

loop a piece of code executed repeatedly; in C++, typically a **for**-statement or a **while**-statement.

move an operation that transfers a value from one object to another, leaving behind a value representing "empty." See also **copy**.

mutable changeable; the opposite of immutable, constant, and variable.

object (1) an initialized region of memory of a known type which holds a value of that type; (2) a region of memory.

object code output from a compiler intended as input for a linker (for the linker to produce executable code).

object file a file containing object code.

object-oriented programming a style of programming focused on the design and use of classes and class hierarchies.

operation something that can perform some action, such as a function and an operator.

output values produced by a computation (e.g., a function result or lines of characters written on a screen).

overflow producing a value that cannot be stored in its intended target.

overload defining two functions or operators with the same name but different argument (operand) types.

override defining a function in a derived class with the same name and argument types as a virtual function in the base class, thus making the function callable through the interface defined by the base class.

owner an object responsible for releasing a resource.

paradigm a somewhat pretentious term for design or programming style; often used with the (erroneous) implication that there exists a paradigm that is superior to all others.

parameter a declaration of an explicit input to a function or a template. When called, a function can access the arguments passed through the names of its parameters.

pointer (1) a value used to identify a typed object in memory; (2) a variable holding such a value.

post-condition a condition that must hold upon exit from a piece of code, such as a function or a loop.

pre-condition a condition that must hold upon entry into a piece of code, such as a function or a loop.

program code (possibly with associated data) that is sufficiently complete to be executed by a computer.

programming the art of expressing solutions to problems as code.

programming language a language for expressing programs.

pseudo code a description of a computation written in an informal notation rather than a programming language.

pure virtual function a virtual function that must be overridden in a derived class.

RAII ("Resource Acquisition Is Initialization") a basic technique for resource management based on scopes.

range a sequence of values that can be described by a start point and an end point. For example, [0:5) means the values 0, 1, 2, 3, and 4.

recursion the act of a function calling itself; see also **iteration**.

reference (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.

regular expression a notation for patterns in character strings.

requirement (1) a description of the desired behavior of a program or part of a program; (2) a description of the assumptions a function or template makes of its arguments.

resource something that is acquired and must later be released, such as a file handle, a lock, or memory. See also **handle**, **owner**.

rounding conversion of a value to the mathematically nearest value of a less precise type.

scope the region of program text (source code) in which a name can be referred to.

sequence elements that can be visited in a linear order.

software a collection of pieces of code and associated data; often used interchangeably with **program**.
source code code as produced by a programmer and (in principle) readable by other programmers.
source file a file containing source code.
specification a description of what a piece of code should do.
standard an officially agreed-upon definition of something, such as a programming language.
state a set of values.
string a sequence of characters.
style a set of techniques for programming leading to a consistent use of language features; sometimes used in a very restricted sense to refer just to low-level rules for naming and appearance of code.
subtype derived type; a type that has all the properties of a type and possibly more.
supertype base type; a type that has a subset of the properties of a type.
system (1) a program or a set of programs for performing a task on a computer; (2) a shorthand for "operating system," that is, the fundamental execution environment and tools for a computer.
template a class or a function parameterized by one or more types or (compile-time) values; the basic C++ language construct supporting generic programming.
testing a systematic search for errors in a program.
trade-off the result of balancing several design and implementation criteria.
truncation loss of information in a conversion from a type into another that cannot exactly represent the value to be converted.
type something that defines a set of possible values and a set of operations for an object.
uninitialized the (undefined) state of an object before it is initialized.
unit (1) a standard measure that gives meaning to a value (e.g., km for a distance); (2) a distinguished (e.g., named) part of a larger whole.
use case a specific (typically simple) use of a program meant to test its functionality and demonstrate its purpose.
value a set of bits in memory interpreted according to a type.
variable a named object of a given type; contains a value unless uninitialized.
virtual function a member function that can be overridden in a derived class.
word a basic unit of memory in a computer, usually the unit used to hold an integer.