Glossary

2-node—A tree node that contains one data item and has two children. Also called a binary node. See also *3-node* and *4-node*.

2-3 tree—A tree such that each internal node (nonleaf) has either two or three children, and all leaves are at the same level. A node can have a left subtree, a middle subtree, and a right subtree. If a node has two children and contains one data item, the item's value must be greater than the value of the item in the left child and smaller than the value of the item in the right child. If a node has three children and contains two data items, the value of the smaller item in the node must be greater than the value of the item in the left child and smaller than the value of the item in the middle child; the value of the larger item in the node must be greater than the value of the item in the middle child and smaller than the value of the item in the right child.

2-3-4 tree—A tree such that each internal node (nonleaf) has either two, three, or four children, and all leaves are at the same level. A node can have a left subtree, a middle-left subtree, a middle-right subtree, and a right subtree. If a node has two or three children, it adheres to the specifications of a 2-3 tree. If a node has four children and three data items, the value of the smallest item in the node must be greater than the value of the item in the left child and smaller than the value of the item in the middle-left child; the value of the middle item in the node must be greater than the value of the item in the middle-left child and smaller than the value of the item in the middle-right child; and the value of the largest item in the node must be greater than the value of the item in the middle-right child and smaller than the value of the item in the right child.

3-node—A tree node that contains two data items and has three children. See also *2-node* and *4-node*.

4-node—A tree node that contains three data items and has four children. See also *2-node* and *3-node*.

abstract base class—A class without instances that forms the basis of other classes that descend from it. An abstract base class must contain at least one pure virtual method. Also known as an abstract class.

abstract class—Another term for an *abstract base class*.

abstract data type (ADT)—A specification of data values and the operations on that data. This specification does not indicate how to store the data or how to implement the operations, and it is independent of any programming language.

abstraction—See *data abstraction* and *functional abstraction*.

acceptance testing—Testing a system to see that it complies with its requirements. See also *beta testing*, *closed-box testing*, *integration testing*, *open-box testing*, *system testing,* and *unit testing.*

access time—The time required to access a particular item in a data structure such as an array, a linked chain, or a file.

accessor—Another term for an *accessor method*.

accessor method—A method of a class that returns the value of a data member. Also known as an accessor. See also *mutator method*.

activation bar—An open box, drawn on an object's lifeline in a UML sequence diagram, that represents a message handled by the object.

activation record—A record that contains the local environment of a function or method at the time of and as a result of the call to the function or method.

adapter container—A container class that provides a restricted interface to another container used in its implementation.

address—A number that labels a location in a computer's memory.

adjacency list—The n chains of linked nodes that implement a graph of n vertices numbered $0, 1, \ldots, n-1$ such that there is a node in the i^{th} linked chain for vertex j if and only if there is an edge from vertex i to vertex j.

adjacency matrix—An n-by-n array a that implements a graph of n vertices numbered $0, 1, \ldots, n-1$ such that `a[i][j]` is 1 if and only if there is an edge from vertex i to vertex j.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

adjacent vertices—Two vertices of a graph that are joined by an edge. In a directed graph, vertex *y* is adjacent to vertex *x* if there is a directed edge from vertex *x* to vertex *y*.

ADT—See *abstract data type*.

aggregate data type—A data type composed of multiple elements. Some examples of aggregate data types are arrays, collections, containers, and files.

aggregation—A *has-a* relationship between two classes such that the instances of the contained class can exist independently of an instance of the containing class. See also *composition*.

algorithm—A step-by-step recipe for performing a task within a finite amount of time.

alias—Another name for a variable, typically associated with reference types. In the code

```
int y = 0;
int* x = &y;
```

the reference `x` is an alias for the variable `y`.

allocate—To assign memory to an object when it is created. See also *dynamic allocation* and *static allocation*.

allocator—An object that manages the memory allocation for a container.

alternate scenario—A textual story within a use case that describes the action taken in an exceptional situation.

analysis—The process of understanding a problem and the requirements of its solution. See also *object-oriented analysis*.

analysis of algorithms—A branch of computer science that measures the efficiency of algorithms.

ancestor class—Any base class within a chain of base classes that occurs when inheritance is used to define a given class. See also *derived class*.

ancestor of a node *n*—A tree node on the path from the tree's root to *n*.

application heap—See *free store*.

argument—A variable or expression that is passed to a function or method. An argument appears in a call to a function (method) and corresponds to a parameter in the function's (method's) declaration. See also *parameter, reference parameter, and value parameter*.

arithmetic type—A data type that represents an integer or a floating-point value.

array—A data structure that contains a fixed maximum number of elements of the same data type that are referenced directly by means of an index, or subscript.

array-based implementation—An implementation of an ADT that uses an array to store the data values.

artifact—A document generated during object-oriented analysis and design (OOA/D) that represents some portion of the analysis or design work.

***as-a* relationship**—A relationship between classes whereby one class is implemented in terms of another through the use of private inheritance. See also *has-a relationship* and *is-a relationship*.

assertion—A statement that describes the state of an algorithm or program at a certain point in its execution.

association—A relationship between two classes in a UML class diagram whereby one of the classes uses the methods of the other class.

associative container—A variable-sized container that supports efficient retrieval of entries based on search keys.

attribute—In software engineering, a characteristic of an object. In the UML, a data member of a class. See also *data member*.

average-case time—An estimate of the average amount of time that a given algorithm requires to solve problems of size *n*. See also *best-case time* and *worst-case time*.

AVL tree—A balanced binary search tree in which rotations restore the tree's balance after each insertion or deletion of a node.

axiom—A mathematical rule or relationship. Axioms can be used to specify the behavior of an ADT operation.

back of a queue—The end of a queue at which items are inserted.

backtracking—A problem-solving strategy that, when it reaches an impasse, retraces its steps in reverse order before trying a new sequence of steps.

bag—A container of items in no particular order. Duplicate items are possible in a bag. See also *set*.

balanced binary tree—A binary tree in which the left and right subtrees of any node have heights that differ by at most 1. Also called a height-balanced tree.

base case—The known case in either a recursive definition or an inductive proof. Also called the basis or degenerate case.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

base class—A class from which another class is derived. The derived class inherits the base class's members. Also called the parent class or superclass. See also *ancestor class*, *derived class*, and *inheritance*.

basis—See *base case*.

behavior—An action that an object performs. See also *operation*.

best-case time—An estimate of the minimum amount of time that a given algorithm requires to solve problems of size *n*. See also *average-case time* and *worst-case time*.

beta testing—The use of advanced end users to test a nearly complete system in a real-world environment. See also *acceptance testing*, *closed-box testing*, *integration testing*, *open-box testing*, *system testing,* and *unit testing.*

BFS—See *breadth-first search*.

BFS spanning tree—A spanning tree formed by using a breadth-first search to traverse a graph's vertices.

bidirectional iterator—An iterator that can move either ahead or back from the current item in a container.

Big O notation—A notation that uses the capital letter O to specify an algorithm's order. For example, "O($f(n)$)" means "order $f(n)$." See also *order of an algorithm*.

binary file—A file whose data is in the computer's internal representation. A binary file is not organized into lines. Also called a general file or nontext file. See also *text file*.

binary node—See *2-node*.

binary operator—An operator that requires two operands, for example, the + in 2 + 3. See also *unary operator*.

binary search—An algorithm that searches a sorted collection for a particular item by repeatedly halving the collection and determining which half could contain the item.

binary search tree—A binary tree such that each node's data is greater than all the data in the node's left subtree but less than all the data in its right subtree.

binary tree—A set of nodes that is either empty or partitioned into a root node and two possibly empty sets that are binary trees. Thus, each node in a binary tree has at most two children, the left child and the right child. See also *n-ary tree*.

binding—The association of a variable with a memory address and the type of data the variable holds. See also *early binding* and *late binding*.

black-box testing—See *closed-box testing*.

block—(1) Another term for a *compound statement*. (2) A group of data records in a file.

block access—Reading or writing a block of data associated with a file.

boundary values—Values at the edges of valid data used during the testing of a solution. Testing should include both sides of the boundary, using both valid and invalid values.

box trace—A systematic way to trace the actions of a recursive function.

breadth-first search (BFS)—A graph-traversal strategy that visits every vertex adjacent to a vertex v that it can before it visits any other vertex. Thus, the traversal will not embark from any of the vertices adjacent to v until it has visited all possible vertices adjacent to v. See also *depth-first search*.

breakpoint—A debugger switch, set on a specific line of source code, that will cause execution to pause just before that line is executed, thereby enabling you to observe the values of variables or to single-step through the source code. See also *debugger* and *single-step*.

B-tree of degree *m*—A balanced search tree whose leaves are at the same level and whose nodes each contain between $m - 1$ and $[m / 2]$ data items. Each nonleaf has one more child than it has data items. The root of the tree can contain as few as one data item and can have as few as two children. Typically, a B-tree is stored in an external file.

bubble sort—A sorting algorithm that compares adjacent entries in an array and exchanges them if they are out of order. Comparing the first two entries, the second and third entries, and so on will move the largest entry to the end of the array. Repeating this process will eventually sort the array into ascending order.

bucket—A structure associated with a hash address that can accommodate more than one item. An array of buckets can be used as a hash table to resolve collisions.

buffer—A location that temporarily stores data as it makes its way from one process or location to another. A buffer enables data to leave one process or location at a different rate than the rate at which it enters another process or location, thus compensating for the difference in these rates.

cardinality—The number of entries in a set.

catch—To handle, or deal with, an exception.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

`catch` block—A group of C++ statements occurring immediately after a `try` block or another `catch` block that react to a particular exception thrown within the `try` block.

`catch` block parameter—An identifier in a `catch` block that represents the thrown exception of a specified type.

ceiling of *x*—Denoted by $\lceil x \rceil$, the value of x rounded up. For example, $\lceil 6.1 \rceil = 7$.

chain—(1) A data structure whose elements, or nodes, each contains both data and a pointer to another node and thus are linked to one another. Also called a singly linked chain. See also *circular chain*, *circular doubly linked chain*, and *doubly linked chain*. (2) A chain of linked nodes used within separate chaining, which is a collision-resolution scheme associated with hashing.

chaining—See *separate chaining*.

child class—See *derived class*.

child of a node *n*—A node directly below node *n* in a tree.

circuit—A special cycle that passes through every vertex (or edge) in a graph exactly once.

circular chain—A linked chain whose last node points to the first node in the chain.

circular doubly linked chain—A doubly linked chain whose first node contains a pointer to the chain's last node and whose last node contains a pointer to the chain's first node.

class—A programming construct that describes a group of objects, thereby defining the objects' data type as a new data type.

class diagram—A diagram in the Unified Modeling Language (UML) to represent a class. It consists of a three-part box that lists a class's name, attributes (data fields), and operations (methods). Also called a static diagram.

class-responsibility-collaboration card—See *CRC card*.

class template—A specification of a class in terms of a data-type parameter.

client—The program, module, or ADT that uses a class.

client interface—The header file that completely specifies the methods of an ADT and is all that a programmer needs to know about a class to be able to use it in a program. See also *interface*.

closed-box testing—The testing of a method, knowing only its name, parameter list, return type, and operation contract. Also called black-box testing or functional testing. See also *acceptance testing*, *beta testing*, *integration testing*, *open-box testing*, *system testing,* and *unit testing.*

closed-form formula—A nonrecursive algebraic expression.

clustering—The tendency of items to map into groups, called clusters, of locations in a hash table rather than randomly scattered locations. This difficulty, typical of the linear-probing collision-resolution scheme in hashing, can cause lengthy search times. See also *primary clustering* and *secondary clustering.*

code—Statements in a programming language.

code storage—Memory that the operating system reserves for program code when the program begins execution. Also known as text storage.

coding—Implementing an algorithm in a programming language.

cohesion—The degree to which the portions of a module are related.

collaboration—The process of objects sending messages to one another—that is, calling each other's methods—in order to solve a problem.

collision—A condition that occurs when a hash function maps two or more distinct search keys into the same location.

collision-resolution scheme—A technique used in hashing that assigns locations in the hash table to items with different search keys when the items are involved in a collision. See also *bucket, chain, clustering, double hashing, folding, linear probing, open addressing, probe sequence, quadratic probing,* and *separate chaining*.

comma operator—A comma that separates expressions and causes them to evaluate from left to right.

compile time—The time during which a compiler translates a program from source form into machine language. See also *runtime*.

compiler—A program that translates a program written in a high-level language, such as C++, into machine language.

complete binary tree—A binary tree of height h that is full to level $h - 1$ and has level h filled from left to right.

complete graph—A graph that has an edge between every pair of distinct vertices.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

complete interface—A class interface that allows any reasonable task to be accomplished, given the responsibilities of that class. See also *interface* and *minimal interface*.

completely balanced binary tree—A binary tree in which the left and right subtrees of any node have the same height.

composition—A *has-a* relationship between two classes such that the lifetimes of the instances of the contained class are the same as the lifetimes of an instance of the containing class. See also *aggregation*.

compound statement—A sequence of statements enclosed in braces. Also called a block.

connected component—For a graph that is not connected, a subset of the graph's vertices that a traversal visits beginning at a given vertex.

connected graph—A graph that has a path between every pair of distinct vertices.

constant reference parameter—A parameter of a function or method whose corresponding argument is passed by reference but cannot be changed during the execution of the function or method.

construction phase—The RUP phase after elaboration, during which the iterative development of the system occurs with focus on implementation and testing.

constructor—A method that initializes new instances of a class. See also *default constructor* and *parameterized constructor*.

container—An object that holds a collection of data.

container size—The number of entries a sequence container or associative container in the STL can or does hold.

containment—See *aggregation*.

copy constructor—A constructor in a C++ class that makes a copy of an object. It is invoked implicitly when you either pass an object to a method by value, return an object from a valued method, or define and initialize an object to a copy of an existing object, as in

```
LinkedBag bag2(bag1);
```

core architecture—Those software components that are central to the system.

core group—A class's core methods.

core method—A method central to the behavior of a class and that should be implemented before methods that are not core methods.

cost of an algorithm—An algorithm's time and space requirements.

cost of a path—The sum of the weights of the edges of a path in a weighted graph. Also called the weight or length of a path.

cost of a program—Factors such as the computer resources (computing time and memory) that a program consumes, the difficulties encountered by those who use the program, and the consequences of a program that does not behave correctly.

cost of a spanning tree—The sum of the weights of the edges in a weighted graph's spanning tree.

coupling—The degree to which the modules in a program are interdependent.

CRC card—Typically an index card to represent a class during its preliminary design. It lists the class's name, responsibilities, and collaborations with other classes. CRC is an abbreviation for class-responsibility-collaboration.

cubic algorithm—An algorithm whose time requirement is $O(n^3)$.

cycle—A path in a graph that begins and ends at the same vertex. See also *circuit* and *simple cycle*.

data abstraction—A design principle that separates the operations that can be performed on a collection of data from the implementation of the operations. See also *functional abstraction*.

data field—See *data member*.

data flow—The flow of data between modules.

data member—A portion of a class that stores data of a particular type.

data record—An entry in a file. A data record can be anything from a simple value, such as an integer, to multivalued data such as an employee record. See also *block* and *record*.

data structure—A construct that is defined within a programming language to store a collection of data.

data-type parameter—See *generic type*.

debugger—A software component that enables you to debug a program. A debugger allows you to set break-

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

points and watches, and to single-step through a program. See also *breakpoint, single-step*, and *watch*.

deep copy—A copy of an object that includes the data structures to which the object's data member(s) point. See also *shallow copy*.

default constructor—A constructor that has no parameters. See also *parameterized constructor*.

degenerate case—See *base case*.

depth-first search (DFS)—A graph-traversal strategy that proceeds along a path from a given vertex as deeply into the graph as possible before backtracking. That is, after visiting a vertex, a DFS visits, if possible, an unvisited adjacent vertex. If the traversal reaches a vertex that has no unvisited adjacent vertices, it backs up and then visits, if possible, an unvisited adjacent vertex. See also *breadth-first search*.

deque—A double-ended queue.

derived class—A class that inherits the members of another class called the base class. Also called a child class, descendant class, or subclass. See also *ancestor class, base class, inheritance*, and *multiple inheritance*.

descendant class—See *derived class*.

descendant of a node *n*—Any tree node along a path from *n* to a leaf of a tree.

design—The process of describing a solution to a problem that fulfills the requirements discovered during analysis. See also *analysis*.

destructor—A method that performs all tasks necessary to deallocate an object.

DFS—See *depth-first search*.

DFS spanning tree—A spanning tree formed by using a depth-first search to traverse a graph's vertices.

dictionary—An ADT whose data items are stored and retrieved according to their search-key values. Also called a table or map.

digraph—See *directed graph*.

Dijkstra's shortest path algorithm—An algorithm that solves the problem of finding the shortest path from a point in a graph (the source) to a destination.

direct access—A process that provides access to any entry in a data structure by position without the need to first access other entries in the structure. Also called random access. See also *sequential access*.

direct access file—A file whose entries are accessible by position without first accessing preceding entries within the file.

directed edge—An edge in a directed graph; that is, an edge that has a direction.

directed graph—A graph whose edges indicate a direction. Also called a digraph. See also *undirected graph*.

directed path—A sequence of directed edges that begins at one vertex and ends at another vertex in a directed graph. See also *path* and *simple path*.

disconnected graph—A graph that is not connected; that is, a graph that has at least one pair of vertices without a path between them.

distance between iterators—The number of elements or positions between the current positions of two iterators. For example, if an iterator points to the first item in a list and another iterator points to the fifth item in the same list, their distance is 4.

divide and conquer—A problem-solving strategy that divides a problem into smaller problems, each of which is solved separately.

domain expert—A person who is an expert within the domain of a problem and for whom a software system is created.

double-ended queue—A variation of the ADT queue that allows additions to and removals from its front and its back. Also called a deque.

double hashing—A collision-resolution scheme that uses two hash functions. The hash table is searched for an unoccupied location, starting from the location that one hash function determines and considering every n^{th} location, where n is determined from a second hash function.

doubly linked chain—A linked chain whose nodes each contain two pointers, one to the next node and one to the previous node.

doxygen—One of many software documentation systems that extract documentation from `javadoc`-style comments embedded in C++ source code and generate HTML-based documentation. See also *`javadoc`-style comment*.

driver—A module whose purpose is to test another module.

dummy head node—In a linked chain, a first node that is not used for data but is always present.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

dynamic allocation—The assignment of memory to a variable during program execution, as opposed to during compilation. See also *static allocation*.

dynamic binding—See *late binding*.

dynamic object—An object whose memory is allocated at execution time and remains allocated only as long as you want. Also called a dynamically allocated object. See also *static object*.

early binding—The association of a variable with its type at compilation time. Also called static binding. See also *late binding, static method,* and *virtual method*.

edge—The connection between two nodes of a graph.

elaboration phase—The RUP phase after analysis that refines the project vision by iteratively developing a core system and more accurately describing the system requirements, time estimates, and cost estimates.

element—A memory location within an array or other data structure.

empty string—A string of length zero.

empty tree—A tree with no nodes.

encapsulation—An information-hiding technique that combines data and operations to form an object.

end user—See *user*.

entry—A data item in an array or other data structure.

enumeration—A C++ construct that begins with the keyword `enum` and names integer constants called enumerators.

enumerator—A named integer constant within an enumeration.

Euler circuit—In a graph, a circuit that begins at a vertex *v*, passes through every edge exactly once, and terminates at *v*.

event—An occurrence, such as an arrival or a departure, in an event-driven simulation. See also *external event* and *internal event*.

event-driven simulation—A simulation that uses events generated by a mathematical model that is based on statistics and probability. The times of events are either read as input or computed from other event times. Because only those times at which the events occur are of interest, and because no action is required at times between the occurrence of events, the simulation can advance from the time of one event directly to the time of the next. See also *time-driven simulation*.

event list—An ADT within an event-driven simulation that keeps track of arrival and departure events that will occur but have not occurred yet.

event loop—The process within a simulation or gaming application that repeatedly determines the times at which events occur and processes the events when they do occur.

evolutionary development—Any iterative, incremental development process that uses feedback from one iteration to guide the next iteration, thus allowing the system to adapt to a changing environment.

exception—An object generated as a result of an unusual or exceptional event that occurs during the execution of a program.

exception handler—C++ code that executes to deal with an exception when one occurs. See also *`try` block* and *`catch` block*.

exhaustive search—A search strategy that must examine every item in a collection of items before it can determine that the item sought does not exist.

extensible class—A class that enables you to add capabilities to its descendants without having access to the base class's implementation. Extensible classes should define virtual methods.

external event—An event that is determined from the input data to an event-driven simulation. See also *internal event*.

external methods—Algorithms that require external files because the data will not fit entirely into the computer's main memory.

external sort—A sorting algorithm that is used when the collection of data will not fit in the computer's main memory all at once but must reside on secondary storage such as a disk. See also *internal sort*.

fail-safe programming—A technique whereby a programmer includes checks within a program for anticipated errors.

feedback—During iterative development of a solution, the comments by end users and domain experts about a partial system at the end of iteration *n* that influence the direction of iteration *n* + 1. See also *iterative development*.

Fibonacci sequence—The sequence of integers 1, 1, 2, 3, 5, . . . defined by the recurrence relationship $a_1 = 1$, $a_2 = 1$, $a_n = a_{n-1} + a_{n-2}$ for $n \geqslant 2$.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

field—(1) Another term for a class's *data member*. (2) A component of a record in a file.

FIFO—See *first in, first out*.

file—A data structure that contains a sequence of components of the same data type. See also *binary file, index file,* and *text file*.

file component—An indivisible piece of data in a file.

file variable—A C++ identifier that names a file.

file window—An imaginary marker of the current position in a file.

first child—See *oldest child*.

first in, first out (FIFO)—The behavior exhibited by a container such as a queue, whereby the removal and retrieval operations access the item that was inserted the earliest among the items currently in the container. See also *last in, first out*.

fixed size—A characteristic of a data structure whose memory allocation is determined at compilation time and cannot change during program execution. See also *static allocation*.

floating-point type—The data type of a numeric value that has both an integral and a fractional part.

folding—A hashing technique that breaks a search key into parts and combines some or all of those parts, by using an operation such as addition, to form a hash address.

free list—A list of available nodes used in some array-based implementations of an ADT.

free store—Memory assigned to a program for storing data. Also called an application heap.

friend—A class or method that can access the private and protected members of a given class.

front of a queue—The end of a queue at which items are removed and retrieved.

full binary tree—A binary tree of height h with no missing nodes. All leaves are at level h, and all other nodes each have two children.

function—A module that performs a single task.

function declaration—See *method declaration*.

function header—See *method header*.

function signature—See *method signature*.

functional abstraction—A design principle that separates the purpose and use of a module from its implementation. Also called procedural abstraction. See also *data abstraction*.

functional testing—See *closed-box testing*.

general file—See *binary file*.

general tree—A set of one or more nodes, partitioned into a root node and subsets that are general subtrees of the root.

generalization—A relationship between two classes whereby one of the classes is a specialization of the other, more general class. Implemented in C++ using inheritance.

generic type—An identifier that serves as a placeholder for an actual data type within the definition of a template or class. Classes that represent containers of objects often use a generic type so that the actual data type of the objects in the container can be specified when it is created.

glass-box testing—See *open-box testing*.

global namespace—A collection of identifiers declared outside of a namespace. The identifiers in the global namespace are known everywhere in a program.

global variable—A variable declared in the main function of a program; that is, a variable whose scope is the entire program. See also *local variable*.

grammar—The rules that define a language.

graph—A set V of vertices, or nodes, and a set E of edges that connect the vertices.

graph traversal—A process that starts at vertex v and visits all vertices w for which there is a path between v and w. A graph traversal visits every vertex in a graph if and only if the graph is connected, regardless of where the traversal starts.

growth-rate function—A function of the size of a problem, used to represent an algorithm's time requirements.

***has-a* relationship**—A relationship between two classes such that one class contains one or more instances of the other class. Also called containment. See also *aggregation*, *as-a relationship*, *composition*, and *is-a relationship*.

hash function—A function that maps the search key of a dictionary item into a location within a hash table that will contain the item.

hash table—An array that contains a dictionary's items in locations determined by a hash function.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

hashing—A technique that enables access to dictionary items in time that is relatively constant and independent of the items by using a hash function and a scheme for resolving collisions.

head—See *head pointer*.

head pointer—A pointer to the first node in a chain of linked nodes. Also called a head.

header—A mechanism that provides information about the contents of a library, including the definition of classes, declarations of functions, data types, and constants. The header is usually a file. Also called a header file or specification file. See also *implementation file*.

header file—See *header*.

heap—(1) A complete binary tree whose nodes each contain a value that is greater than or equal to the values in the node's children. See also *maxheap* and *minheap*. (2) Another term for the free store.

heap sort—A sorting algorithm that first transforms an array into a heap, then removes the entry in the heap's root (the largest entry) by exchanging it with the entry in the heap's rightmost leaf, and finally transforms the resulting semiheap back into a heap.

height-balanced tree—See *balanced binary tree*.

height of a tree—The number of nodes on the longest path from the root of the tree to a leaf.

hierarchical relationship—The "parent-child" relationship between the nodes in a tree.

highly cohesive module—A module that performs one well-defined task. See also *cohesion*.

highly coupled module—A module that is highly dependent on one or more other modules. See also *coupling* and *loosely coupled module*.

IDE—See *integrated development environment*.

identifier—In C++, a sequence of letters, digits, and underscores, beginning with either a letter or an underscore, used to name various parts of a program.

implement—(1) To create a program for an algorithm. (2) To use a data structure to realize an ADT.

implementation file—A file that contains definitions for each function and class method declared in a corresponding header.

inception phase—The first RUP phase that defines the scope of a project by creating an initial set of system requirements, determines the project's feasibility, and sets the timebox length. While defining the system requirements, this phase generates a core set of use case scenarios and identifies the aspects of the solution with the greatest risk.

increment—(1) *v*. To add 1 to a variable. (2) *n*. An amount that is added to a variable. (3) A software engineering term referring to one iteration in an iterative development process.

incremental development—See *iterative development*.

index—(1) An integral value that references the elements of an array. Also called a subscript. (2) Another name for an index file.

index file—A data structure whose entries—called index records—are used to locate items in an external file. Also called the index.

index record—An entry in an index file that points to a record in the corresponding external data file. This entry contains a search key and a pointer.

induction—See *mathematical induction*.

inductive conclusion—In an inductive proof, the validity of the property $P(k + 1)$ if the inductive hypothesis ("if $P(k)$ is true for any $k \geq 0$") is true. See also *inductive step*.

inductive hypothesis—In an inductive proof, the assumption that a property $P(k)$ is true when $k \geq 0$. See also *inductive step*.

inductive proof—A proof that uses the principle of mathematical induction.

inductive step—In an inductive proof, the step that assumes an inductive hypothesis ("if $P(k)$ is true for any $k \geq 0$") and uses it to demonstrate the inductive conclusion ("then $P(k + 1)$ is true").

infix expression—An algebraic expression in which every binary operator appears between its two operands. See also *postfix expression* and *prefix expression*.

information hiding—A process that hides certain implementation details within a module and makes them inaccessible from outside the module.

inheritance—A relationship among classes whereby a class derives properties from a previously defined class. See also *derived class* and *multiple inheritance*.

initializer—A functional notation used in the implementation of a class constructor that consists of a data member name followed by its initial value enclosed in parentheses. The first (or only) initializer follows a colon at the end of

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

the constructor's header. If you write more than one initializer, you separate them with commas.

inorder successor of a node *n*—The inorder successor of *n*'s data item. The inorder successor is in the leftmost node of *n*'s right subtree.

inorder successor of *x*—The value in a search tree that an inorder traversal visits immediately after *x*.

inorder traversal—A traversal of a binary tree that processes (visits) a node after it traverses the node's left subtree, but before it traverses the node's right subtree. See also *level-order traversal*, *postorder traversal*, and *preorder traversal*.

input iterator—An iterator that has the following operations: copy/assign (=), increment (++), equality/inequality (==, !=), and access entry (*).

insertion sort—A way of sorting the entries in an array that involves dividing, or partitioning, the array into two groups, one of which is sorted and the other unsorted. One by one, an entry in the unsorted portion is inserted into its proper position within the sorted portion until the entire array is sorted.

instance—An object that is the result of either declaring a variable of a particular class type or using the `new` operator with a particular class constructor and a pointer to that class type.

instantiate—To create an object.

integral promotion—The conversion of any values of type `char` or `short` to `int` during the evaluation of an expression. Also, the conversion of any enumerator value to `int`, if `int` can represent all the values of that particular `enum`.

integral type—A boolean, character, or integer type.

Integrated development environment (IDE)—An application that provides programmers with tools such as an editor, a compiler, and a debugger, to facilitate coding.

integration testing—Testing interactions among the modules in a solution. See also *acceptance testing*, *beta testing*, *closed-box testing*, *open-box testing*, *system testing,* and *unit testing.*

interaction diagram—See *sequence diagram*.

interface—The communication mechanisms between modules or systems. See also *client interface*.

internal event—An event that is determined by a computation within an event-driven simulation. See also *external event*.

internal node of a tree—A node that is not a leaf.

internal sort—A sorting algorithm that requires the collection of data to fit entirely in the computer's main memory. See also *external sort*.

invariant—An assertion that is always true at a particular point in an algorithm or program.

***is-a* relationship—**A relationship between classes whereby one class is a special case of another class. You implement an *is-a* relationship by using public inheritance. See also *as-a relationship* and *has-a relationship*.

iteration—(1) A process that is repetitive. (2) Each repetition in a repetitive process. See also *iterator*. (3) A single pass through a loop. (4) A short, fixed-duration process that develops a small portion of a system. At the end of each iteration, the partial system should be functional and completely tested. This partial system is used to generate feedback to guide the next iteration. See also *iterative development*.

iterative development—A software development process that progresses through many short, fixed-length iterations, with each iteration cycling through the analysis, design, implementation, testing, and integration of a small portion of the problem domain. Through iterative development, the system comes into being incrementally. See also *feedback* and *iteration*.

iterative solution—A solution that involves repetition using loops.

iterator—An object that provides access to either the next or the previous item within a container. An iterator provides a way to cycle through the objects in the container.

iterator category tag—An identifier to indicate the desired functionality of an iterator .

`javadoc`-style comment—A special multiline comment beginning with a `/**` and ending with a `*/`. Used by software documentation systems to generate HTML-based documentation from the contents of the comment. See Appendix I.

key—(1) Another name for *search key*. (2) The portion of an index record that corresponds to the search key in a record in an external data file.

key independent—In a collision-resolution scheme for hashing, the characteristic of a probe sequence that does not depend on the search key of the item in question.

keyword—An identifier within a programming language such as C++ that is reserved and has a special meaning.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

language—A set of strings of symbols that adhere to the rules of a grammar.

last in, first out (LIFO)—The behavior exhibited by a container such as a stack, whereby the removal and retrieval operations access the item that was inserted most recently among the items currently in the container.

late binding—The association of a variable with its data type during program execution. Also called dynamic binding. See also *early binding, static method,* and *virtual method*.

leaf—A tree node with no children.

left child of a node *n*—A node directly below and to the left of node *n* in a binary tree.

left subtree of a node *n*—The left child of node *n* plus its descendants in a binary tree.

length of a path—See *cost of a path*.

level of a node—The root of a tree is at level 1. If a node is not the root, then its level is 1 greater than the level of its parent.

level-order traversal—A traversal of a tree that processes (visits) nodes one level at a time, from left to right, beginning with the root. See also *inorder traversal*, *preorder traversal*, and *postorder traversal*.

library—A collection of classes, templates, or other software components that can be used by programmers in their software applications.

life cycle—See *software life cycle*.

lifeline—A representation in a UML sequence diagram of an object's lifetime.

LIFO—See *last in, first out*.

linear algorithm—An algorithm whose time requirement is O(*n*).

linear chain—See *linear linked chain*.

linear implementation—An implementation, either array-based or link-based, in which each element other than the first and last ones has one successor and one predecessor.

linear linked chain—A linked chain that is not circular.

linear probing—A collision-resolution scheme that searches the hash table sequentially, starting from the original location specified by the hash function, for an unoccupied location.

link-based implementation—An implementation of an ADT that uses pointers to organize its elements.

linked chain—See *chain*.

list—A container whose elements are referenced by their position. See also *sorted list*.

literal constant—A specific value such as `3`, `1.2`, and `'A'` in a C++ program.

load factor—A measure of the relative fullness of a hash table, defined as the ratio of the table's current number of items to its maximum size.

local environment—In a function or method, the local variables, copies of the value arguments, the return address in the calling routine, and the value of the function or method itself.

local identifier—An identifier whose scope is the block that contains its declaration.

local variable—A variable declared within a function or method and available only within that module's body. See also *global variable*.

loop invariant—An assertion that is true before and after each cycle of a loop within an algorithm or program.

loosely coupled module—A module that has little dependence on one or more other modules. See also *coupling* and *highly coupled module*.

machine language—A language composed of the fundamental instructions that a computer can execute directly.

macro—Instructions or program statements that are given a name. The name serves as a representation of those statements when it is used subsequently. See also *preprocessor* and *preprocessor directive*.

main success scenario—A description within a use case of the UML of how the proposed system satisfies the goals of the user under normal circumstances.

map—See *dictionary*.

mathematical induction—A technique for proving properties that involve nonnegative integers. Starting from a base case, you show that if a property is true for an arbitrary nonnegative integer *k*, then the property is true for the integer *k* + 1.

maxheap—Another name for a heap. See also *minheap*.

median-of-three pivot selection—A pivot selection scheme in the quick sort that uses the median of the first, middle, and last entries in the array as the pivot.

member—A component of a class that is either data or a method. See also *data member* and *method*.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

member function—See *method*.

memory leak—Inaccessible memory that was dynamically allocated but not deallocated and has no pointer to it.

merge sort—A sorting algorithm that divides an array into halves, sorts each half, and then merges the sorted halves into one sorted array. Merge sort can also be adapted for sorting an external file.

message—A request, in the form of a method call, that an object perform an operation.

message expression—A label on a message arrow within a UML sequence diagram. The message expression ultimately represents a method and its parameters.

method—A function that is a member of a class. Also called a member function.

method declaration—A method's signature preceded by its return type and followed by a semicolon. Also called a function declaration.

method definition—The implementation of a method.

method header—The first line of a method definition. A method header is the same as the method's declaration, minus the trailing semicolon. Also called a function header or prototype.

method signature—A method's name; the number, order, and types of its parameters; and any qualifiers that might apply, such as `const`. The signature does not include the method's return type. Also called a function signature.

minheap—A complete binary tree whose nodes each contain a value that is less than or equal to the values in the node's children. See also *heap* and *maxheap*.

minimal interface—A class interface that declares a method if and only if that method is essential to the accomplishment of the class's responsibilities. See also *interface* and *complete interface*.

minimum spanning tree—A graph's spanning tree for which the sum of its edge weights is minimal among all spanning trees for the graph.

model—(1) A description of a solution to a problem used in object-oriented design. (2) A program that simulates the behavior of a real-world system.

modular program—A program that is divided into isolated components, or modules, that have clearly defined purposes and interactions.

module—An individual component of a program, such as a function, a method, a class, a group of functions or classes, or other block of code.

multigraph—A graphlike structure that allows duplicate edges between its vertices.

multiple indexing—A process that uses more than one index file to an external data file.

multiple inheritance—A relationship among classes whereby a class derives properties from more than one previously defined class. See also *derived class*.

multiplicity—The number of objects involved at each end of a relationship in a UML class diagram.

mutator—Another term for a *mutator method*.

mutator method—A method of a class that changes the value of a data member. Also known as a mutator. See also *accessor method*.

namespace—A mechanism in C++ for logically grouping declarations of data and methods under a common name.

namespace indicator—The name of a namespace followed by `::`. You write a namespace indicator immediately before the name of a method to tell the compiler that the method is a part of a class's namespace. See also *scope resolution operator*.

***n*-ary tree**—A set of nodes that is either empty or partitioned into a root node and *n* possibly empty sets that are *n*-ary trees. Thus, each node can have no more than *n* children. See also *binary tree*.

new operator—An operator that allocates memory for a variable on the free store, which is often called the application heap.

node—An element in a linked chain, tree, or graph that usually contains both data and a pointer to the next node in the data structure.

nontext file—See *binary file*.

O—See *Big O notation*.

object—An instance of a class. An object has a set of characteristics and behaviors described by the class.

object-oriented analysis (OOA)—A process that investigates a problem to discover and document its nature, its scope, and the requirements of a solution in terms of objects that interact within the problem's domain. See also *analysis* and *object*.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

object-oriented analysis and design (OOA/D)—See *object-oriented analysis* and *object-oriented design*.

object-oriented design—A process that describes a solution to a problem in terms of objects and their collaborations with one another. See also *design* and *object*.

object-oriented programming (OOP)—A software engineering technique that views a program as a collection of components called objects that interact. OOP embodies three fundamental principles: encapsulation, inheritance, and polymorphism.

object type compatibility—A characteristic of objects that enables you to use an instance of a derived class instead of an instance of a base class, but not the converse. The object type of an argument in a call to a function or method can be a descendant of the corresponding parameter's object type.

O(*f* (*n*))—Order *f* (*n*). See *Big O notation* and *order of an algorithm*.

oldest child—The leftmost child of a node in a general tree. Also called the first child.

OOA—See *object-oriented analysis*.

OOA/D—See *object-oriented analysis* and *object-oriented design*.

OOD—See *object-oriented design*.

OOP—See *object-oriented programming*.

open—A process that prepares a file for either input or output and positions the file window. A state of readiness for I/O.

open addressing—A category of collision-resolution schemes in hashing that probe for an empty, or open, location in the hash table in which to place the item. See also *double hashing, linear probing,* and *quadratic probing*.

open-box testing—The testing of a method knowing its implementation. Also called glass-box testing. See also *acceptance testing*, *beta testing*, *closed-box testing*, *integration testing*, *system testing,* and *unit testing.*

operation—The software engineering term for a behavior of an object. The UML term for a method of a class. See also *behavior* and *method*.

operation contract—A specification of how a module can be used and what limitations it has.

operator method—A method that overloads an operator, thereby giving the operator meaning when used with objects of the method's class. See also *overloaded operator*.

order *f* (*n*)—O(*f* (*n*). See *Big O notation* and *order of an algorithm*.

order of an algorithm—An algorithm's time requirement as a function of the problem size. An algorithm A is order $f(n)$ if constants K and n_0 exist such that A requires no more than $K \times f(n)$ time units to solve a problem of size $n \geq n_0$. See also *Big O notation*.

order-of-magnitude analysis—An analysis of an algorithm's time requirement as a function of the problem size. See also *order of an algorithm*.

overload—To define two or more methods in either the same class or classes related by inheritance that have the same name but different signatures.

overloaded operator—An operator with multiple meanings, each of which is determined by the context in which the operator appears. See also *operator method* and *overload*.

override—To define a method in a derived class and give it the same name and parameter declarations as a virtual method in the base class. That is, to redefine within a derived class a virtual method of the base class. The method is then polymorphic. See also *redefine*.

palindrome—A character string that reads the same from left to right as it does from right to left; for example, "deed."

parameter—An identifier that appears in the declaration of a function or method and represents the argument that the calling program will pass to the function or method. See also *argument, reference parameter,* and *value parameter*.

parameterized constructor—A constructor that has parameters. See also *default constructor*.

parent class—See *base class*.

parent of a node *n*—The node directly above node *n* in a tree.

partition—(*v*.) To divide a data structure such as an array into segments. (*n*.) A segment of a data structure.

pass by constant reference—A technique of passing an object as input to a method as a constant reference argument. Such an argument has the efficiency of a reference argument—since it is not copied—and the protection of a value argument—since the method cannot change its val-

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

ue. This technique is particularly useful for large objects. See also *reference parameter* and *value parameter*.

pass by reference—The act of passing an argument to a reference parameter of a function or method.

pass by value—The act of passing an argument to a value parameter of a function or method.

path—A sequence of edges in a graph that begins at one vertex and ends at another vertex. Because a tree is a special graph, you can have a path through a tree. See also *directed path* and *simple path*.

perfect hash function—An ideal hash function that maps each search key into a unique location in the hash table. Perfect hash functions exist when all possible search keys are known.

pivot—A central element in an algorithm. For example, the quick sort algorithm partitions an array about a particular item called the pivot.

pivot selection—The act of choosing a pivot for an algorithm. Various schemes exist for doing so. See also *median-of-three pivot selection*.

planar graph—A graph that can be drawn in a plane in at least one way so that no two edges cross.

pointer—(1) A pointer variable in C++. (2) Generically, an element that references a memory cell. (3) An indicator, such as an integer, to an element within a data structure. For example, an index record, which points to a data record in an external data file, contains such an indicator, namely, the number of the block that contains the data record.

pointer variable—A C++ variable that references a memory cell. Also called a pointer.

polymorphic—The ability of a variable name to represent, during program execution, instances of different but related classes that descend from a common base class. See also *polymorphism*.

polymorphism—A feature of object-oriented programming whereby the correct version of a method is determined during program execution instead of during compilation. See also *polymorphic*.

pop—To remove an item from a stack.

position-oriented ADT—An ADT whose operations involve the positions of its items. See also *value-oriented ADT*.

postcondition—A statement of the conditions that exist at the end of a module's execution.

postfix expression—An algebraic expression in which every binary operator follows its two operands. See also *infix expression* and *prefix expression*.

postorder traversal—A traversal of a binary tree that processes (visits) a node after it traverses both of the node's subtrees. See also *inorder traversal, level-order traversal,* and *preorder traversal*.

precondition—A statement of the conditions that must exist at the beginning of a module so that the module will work correctly.

predecessor—(1) In a linked chain, the predecessor of node *x* is the node that points to *x*. (2) In a directed graph, vertex *x* is a predecessor of vertex *y* if there is a directed edge from *x* to *y*, that is, if y is adjacent to *x*. See also *successor*.

prefix expression—An algebraic expression in which every binary operator precedes its two operands. See also *infix expression* and *postfix expression*.

preorder traversal—A traversal of a binary tree that processes (visits) a node before it traverses both of the node's subtrees. See also *inorder traversal, level-order traversal,* and *post-order traversal*.

preprocessor—A software component that processes source code before it is compiled. The preprocessor is able to include other files that are named in `#include` statements, expand any macros that are defined in a `#define` statement, and either include or exclude source statements that are between `#if`—or `#ifndef`—and `#endif`. See also *macro* and *preprocessor directive*.

preprocessor directive—A command to the preprocesssor that begins with the symbol `#`. One of `#define`, `#endif`, `#if`, `#ifndef`, and `#include`. See also *macro* and *preprocessor*.

primary clustering—The forming of groups, or clusters, of occupied and consecutive locations within a hash table due to linear probing. See also *clustering* and *secondary clustering.*

primitive data type—See *simple data type*.

priority queue—A container that orders its items by a priority value. The first item removed is the one having the highest priority value.

priority value—A value assigned to each entry in a priority queue to indicate the entry's priority.

private—The portion of a module that is hidden from other modules.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

private inheritance—A form of inheritance whereby the public and protected members of a base class are private members of the derived class.

private section—The portion of a class that is accessible only by methods and friends of the class.

probe sequence—The sequence of locations in the hash table that a collision-resolution scheme examines.

problem solving—The entire process of taking the statement of a problem and developing a computer program that solves that problem.

procedural abstraction—See *functional abstraction*.

proof by induction—See *mathematical induction*.

protected inheritance—A form of inheritance whereby the public and protected members of a base class are protected members of the derived class.

protected section—The portion of a class that is accessible by methods of both the class and a derived class.

prototype—See *method header*.

public—The portion of a module that is accessible by other modules.

public inheritance—A form of inheritance whereby the public and protected members of a base class remain, respectively, public and protected members of the derived class.

public section—The portion of a class that is accessible by any user of the class, including methods of other classes.

pure virtual method—A virtual method with an undefined body, written as `virtual` *method_header* `= 0` within the class definition.

push—To add an item to a stack.

quadratic algorithm—An algorithm whose time requirement is $O(n^2)$.

quadratic probing—A collision-resolution scheme that searches the hash table for an unoccupied location beginning with the original location that the hash function specifies and continuing at increments of 1^2, 2^2, 3^2, and so on.

qualify—In C++, to precede an identifier `x` with an identifer `y` and a separator such as a period or two colons. For example, you qualify `y` with `x` by writing `x.y`.

queue—A container whose oldest item—that is, the one that has been in the queue the longest— is the first item removed or retrieved. This property is called first in, first out, or simply FIFO. Items enter a queue at its back and leave at its front.

quick sort—A sorting algorithm that partitions an array's entries around a pivot p to generate two smaller sorting problems: Sort the array's left section, whose entries are less than or equal to p, and sort the array's right section, whose entries are greater than or equal to p.

radix sort—A sorting algorithm that treats each data entry as a character string and repeatedly organizes the data into groups according to the i^{th} character in each entry.

random access—See *direct access*.

range query—An operation that retrieves all dictionary items whose search keys fall within a given range of values.

Rational Unified Process (RUP)—An iterative software development process. See *iterative development*.

rear of a queue—See *back of a queue*.

recognition algorithm—An algorithm, based on a language's grammar, that tests whether a given string is in the language.

record—A group of related items, called fields, that are not necessarily of the same data type. See also *data record*.

recurrence relation—A mathematical formula that generates the terms in a sequence from previous terms.

recursion—A process that solves a problem by solving smaller problems of exactly the same type as the original problem.

recursive call—A call within a function or method to the function (method) itself.

red-black tree—A representation of a 2-3-4 tree as a binary tree whose nodes have red and black child pointers.

redefine—To define a method in a derived class and give it the same name and parameter declarations as a method in the base class. Although redefining nonvirtual methods is possible, doing so is not recommended. See also *override*.

reference parameter—A parameter that behaves as an alias of its corresponding argument. Any change that the function or method makes to a reference parameter changes the corresponding argument in the calling module. You designate a reference parameter by following its data type with `&`. See also *pass by constant reference* and *value parameter*.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

rehash—To compute new hash indices for the entries in a hash table after expanding the table's size when it becomes too full.

requirements—A description of what a solution must do, without imposing a particular design or implementation of that solution. The requirements of a solution come from an analysis of the problem. See also *analysis*.

resize—To move the entries from an array to a new, larger or smaller array and then give the new array the name of the original array. The apparent effect is to either expand or reduce the size of the original array.

responsibilities—Those things that an object needs to remember (or keep track of), and those things that an object needs to do for other objects.

right child of a node *n*—A node directly below and to the right of node *n* in a tree.

right subtree of a node *n*—The right child of node *n* plus its descendants in a tree.

rightward drift—(1) In an array-based implementation of a queue, the problem of the front of the queue moving toward the end of the array. (2) In a C++ program, the problem of nested blocks bumping against the right-hand margin of the page.

robust—A characteristic of software that performs well even under unusual or unplanned conditions.

root—The only node in a tree with no parent.

rotation—An operation used to maintain the balance of a red-black or AVL tree.

runtime—The execution phase of a program. The time during which a program's instructions execute. See also *compile time*.

run-time stack—An area of an application's memory that stores the activation records generated when functions or methods are called.

RUP—See *Rational Unified Process*.

safe and secure programming—A methodology that extends fail-safe programming by validating any input data and arguments to a function or method, by eliminating a function's (method's) side effects, and by making no assumptions about the actions of clients and users.

scenario—A textual story within a use case that describes a solution to a problem or part of a problem. See also *alternate scenario* and *main success scenario*.

scope of an identifier—The part of a program in which an identifier has meaning.

scope resolution operator—The C++ operator `::`. This operator specifies the scope within which a C++ entity is declared. The scope is typically a namespace or a class. The C++ entity is usually a method, a user-defined type, a constant, or a variable. See also *namespace indicator*.

search—A process that locates a certain item in a collection of items.

search key—The part of an item that identifies it within a collection of items and that a search algorithm uses to locate the item. Also called a key.

search tree—A tree whose organization facilitates the retrieval of its items. See also *2-3 tree, 2-3-4 tree, AVL tree, binary search tree, B-tree of degree m, and red-black tree.*

secondary clustering—A phenomenon that occurs during hashing when quadratic probing is used to resolve a collision, because the same probe sequence is followed for the entries that hash into the same table location. The result is a delay in resolving the collision. See also *clustering* and *primary clustering.*

selection sort—A sorting algorithm that selects the largest (smallest) item and puts it in its correct place, then selects the next largest (smallest) item and puts it in its correct place, and so on.

semiheap—A complete binary tree in which the root's left and right subtrees are both heaps.

separate chaining—A collision-resolution scheme that uses an array of linked chains as a hash table. The i^{th} chain contains all items that map into location i.

sequence diagram—A diagram in the UML that shows how two or more objects interact within a single use case scenario.

sequential access—A process that stores or retrieves entries in a data structure one after another, starting at the beginning. See also *direct access*.

sequential access file—A file whose data must be processed sequentially. That is, to process the data stored at a given position, you must advance the file window beyond all the data that precedes it.

sequential search—An algorithm that locates an item in a collection by examining items in order, one after another, beginning with the first item.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

set—(*n*.) A bag whose entries are unique, that is, cannot be duplicate. (*v*.) To assign a value to a variable or to change an object's data. See also *mutator method*.

shallow copy—A copy that does not include any data structures to which the object's data members might point. See also *deep copy*.

Shell sort—A way of sorting the entries in an array using a modified insertion sort. Rather than always exchanging adjacent items, the Shell sort arranges the array so that every h^{th} item begins a sorted subarray in a decreasing sequence of values. Ultimately, if h is 1, the entire array will be sorted.

short-circuit evaluation—A way of evaluating a boolean expression involving either `&&` or `||` that does not evaluate this operator's second operand if the expression's value is evident from the value of the first operand.

shortest path—In a weighted graph, the path between two given vertices that has the smallest sum of its edge weights.

siblings—Tree nodes that have a common parent.

side effect—(1) A change to a variable that exists outside of a function or method and that is not passed as an argument. (2) An occurrence that is not specified by a module.

signature—See *method signature*.

simple cycle—A cycle in a graph that does not pass through a vertex more than once.

simple data type—A data type that is not aggregate, such as `int` or `double`. Also called a primitive data type.

simple path—A path in a graph that does not pass through a vertex more than once. See also *directed path*.

simulation—A technique for modeling the behavior of both natural and artificial systems. Generally, its goal is to generate statistics that summarize the performance of an existing system or to predict the performance of a proposed system. A simulation reflects the long-term average behavior of a system rather than predicting occurrences of specific events.

single-step—Executing a program one statement at a time, after a debugger has paused at a breakpoint. See also *breakpoint* and *watch*.

singly linked chain—See *chain*.

software engineering—A branch of computer science that provides techniques to facilitate the development of computer programs.

software life cycle—The phases of software development from conception to replacement by another system. In the Rational Unified Process, the phases are inception, elaboration, construction, and transition. See also *iterative development*.

solution—A computer program consisting of a system of interacting objects. Each object is responsible for some aspect of the solution.

sort key—The part of an item that a sorting algorithm uses to determine the item's order within a collection of items.

sorted list—A container that maintains its entries in sorted order and retrieves them by their position number within the list. See also *list*.

sorted order—The order of a collection of data that is either ascending or descending.

sorted run—Sorted data that is part of an external sort.

sorting—A process that organizes a collection of data into either ascending or descending order. See also *external sort* and *internal sort.*

source code—See *source program***.**

source-inclusion facility—A facility that automatically places the contents of a file at a specified point in a program before the program is compiled. The `#include` statement provides this facility in C++.

source program—A program written in a programming language that needs to be compiled. For example, a C++ program. Also called source code.

spanning tree—A subgraph of a connected, undirected graph that contains all of the graph's vertices and enough of its edges to form a tree. See also *BFS spanning tree* and *DFS spanning tree*.

specification file—See *header.*

stable sort—A sorting algorithm that does not change the relative order of entries that are equal. For example, if entries x and y are equal and x appears before y in a collection of data, a stable sorting algorithm will leave entry x before entry y after sorting the data.

stack—A container whose most recently inserted item is the first item removed or retrieved. This property is called last in, first out, or simply LIFO. Items enter and leave a stack at its top.

Standard Template Library (STL)—A library containing template classes for many commonly used ADTs, includ-

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

ing lists, stacks, and queues. It also contains template functions for common algorithms such as sorting.

static allocation—The assignment of memory to a variable during compilation, as opposed to during program execution. See also *dynamic allocation*.

static binding—See *early binding*.

static diagram—See *class diagram*.

static method—A method whose body is determined (bound to the object) at compilation time. See also *early binding, late binding, and virtual method*.

static object—A statically allocated object. An object whose memory is allocated at compilation time and remains allocated for the duration of the program's execution. See also *dynamic object.*

static storage—Memory that the operating system reserves for global variables and static variables when the program begins execution.

stereotype—A label, such as «`create`», in a UML diagram that identifies a special characteristic of an element. Stereotypes are delimited by guillemets (« »). Stereotypes can be applied to many elements in all types of UML diagrams.

STL—See *Standard Template Library*.

stream—An object that represents a flow of data.

stream variable—A program variable that references a stream and is used to access a file.

string—A sequence of characters. A C++ string is an object of type `string`.

structure chart—An illustration of the hierarchy of modules that solve a problem.

stub—An incomplete method that acknowledges that it has been called and perhaps performs some simple task that aids the testing of other methods.

subclass—See *derived class*.

subgraph—A subset of a graph's vertices and edges.

subscript—An integral value that references the elements of an array. Also called an index.

subtree—Any node in a tree, together with all of the node's descendants.

subtree of a node *n*—A tree that consists of a child of *n* and the child's descendants.

successor—(1) In a linked chain, the successor of node *x* is the node to which *x* points. (2) In a directed graph, vertex *y* is a successor of vertex *x* if there is a directed edge from *x* to *y*, that is, if *y* is adjacent to *x*. See also *predecessor*.

superclass—See *base class*.

symmetric matrix—An *n*-by-*n* matrix *A* whose elements satisfy the relationship $A_{ij} = A_{ji}$.

system testing—The testing of a solution in the environment in which it was designed to work. See also *acceptance testing*, *beta testing*, *closed-box testing*, *integration testing*, *open-box testing*, and *unit testing.*

table—See *dictionary*.

tag—A textual indicator of an element in a `javadoc`-style comment. Tags begin with the symbol `@`.

tail —(1) The end of a list. (2) Another name for a tail pointer.

tail pointer—A pointer to the last node in a linked chain. Also called a tail.

tail recursion—A type of recursion in which the recursive call is the last action taken.

target —The item sought by a search algorithm.

template—See *class template*.

testing—See *acceptance testing*, *beta testing*, *closed-box testing*, *integration testing*, *open-box testing*, *system testing,* and *unit testing.*

text file—A file of characters that are treated as if they are organized into lines. See also *binary file*.

text storage—See *code storage*.

throw—To indicate an exception.

timebox—The fixed duration of each iteration in the Rational Unified Process for software development. See also *iteration.*

time-driven simulation—A simulation in which the time of an event, such as an arrival or departure, is determined randomly and compared with a simulated clock. See also *event-driven simulation*.

top of a stack—The end of a stack at which items are inserted, retrieved, and removed.

topological order—A sequence of vertices in a directed graph without cycles such that vertex *x* precedes vertex *y* if there is a directed edge from *x* to *y* in the graph. A topological order is not unique, in general.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved

topological sorting—In a directed graph without cycles, the process of arranging the vertices into a topological order.

transition phase—The fourth phase of the RUP, during which beta testing and deployment of the system occur.

traversal—An operation that visits each entry in a container.

traverse—To visit each entry in a container.

tree—(1) A container that provides a hierarchical, or nonlinear, arrangement of its entries. (2) A connected, undirected graph without cycles. See also *binary tree* and *general tree*.

tree sort—A way of sorting the entries in an array using a binary search tree.

`try` block—A group of C++ statements that can throw an exception. A `try` block is followed immediately by one or more `catch` blocks.

type compatibility—See *object type compatibility*.

UML—See *Unified Modeling Language*.

unary operator—An operator that requires only one operand, for example, the – in –5. See also *binary operator*.

undirected graph—A graph that has at most one edge between any two vertices and whose edges do not indicate a direction. See also *directed graph*.

Unified Modeling Language (UML)—A modeling language used to express object-oriented designs in terms of diagrams and text-based descriptions. See also *class diagram*.

unit testing—The testing of an individual module in a solution. See also *acceptance testing*, *beta testing*, *closed-box testing*, *integration testing*, *open-box testing*, and *system testing*.

use case—A set of textual scenarios, or stories, that describe a solution to a problem and are a part of the analysis of the problem. See also *analysis*.

user—A person who will use a software system to get work done. Also known as an end user.

user interface—The portion of a program that provides for user input or control.

valued function or method—A function or method that returns a value. See also *void function or method.*

value-oriented ADT—An ADT whose operations involve the values of its data items. See also *position-oriented ADT*.

value parameter—A parameter whose value is initially the value of its corresponding argument. Any change that the function or method makes to a parameter is not reflected in the corresponding argument in the calling module. The default when you do not follow the parameter's data type with `&`. See also *reference parameter*.

vector—An object of the STL class `vector`. A vector behaves like a high-level array.

vertex—A node in a graph.

virtual method—A method of a class that a derived class can override, that is, redefine. The body of a virtual method is determined at execution time. See also *early binding*, *late binding*, *static method*, and *virtual method table*.

virtual method table (VMT)—The table that exists for an object type that defines a virtual method. For every virtual method in the object, the object's VMT contains a pointer to the actual instructions that implement the method. This pointer is established by the constructor during program execution.

visit—The act of processing an item during a traversal of a container or a data structure.

VMT—See *virtual method table*.

void function or method—A function or method that does not return a value. See also *valued function or method*.

watch—A debugger tool that allows you to choose variables in a program and observe their values as the program executes. See also *debugger*.

waterfall method—An outdated development process in which development moves sequentially through phases such as requirements analysis, design, implementation, testing, and deployment.

weight of an edge—The numeric label on an edge in a weighted graph.

weight of a path—See *cost of a path*.

weighted graph—A graph whose edges are labeled with numeric values.

white-box testing—See *open-box testing*.

worst-case time—A determination of the maximum amount of time that a given algorithm requires to solve problems of size *n*. See also *average-case time*.

© 2013 Pearson Education, Inc., Upper Saddle River, NJ. All Rights Reserved