1. What are the **programming** languages used for?

2. Why **learn different** programming languages?

3. Are the modern programming languages converging (having common features) and/or diverging (becoming more specialized)?

4. What are 3 **common characteristics** of modern languages (Hint: 3 S’s)?

* Structure, Syntax, Semantics

5. Why are there so many programming languages?

6. What are some of the common language **constructs** (name three)

* Loops, binary logic/Booleans, if/else

7. (**Fully**) Object-Oriented languages have **3 or 4 features** that set them apart from the procedural languages; what are they (name **at least** 3)?

* Polymorphism, inheritance, encapsulation, abstraction

8. What is the **definition** of a program (be concise)?

* A set of instructions to be carried out by the machine

9. **Define** *imperative language*.

* Language is compiled -> Exact steps is what is handed out
* Adv- Run quickly because compiled and closed the loopholes
* Disadv- More rigid, doesn’t give you any wiggle room

10. **Define** Non-imperative/functional/logical/declarative.

* Give the program the tools to handle the situations rather than the exact steps.

11. What is *role* of *standardization* with respect to various programming languages?

* Easy to switch languages, learn new ones, common themes.

12. The topic of “Algorithms”, is typically a precursor for the topic of “Programming languages”. What are the two top-most concepts with which “**Algorithms**” is concerned?

* Temporal complexity – amount of time used
* Spatial complexity – amount of memory used
* Often opposed to each other. Use more memory to gain speed or use less memory but lose speed

13. What is a Context-Free Grammar (**CFG**) and its utility?

* Pattern detection

14. What is the difference between *compiler* and *interpreter*?

* Compiler – compiling command which creates and executable (runs really quickly)
* Interpreter – run the code, code is interpreted as it is run (runs slower)

15. **Name** 3 different **types** of prevalent applications?

* Data analysis, web development, word processing, robotics, etc

16. What is meant by **complexity** of an application?

* Any big road-blocks or major issues/hard things to work with

17. What is meant by **company culture**?

* How your company operates and social structure

18. What is meant by **time-to-market**?

* How long it takes to get product into hands of customer

19. What is meant by **maintainability**?

* How hard is upkeep?

20. What is the inter-relationship (dynamism) between **scalability** and **performance**?

* Scalability – How versatile program is to change and growth
* Performance – How quickly the program runs
* Go against each other as a more diverse problem set requires more solutions and will be slower

21. What is meant by **application-level security**?

* Only get what is applicable to your level, higher up gets higher level

22. What is the role of **testability** while in development/deployment stages?

* Have to be able to see if your product will work as imagined

23. What was (is) the justification for “Data-Oriented Languages”?

* Intuitively and cleanly interact with a database -> Needed to analyze data

24. What do RISC and CISC stand for? and their differences/commonalities and utilities?

* RISC – Reduced instruction set code
  + A few building block instructions that can do an incredibly large amount of things
* CISC – Complex instruction set code
  + Instructions for every single problem that may pop up

25. What are the two simple models of Computability?

26. What is the **formal** definition of *syntax*?

* Set of rules defining symbols/structure

27. What is Extended Backus-Naur Form (EBNF) and its purpose/utility?

* Syntax

28. What is the implication of “formalization of semantics” (*Hint:* the overarching of this concept is used in “Deign-By-Contract” or “Black-Box” design)?

29. Briefly, describe the following terms:

a. Value: An entity that can be manipulated or changed

b. Literal: Representable value

c. Representation: How data is stored, processed

d. Constant: Unchanging value

e. Object: Variable, structure type, function

30. What is the contrasting difference between an “assignment” (using a computer) and an “equation” (in Mathematics)?

* Giving a value to something vs a balance something

31. What is *type checking*?

* Ensuring the variable is the correct type

32. What are the possible **approaches** to *type checking*?

* Static = Compiled
* Dynamic = During run

33. What is the difference between “Sub-programs” and a “Modules” (Hint: think of the term, “Encapsulation”)?

* Sub-Program: Separate parts creating a whole
* Modules: One part of the whole

34. Name (AT LEAST) three components of “Programming Environments”.

* IDE, Compiler, Flags

35. What is a “configuration tool(s)”? use an example.

* Helps with deployment
* Include Paths, Compiler Flags

36. Security in Java – name three?

* Memory management
* Strong type-checking
* Can’t pass by reference

37. In what mechanisms (thinking mathematically) are integers (positive and/or negative) produced/represented?

* 2’s compliment = Flip all bits, then add 1

38. In what/which order do the numerical computation are performed? Is it possible to change that order? If so, **how**?

* PEMDAS
* Change using parentheses

39. What is *overflow*?

* When you store more data than you can (store 32 +1 bits in a 32 bit memory)

40. What does “integer division” imply and whether it is useful?

* Divides but ignores remainders/creating double

41. What is “mod”/modulus operator and whether it is useful?

* Finds remainder, def useful

42. Could computations be performed with characters? If so, what is the underlying mechanism that makes it possible?

* Yes, ACII values

43. Boolean types – why are they important?

* Easy to deal with, handle

44. What is *an expression*?

* Values and conditions evaluated giving us T/F value

45. What’s the outstanding characteristic of an **assignment** statement?

* Right side is assigned to left

46. What ARE the outstanding characteristics/features of an array (in C/C++).

* All need same data type, static number of elements when declared, contigeous memory

47. What does an array of characters have in common with a string (in C/C++)?

* Array has a set size, string can be any size. Otherwise, they are the same thing

48. What’s the characteristic of a string (in C/C++)?

* String is terminated with \n
* Array of characters

49. What’s the key-syntax of a 2-D array or higher dimension (in C/C++) when used in function *prototyping*?

* Have to set bounds of 2D array

50. What’s “Bitwise computation”?

* And, Or, Xor, Not

51. What’s the role of “break” statement in “switch “construct? What happens if the “break” statement**s** are eliminated?

* Exit switch statement or all cases happen

52. Sometimes, “switch” constructs could be implemented with “if-else” (in C/C++/Java). Are there any other times that “switch” and “if-else” constructs CAN NOT be transparently interchanged? When/how?

* Want to do all scenarios, not an if-else scenario

53. What’s the **term** called in which the logically compounded-AND “if” expression stops checking the rest of the conditions upon the failure of the first condition?

54. What is the (**potential**) deterrence when using loops? Is it possible that a loop is never executed?

* Can go forever, not enough, or not at all -> Condition is already met, so loop doesn’t run at all.

55. What is the *nature* of parameters (in functions and the calls)?

* Place holders and defining

56. What properties are associated with variables that makes them viable or inviable?

* Const, type
* Wrong type, can’t change but need to be

57. What is Stack (and its assigned behavioral acronym)? What data structure is capable of **simulating** a stack?

* First in last out
* Array of items is a similar data structure

58. How does overall “stack size” determined - whether when it is used at the time recursion or otherwise)?

* Has a max stack size set (assigned chunk of memory)

59. What is Queue (and its assigned behavioral acronym)? What data structure is capable of **simulating** a Queue?

* First in first out
* Again, arrays could be applied here

60. What are the two salient-features of recursion – that makes it work?

* Base case
* Recursive case

61. What are the pointer’s characteristics (in C/C++) in contrast to a variable – both in compile time vs. runtime?

* Base address + offset

62. How does the notion of pointer in Java differ from that of C/C++?

* Can’t do arithmetic

63. Most modern languages use “typed pointers” (e.g., int\* i\_ptr). If a pointer is addressing a specific address, then what’s the use/significance in specifying the type?

* So it can set aside the correct amount of memory for that value to go

64. What is “pointer to sub-programs”?

* Beginning address of sub-program

65. What does *memory allocation* involve?

* Set aside memory for vars you’re planning on using

66. What are meant by “allocation” and “deallocation”?

* Allocation: Take up memory to be used
* Deallocation: Free up memory to be used

67. What is a *dangling* pointer?

* Pointer pointing at memory no longer allocated

68. What is the main deterrence in heap allocation?

* Could run out of heap (use too much memory)

69. When is heap allocation (essentially) risk-free?

* Very little memory, predictable memory, maybe never?

70. What is meant by *Garbage collection*?

* Cleans up memory leaks, dangling pointers after run

71. Why computation with integers could cause *overflow*?

* If really big, overflows and creates a very large negative number wrapping around

72. What are the two binary approaches in representing decimal values in computers?

* 1 or 2’s compliment

73. What are fixed-point (accuracy – absolute error, is fixed) numbers?

* Fixed number of decimal points

74. What are floating-point numbers?

* Mantissa

75. What is “Language Support” for *real* numbers?

76. What is “Portability” of floating-point?

* Diff. representations of floating points in diff. languages
* Does it hold if switched between languages

77. Floating-point operation major flaws:

* Loss of significance/accuracy

78. What is polymorphism?

* One function capable of many different things (frog survives anywhere)

79. What is Exception Handling?

* Throwing and catching errors

80. Does C have “**Exception Handling**” (i.e., similar to C++/Java)?

* No, asserts only

81. What is the (technical) difference between *concurrency* and *parallelism*?

* Concurrency is multiple jobs on one cpu
* Parallelism is one job with multiple cpu’s

82. What is “shared memory”?

* Memory multiple processes can access

83. What is *mutual exclusion*?

* XOR

84. What are *monitors* and *protected objects* - Not a physical H/W display?

* Monitors – Watch dogs
* Protected Objects – Not being changed/altered

85. What is *message passing*?

* Passing message between 2 diff. processes

86. What is *program decomposition*?

* Break up program into multiple sections

87. What is *separate compilations*?

* Completely separate jobs

88. Modularity vs. Reusability? Are they the same? If **yes**, why? If **no**, why?

* Modularity – Sub tasks does one job
* Reusability – Does one or many jobs

89. What is “**name spaces**” (in C++) and its utility?

* Abstract environment container

90. What are *constructors* in *destructors* (in C++)

* Constructor: Allocates memory for class or var
* Destructor: Frees up the memory used from the class or var

91. What is a *default constructor*?

* If not customized, default instance is made

92. Does *destructor* have a *detrimental* feature? why/why not?

* If you free up memory, chance of losing memory by accident (Memory Leak)

93. What is *memory leak*? Explain succinctly.

* Losing the pointer to a memory location (no longer accessible)

94. What is *function-name* overloading?

* Polymorphism, same name but requires diff. func. calls

95. The article, “Large scale study of programming Languages” outlines “Language Classes” and “Categories”. What are the Classes and the Categories?

* Programming paradigm, compilation class, type class, memory class
* Paradigms: Procedural and scripted
* Compilation: Static and dynamic
* Type: Strong and weak
* Memory: Managed and unmanaged

96. In the Article above, bugs are categorized along six parameters. Name three.

* Runtime errors, security, memory, concurrency