1. Syntax: How programming look, their form and structure
2. Semantics: what programs do, their behavior and meaning
3. Imperative language: Local variable / Assignment, Iteration, Order of execution is critical
4. OOP: A class can be a type, data and operations on that data, bundled together
5. Functional language: No variable just parameter, recursive 하게 부름 / Basic unit of design is function, Recursion is fundamental
6. Logical Language: Program expressed as rules in formal logic
7. Context Free Grammar: Tokens, Non-terminal, Start symbol, Production rule
8. Token: the smallest units of syntax
9. Non-terminal symbols: larger pieces of syntax
10. Start symbol: particular non-terminal that forms the root of any parse tree for the grammar
11. Productions: tree-building rules
12. Compiler: before) Preparation Time / Execution) 프로그램 전체 번역 whole result of code→ 프로그램의 일부를 수정하는 경우에도 전체 프로그램을 다시 컴파일 해야 함 /instructions of the programming language are complex
13. Interpreter: Before) nothing / Execution) 실행되는 줄 단위 번역 results by one line → 실행할 때마다 매번 기계어로 바꾸는 과정을 다시 수행해야 하므로 항상 인터프리터가 필요함/ Start right away
14. Dynamic Compilation: Compiling that takes place after running
15. Just-in-Time Compilation: The program is stored in memory as byte code, but the code segment currently running is preparatively compiled to physical machine code in order to run faster
16. Virtual Machine: Simulated in software-interpreted, no Hard ward - 장점: Cross-platform execution (Simulating physical machines is harder) / Heightened security / Platform-independent
17. Interpretive Compiler: use an intermediate language - more high-level than machine code (compiler), more low-level than source language (implement as an interpreter)
18. Scope rules: regulate visibility of identifiers
19. Type rules: regulate the expected types of arguments and returned value
20. Polymorphism
21. Ad hoc polymorphism: if it as at least two but only finitely many possible types
22. Overloading: that has at least two definitions, all of different types (int p()~~, double p()~~)
23. Parameter Coercion: 변수 타입 강제 변환 implicit type conversion, (double) 2
24. Universal polymorphism: it has infinitely many possible types 여러개
25. Subtype Polymorphism: 파라미터 타입 여러개 if one or more of its parameter types have subtypes (Animal cat1 = new Cat("Kitty");)
26. Parametric Polymorphism: 여러개 변수 타입 if it has a type that contains one or more type variables, Stack<Object>
27. Static linking
28. Dynamic linking
29. Delayed linking
30. Load-time dynamic linking: loader finds .dll files (which may already be in memory) and links the program to functions it needs, just before running
31. Run-time dynamic linking: Running program makes explicit system call to find .dll files and load specific functions
32. Activation record: Activation of the function: the lifetime of one execution of a function from call to corresponding return - Activation record(Return address, Link to caller's activation record) → Activation-specific variable: each activation has its own binding of a variable to a memory location
33. Static Allocation: Allocate one activation record for every function Simple and fast vs. bad at recursion, multithreading
34. Dynamic Allocation: Activation record allocated when function is called → Stack of activation records: Stack frames pushed on call, popped on return
35. Nesting Links in activation record: inner function to be able to find the address of the most recent activation for the outer function
36. Displays: nesting links collected in a single static array
37. Lambda lifting: Problem references replaced to new, hidden parameters
38. Current heap link
39. Delayed Coalese
40. Garbage Collection
41. Formal parameter
42. Actual parameter
43. Positional parameters: determined by positions (순서 바뀌 X) / Named (or Keyword) parameters: parameter names (순서 바뀌어도 됨)
44. Mixed Positional and named parameter
45. Optional parameters: with default values, 생략 가능
46. By value 값을 전달(copy-in, 초기값 O) int z = plus(x,y);
47. By result (copy-out, 별로 안씀, 초기값 X) int z; plus(x,y,z);
48. By value-result (copy-in/copy-out) int z=1; plus(x,y,z);

49. By reference 주소값 전달 int z=1; plus(x,y,&z); → 함수 실행될 때 동시에 변하기
50. By macro expansion(매크로, Capture) #define MIN(X,Y) (~~)
51. By name (Macro Expansion without capture) 환경에 따라 이름이 딱 정해지게
52. By need (꽤, on functional language) – Lazy evaluation
53. Chomsky Hierarchy: Unrestricted grammar > Context Sensitive Grammar > Context Free Grammar > Regular Grammar
54. Compiler: translates to assembly language
55. Assembler: converts each assembly language instruction into the machine's binary format
56. Linker: collects and combines all the different parts of object file to executable file
57. Loader: Loader loads it into memory and replaces names with addresses in machine language
58. Optimization: optimized to make it faster, smaller
59. Binding: associating some property(Text, Data, Heap, Stack) with an identifier from the program
60. Debugger: examine a running program on the fly / Modify currently running program / examine a snapshot
61. Semantic analyzer: checking scope rules and type rules
62. Profiling: The classical sequence runs twice - First: collects statistics (parts most frequently executed) - Second: uses information to help generate better code
63. activation of function
64. activation-specific variable
65. activation record
66. Lazy evaluation