1. **Abstract--**A concept, thought, or idea apart from any particular instances or material objects. In computer science an organization of data, a manipulation of data, an algorithm, etc. apart from any particular programming language implementation of that thing.

2. **Abstract Data Type--**A data type whose properties (domain and operations) are specified independently of any particular implementation. Examples: List, Stack, Queue, Tree, Graph, Set.

3. **Abstraction--**Formation of an idea, such as the qualities or properties of a thing, separate from any particular instances of a thing or material object. In computer science, a model of a complex system that includes only the details essential to the perspective of the viewer of the system.

4. **Acceptance Testing--**Tests done before a software system is delivered to the customer or placed on the market. Designed to determine if the software meets all the requirements.

5. **Adjacency Matrix--**A way of implementing a graph in computer memory in which the nodes are stored as an array of structures, and the links are defined in a two-dimensional array. Each row of the 2-D array represents the links from one of the vertices and the columns represent the "links to" vertices. This implementation is used when the number of vertices that will be in the graph is not known prior to run time.

6. **Adjacency Set--**A way of implementing a graph in computer memory in which the nodes are stored as a linked list of structures, each of which contains a pointer to a linked list of "link" structures defining the edges from this node to others in the graph. This implementation is used when the number of vertices that will be in the graph is not known prior to run time.

7. **Adjacent Vertices**--Two vertices in a graph are said to be adjacent if there is an edge connecting the two.

8. **Algorithm**--an outline of the procedure for solving a problem. This includes (1) the actions which are to be taken, and (2) the order in which the actions are to be performed.

9. **Analysis of Algorithms**--A measure of the amount of resources necessary to execute a section of code. The efficiency or running time of an algorithm stated as a function relating the input size/length to the number of steps (time complexity) or storage locations (space complexity).

10. **AVL Tree**--A binary tree is an AVL tree if it meets the following criteria: (1) For any node in the tree the height (longest distance from root to leaf) of its left sub-tree is the same as the height of its right sub-tree, or it differs at most by 1, (2) The height of an empty tree is defined as -1. Developed by two Russian mathematicians, Adelson-Velskii and Landis.

11. **Big-O Notation**--A measure used to express the computing time (complexity) of a piece of code relative to the size of the inputs. Examples: O(1)=Constant, O(log n)=Logarithmic, O(n)=Linear, O(n log n)=n Log n, $O(n^2)$=Quadratic, $O(n^3)$=Cubic, $O(2^n)$=Exponential, and $O(10^n)$=Exponential.

12. **Binary Tree**--A tree structure in which each node is an empty tree or a node that has left and/or right children which are binary trees. The key in the parent node is greater than the key in the left child and less than the key in the right child.

13. **Black-box Testing--**Testing a program or function based solely on the defined inputs and expected results without considering what is happening inside the code.

14. **Bottom Up Program Design--**The process of program design in which the problem is defined in terms of low level simple objects that interact in some way. Design progresses from the simple objects and how they interact upward to more complex interactions. Also called **object oriented programming**. Focus is on the data objects in the software system. See Top Down Program Design.

15. **Call By Reference--**A function calling method in which the address or reference to the actual arguments are passed to a function. In C Call By Reference can be achieved by passing the addresses of variables or the values stored in pointers. In C++ functions can be defined that are Call By Reference functions. See the page on pointers for details on how to define a Call By Reference function. See also Call By Value.

16. **Call By Value--**A function calling method in which only copies of the values stored in arguments to the function are passed into the function. All functions in C are Call By Value functions. See Call By Reference.

17. **Cardinallity--**The number of items in a set.

18. **Class--**A structured type in a programming language containing variables and the functions which operate on those variables. Used to represent an abstract data type. See Object Oriented Programming.

19. **Collision--**In hashing, when two keys hash to the same index in a hash table.

20. **Collision Resolution Techniques**--A method for handling collisions in a hash table, e.g. open addressing with linear probing, open addressing with double hashing, chaining, and buckets.

21. **Complete Binary Tree**--Trees with all their leaves on one level, or two adjacent levels with the bottom most leaves as far left as possible. A full binary tree is a complete binary tree.

22. **Component (base) type**--In a set, the data type of the items composing the set.

23. **Connected Component**--In an undirected graph, a set of vertices (which may not necessarily be all the vertices in the graph) in which for any vertex there is a path to every other vertex in the set.

24. **Connected Vertices**--In an undirected graph, two vertices in which there exists, in the graph, a path between them.

25. **Constructors**--Functions that create an abstract data type. Examples: **int X** or **new node()**. Also the "constructor" function(s) in a C++ class.

26. **Cycle**--In a graph, a path greater than one that begins and ends on the same vertex. See **Simple Cycle**.

27. **Data Abstraction**--The separation of a data type's logical properties from its' implementation. Another term for Data Encapsulation.

28. **Data Encapsulation**--The separation of the representation of data from the applications that use the data at a logical level. A programming language feature that enhances information hiding. Another term for Data Abstraction. See Information Hiding.

29. **Degree of a Vertex--**In a graph, the number of edges for which a given vertex is one of the end points of the edge, i.e. how many other vertices a given vertex is connected to. See also **In Degree, Out Degree, Predecessors of a Vertex** and **Successors of a Vertex**.

30. **Difference**--A binary set operation that returns a set made up of all items that are in the first set but not in the second set.

31. **Directed Graph--**A graph in which the paths (edges) between vertices go in only one direction. This is indicated in diagrams by making the lines connecting vertices into arrows indicating the direction of movement.

32. **Domain--**The set or range of data that an abstract data type acts on. Example: the domain of int data types is all whole numbers. See Abstract Data Type.

33. **Driver--**A special function written to use as a testing function. It passes known inputs to selected functions and reports the returned values. Drivers are used to test lower level functions. See Stub.

34. **Dynamic Memory Allocation--**Allocating memory for a variable, structure, or class instance after a program has started running using either **Malloc()** (Standard C) or **new** (C++). See Static Memory Allocation.

35. **Edges--**The connections between vertices in a graph.

36. **Empty Set--**A set with no members.

37. **extern--**A specifier used to specify access to a global variable in one source file when the actual global variable is defined in another source file. A technique that is used more in old style procedural programming in C. Rarely if ever used in object oriented programming in C++.

38. **Flow Chart--**a means of diagramming an algorithm. This also does not have a rigid formal organization. But, there are some conventions we can look at:
    a. Small circles are used to indicate the entry and exit points to a block of code.
    b. Rectangles are used to indicate actions to be taken.
    c. Arrows are used to indicate the flow of control in the program.
    d. Diamonds are used to indicate decisions (like the if statement)

39. **Flow Control--**Controlling the sequence in which program commands are executed. See Sequential Execution and Transfer of Control.

40. **Full Binary Tree--**A tree in which all leaf nodes are on the same level and every non-leaf node has two children..

41. **Function Overloading--**The ability to have two or more functions in a class with the same name but different number and/or types of arguments. The OS determines from the arguments in a call to one of the functions which one to execute.

42. **Functional Decomposition--**See **Top Down Program Design**.

43. **Hash Table--**A large array of data structures in which data is stored by using a hash function to calculate in index into the array based on a data key.

44. **Hashing--**A technique for creating/calculating a unique index for a node in a hash table based on its' key.

45. **Heap--**A complete binary tree with values stored so that no child has a key value greater than its parent.

46. **In Degree--**In a directed graph, the number of edges a given vertex has coming in to it. See also **Degree of a Vertex, Out Degree, Predecessors of a Vertex** and **Successors of a Vertex**.

47. **Information Hiding--**The practice of hiding the details of a function, class, or data structure with the goal of controlling access to the details of the implementation of a class or structure. See Data Encapsulation.

48. **Inheritance--**A mechanism by which one class acquires the properties (member variables and member functions) of another class.

49. **Instantance--**See **Instantiation (noun)**.

50. **Instantiation--**(noun) The object that is created during the process of Instantiation (v). An **instance** of a class.

51. **Instantiation--**(verb) The process of dynamically allocating memory for an object. When a class object is dynamically allocated it has been **instantiated**.

52. **Integration Testing--**Testing of how functions work together. Done after Unit Testing.

53. **Iterators--**Functions that perform some sequential action on all data components in an object, example: setAll(someValue);

54. **Internal Node--**A node in a binary tree that is neither the root nor a leaf;

55. **Intersection--**A binary set operation that returns a set made up of only those items which are in both of the input sets.

56. **Leaf--**A node in a binary tree that has no children.

57. **Length of a Path--**In a graph, the number of edges from the starting vertex to the ending vertex in a path.

58. **Load Factor--**In a hash table, the radio of the number of filled entries (N) to the number of entries in the table (M); $\alpha = N / M$.

59. **Mapping--**In hashing, the process of converting the full range of key values into the full range of hash table indices.

60. **Member functions--**All of the functions declared within a class definition.

61. **Member variables--**All of the variables declared within a class or structure definition.

62. **Module--**A unit of organization of a software system. In most cases a module is a single source code file containing a number of functions designed to provide a set of related services. Classes in C++ can be referred to as modules. A conceptual idea not necessarily a programming unit.

63. **Modularity--**The organization of a program into logical units. An important characteristic of well designed programs. The use of classes in C++ helps to enforce this.

64. **Observers--**Functions that observe the state of one or more data values. This includes (1) Predicates--checks to see if a certain property is true or false, example: isEmpty(), (2) Accessor or Selector--returns a copy of a data object, example: getX(), (3) Summary--returns information about an object as a whole, example: getListLength().

65. **Object--**A software "bundle", usually thought of as being a **class** or **structure** consisting of a set of variables which define the states the object can exist in and a set of functions that define the behavior of that object. Software objects are often used to model the real-world objects that you find in everyday life.

66. **Object Oriented Design--**The process of identifying classes of objects and their interrelationships that are needed to solve a software problem.

67. **Object Oriented Programming--**The use of data abstraction, inheritance and dynamic binding (memory allocation) to construct programs that are collections

of interacting objects. The process of using objects as the building blocks of a program. See Procedural Programming, Top Down Design, Bottom Up Design.

68. **Operations**--All the functions that act on a domain of data. Example: for the int data type the operations include +, -, *, /, %, &, |, etc. See Abstract Data Type.

69. **Out Degree--**In a directed graph, the number of edges a given vertex has going out from it to other vertices. See also **Degree of a Vertex, In Degree, Predecessors of a Vertex** and **Successors of a Vertex**.

70. **Path**--In a graph, a list of vertices indicating those that must be visited in order to get from one vertex to another. If the two vertices are adjacent then this list has only two vertices in it.

71. **Perfect hash function**--A theoretical hash function that maps keys uniformly and randomly into a hash table without any collisions.

72. **Pointer--**A variable whose value is the address in memory of another variable. Pointers are typed. Thus you have **int** pointers that hold the address of **int** variables, **float** pointers that hold the address of **float** variables, etc. See the page on pointers for details on how to declare and use pointers.

73. **Polymorphism**--The ability of different sub-classes of a parent class to inherit functions from the parent class yet implement the functions in very different ways.

74. **Predecessors of a Vertex--**In a directed graph, the set of vertices from which the given vertex has edges coming in to it. See also **Degree of a Vertex, In Degree, Out Degree,** and **Successors of a Vertex**.

75. **Private--**Variables and functions in a structure or class which can only be accessed from within the object. Default state for member variables of classes.

76. **Procedural Programming--**Program design focusing on which functions are part of a program and how the functions interact to accomplish the goals of the program. See Object Oriented Programming, Top Down Design, Bottom Up Design.

77. **Protected--**Variables and functions in a structure or class which may be accessed only from other functions within the class or from functions in objects that are sub-classes of the class.

78. **Pseudocode--**An artificial and informal way of describing an algorithm. NOTE: There is no formal syntax or structure known as pseudocode. It is just a plain English outline of a program. The advantage is it allows you to concentrate on HOW to solve a problem without worrying about the programming language syntax at the same time.

79. **Public--**Variables and functions in a structure or class which are global and may be accessed from anywhere if there is access to the object. Default state for member variables of structures.

80. **Regression Testing--**Testing done after bugs have been fixed to insure the bug fixes have now introduced other problems.

81. **Requirements --**A statement of what is to be provided by a computer system or software product.

82. **Requirements Definition Document (RDD)--**The document, which is prepared from the Statement of Work and presented to the customer. It's primary purpose is to verify with the customer the exact scope of requirements for a piece of software. The RDD is created during the Requirements Specification phase of software engineering.

83. **Scope--**The section of code in which a variable can be accessed. For example, any variable defined inside of a function has scope only within the function. It cannot be accessed from outside the function.

84. **Sequential Execution--**The normal sequence of command execution. Programming commands are executed one after the other. See Flow Control and Transfer of Control.

85. **Set--**An unordered collection of distinct values (items or components) chosen from the possible values of a domain. The domain may also be called the Component (base) type.

86. **Simple Cycle--**In an undirected graph, a cycle that consists of three or more distinct vertices, e.g. A--B--C--A, and no vertex is visited more than once except for the starting/ending vertex which is the same.

87. **Software Design Document (SDD) or Software Development Plan (SDP) --** The document which describes in detail each module of a software system, each function within the module, the function inputs and outputs, and the algorithms each function will perform. The SDD or SDP contains a description

of the interface between each module and the interface with the user. The SDD or SDP is created during the Design phase of software engineering.

88. **Software Engineering --**A disciplined approach to the design, production, and maintenance of computer programs that are developed on time and within cost estimates, using tools to help manage the size and complexity of the resulting software products.

89. **Software Requirements Specification (SRS)--**The document which presents a thorough analysis of the requirements of a piece of software. The SRS is based on the SOW and RDD and is written in terms the software designers and programmers can use. The SRS is created during the Analysis phase of software engineering.

90. **Software Specification--**A detailed description of the functions, inputs, processing, outputs, and special requirements of a software product. This provides the information needed to design and implement the program.

91. **Software Test Plan (STP)--**The document which is a detailed, organized plan for testing a piece of software. The STP includes a description of all tests to be performed, the procedure for performing the test, the inputs for the test, and the expected outputs. The STP is created during the Testing phase of software engineering, but the normal approach is to begin working on a rough draft of the STP during the design phase so that every part of the system included in the design has a test or tests included in the STP.

92. **Statement Of Work--**The document, usually received from a customer, that states specifically the customer's requirements for a software product. The SOW is created during the Requirements Specification phase of software engineering.

93. **static**--An access specifier which has a several uses:

   1. A variable defined as static inside of a function (ex: **static int x = 0;**) maintains the same value it had the last time the function was called. In the example given the first time the function containing **x** is called the variable is created and initialized to the value of zero. If while the function is running **x** is incremented with **x++;** then the next time the function is called **x** will have the initial value of one.
   2. A function in a class definition (.h) file declared as static (ex: **static someFunction()** is declared in **MyClass.h**) can be called without having to create an instance of the class. For example: **MyClass::someFunction()** would call the static function as defined above. However, if there are multiple instances of the class all will share the same instance of the static function and static functions cannot access non-static variables or functions in a class.
   3. A global variable declared as static in a source code file cannot be declared as **extern** in another source file and accessed.

94. **Static Memory Allocation**--Memory that is allocated at program start up or when a function is called. Variable declarations in a program result in static memory allocation. See Dynamic Memory Allocation.

95. **Step-wise Refinement**--The process in software design of starting off at a high level of abstraction and working down through an interative process to add more and more detail to the design.

96. **Structured Programming**--A concentrated effort throughout the programming process to produce well organized code. Structured programming usually follows 5 steps in program development:

   . **Requirements Specification**--A detailed list of what the program must be able to do. In large development projects this is usually referred to as the **Statement of Work**.

   a. **Analysis**--How you plan to solve the problem. This includes: (1) identifying the problemıs imputs, desired outputs, and other constraints, (2) specifying the form of the data input (units, format, order, and arrangement), and (3) specifying the form of data output.

   b. **Design--**Develope the **Algorithm**, a list of steps to be taken to solve the problem.
   c. **Implementation--**Code the algorithm in C.
   d. **Verification and Testing--**This is a two step process. It involves (1) determining if the program does, in fact, produce results that satisfy the original requirements (Verification), and (2) the results are accurate (Validation).

97. **Stub--**A special function written to use as a testing function. A stub returns known outputs to a calling function to determine how the calling function handles the returned values. Used to test higher level functions. See Driver.

98. **Subset--**A set X is a subset of Y if each item in X is also in Y.

99. **Successors of a Vertex--**In a directed graph, the set of vertices to which the given vertex has edges going out from it. See also **Degree of a Vertex, In Degree, Out Degree,** and **Predecessors of a Vertex**.

100. **Top Down Program Design, with Stepwise Refinement--**an approach to structured programming which involves working from the larger problem down to the details (Top Down Design) by expanding the steps in an algorithm to add more detail at each step (Stepwise refinement). Also called **functional decomposition** and **procedural programming.** The focus is on the processes. See Bottom Up Design.

101. **Transfer of Control--**Jumping to a program statement that is not the next one in the sequence. Calling a function is an example of Transfer of Control. See Flow Control and Sequential Execution.

102. **Transformers--**Functions that change the data in some way. Examples: **X=3**. The "equals" function set the value to store in the variable X. This may include setting values, inserting or deleting an item from an object, or combining two objects.

103. **Undirected Graph--**A graph in which the paths (edges) between vertices go in both directions. See also **Directed Graph**.

104. **Union--**A binary set operation that returns a set made up of all items that are in either of the input sets.

105. **Unit Testing--**Testing of individual functions.

106. **Universal Set--**The set containing all the values of the component type. For example, in the set of integers the Universal set consists of all whole numbers from negative infinity to positive infinity.

107. **Validation--**The process of determining the degree to which a piece of software produces results that satisfy the original requirements. Does it do what it is supposed to do, i.e. did you build the right software.

108. **Verification--**The process of determining the degree to which a piece of software produces results that are accurate. Does it do what it is supposed to do **correctly**, i.e. did you build the software right.

109. **Vertices--**The nodes in a graph.

110. **Weighted Graph--**A graph in which each edge has a value associated with it. For example: a graph of routes between cities on a map may have a mileage associated with the route (edge) connecting each city (node) in the graph.

111. **White-box Testing--**Testing a program or function based on knowing the internal working of the program or function. Based on making sure the testing covers all possible paths through the code.