SPCC VIVA QUESTIONS

I. Introduction to Software System

1. System Program:
   - It is a type of computer program that is designed to run a computer's hardware and application programs.
   - Example: compiler, assembler, debugger, drivers.

2. Application Program:
   - It is a computing program designed to carry out a specific task other than one relating to the operation of the computer itself, it is to be used by end-users.
   - Example: media player, word processor

3. Difference between system software and application software:

System Program	Application Program
Computer software which manages and controls computer hardware.	Programs that enable the end user to perform tasks such as word processing.
Developed in Low Level Language like assembly language or machine language.	Developed in High Level Language like Python,C, C++, Java.
Functions are to manage the resources, automate its operations and facilitate program development	Designed to perform specific data processing or computational tasks for the user.
Programming is done using assembly language which interacts with hardware.	Programming is used to build application software which includes software.
System software is used to execute the application software.	Application software is software that has been used by the end user.
Example: compiler, debugger, drivers.	Example: Word processor, web browser.

4. Assemblers:
   - It is a program for converting instructions written in low-level assembly code into relocatable machine code and generating information for the loader.
   - It is basically a translator of language.
   - There are 3 types of assemblers:
     a. Single pass: It requires a single pass to assemble the program into machine code.
     b. Two pass: It requires two passes to assemble the program into machine code.
     c. Multipass: It requires single or multiple passes to assemble the program into machine code.

5. Loaders:
   - Loader is a program which accepts the object program and prepares these programs by loading them into memory.
   - It performs 4 functions:
     a. Allocation: used to allocate space in memory for the programs.
     b. Linking: links two or more objects code and provides the information needed to allow reference between them.
     c. Relocation: modifies the object program by changing certain instructions so that it can be loaded at a different address to avoid address conflict.
     d. Loading: bring object code into memory.

6. Linkers:
    - It is a program in a system which helps to link object modules of a program into a single object file.
    - It performs the process of linking.
    - Relocation and linking of all object modules is done by linker.

7. Compilers:
    - A compiler is a translator that converts the high-level language into the machine language.
    - Compiler is used to show errors to the programmer.
    - The main purpose of a compiler is to change the code written in one language without changing the meaning of the program.
    - When you execute a program which is written in High Level programming language then it executes into two parts.
    - In the first part, the source program compiled and translated into the object program (low level language).
    - In the second part, the object program translated into the target program through the assembler.

8. Applications of compiler:
    - Machine code generation
    - Query interpretation
    - Formal converter
    - Text formatting

9. Phases of Compiler:
    - Lexical Analysis
    - Syntax Analysis
    - Semantic Analysis
    - Intermediate Code Generation
    - Code Optimization
    - Target Code Generation

10. Interpreters:
    - It translates high level instruction into an intermediate form and then executes them.
    - It is a computer program that is used to directly execute program instructions written using high-level programming languages.

11. Difference between compilers and interpreters:

Interpreter	Compiler
It translates high level instruction into an intermediate form line by line.	It compiles the whole program at one time directly into machine language.
Advantage of line by line interpretation is that it can interrupt in between and can change the code again and start again.	Translation of the program is done at one time resulting in faster execution of the program.
Disadvantage is that every line has to be translated every time it is executed.	Code can't be changed without going back to the original source code.
Good for simple applications.	Good for complex applications.

Slow in comparison to compilers	Compiled programs run faster than interpreted programs.
Less compact.	More compact.
Example: Javascript	Example, C, C++

II. Assemblers.

12. Data structure of pass 2 assembler:
    - 3 major data structures are involved in 2 pass assembler
        a. OPTAB - Operator code Table - This is used to look up mnemonic operation codes and translate them into equivalent machine language .
        b. SYMTAB - Symbol Table - This is used to store addresses assigned to labels
        c. LOCCTR - Location Counter - This is used for assignment of addresses.

13. Forward reference:
    - When a symbol is referred before it is defined, it is called a forward reference.
    - The function of the assembler is to replace each symbol by its machine address and if we refer to that symbol before it is defined its address is not known by the assembler such a problem is called forward reference problem.
    - Forward reference is solved by making different passes (i.e., One Pass, Two Pass, and Multi-Pass) over the assembly code.

14. Use of Two pass assembler:
    - It is used to avoid the problem of forward reference.

III. Macros and Macro Processor

15. Macro:
    - Macro is a block which contains a group of instructions.
    - It allows the programmer to write a shorthand version of a program.
    - Using a macro, a programmer can define a single instruction to represent a block of code.

16. Macro Processor:
    - To process macro instruction, most assemblers use pre-processors known as macro processors.
    - The macro processor replaces each macro instruction with the corresponding group of source language statements.
    - The design of a macro processor generally is machine independent.

17. Features of Macro:
    - Associating macro parameters with their arguments
    - Delimiting macro parameters
    - Directives related to arguments
    - Automatic label generation
    - Machine independent features

18. Syntax of Macro:
    - %macro macro_name number_of_params
    - <macro body>
    - %endmacro

19. Data Structures of Macro:

a. DEFTAB or MDT (Definition table):
    a. It stores the macro definition including macro prototype and macro body.
    b. References to the macro instruction parameters are converted to positional notation for efficiency.

b. NAMTAB:
    a. It stores macro names, which serves as an index to DEFTAB and contain pointers to the beginning and ending of the definition.

c. ARGTAB:
    a. Used during the expansion of macro invocations
    b. When a macro invocation statement is encountered, the arguments are stored in this table according to their position in the argument line.

20. Parameterized Macro:
    - It is a type of macro that is able to insert given objects into its expansion.
    - It gives macro some of the power of a function.

IV. Loaders and Linkers

21. Types of Loader:

a. Absolute Loader:
    - The absolute loader is a kind of loader in which relocated object files are created, loader accepts these files and places them at a specified location in the memory.
    - This type of loader is called absolute loader because no relocating information is needed, rather it is obtained from the programmer or assembler.

b. Relocating Loader:
    - A relocating loader is capable of loading a program to begin anywhere in memory.
    - The relocating loader adjusts, or relocates, each address in the program.

c. Direct Linking Loader
d. Dynamic Loader
e. Assemble – and – go or Compile – and – go loader
f. BootStrap Loader

22. Difference between Linkers and Loaders:

LINKER	LOADER
The main function of Linker is to generate executable files.	Whereas the main objective of Loader is to executable files to main memory.
The linker takes input of object code generated by the compiler/assembler.	And the loader takes input of executable files generated by linker.
Linking can be defined as the process of combining various pieces of codes and source code to obtain executable code.	Loading can be defined as the process of loading executable codes to main memory for further execution.
Linker is of 2 types: Linkage Editor and Dynamic Linker.	Loader is of 3 types: Absolute loading, Relocatable loading and Dynamic run-time loading.

Another use of linker is to combine all object modules.	It helps in allocating the address to executable codes/files.
Linker is also responsible for arranging objects in the program's address space.	Loader is also responsible for adjusting references which are used within the program.

V. Compilers:

23. Analysis phase and Synthesis phase:
    - Analysis phase creates an intermediate representation from the given source code.
    - Synthesis phase creates an equivalent target program from the intermediate representation.

24. Phases of Compilers:

- Analysis Phase:
    a. Lexical Analysis:
        - Lexical analyzer phase is the first phase of the compilation process.
        - It takes source code as input.
        - It reads the source program one character at a time and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens.
    b. Syntax Analysis:
        - Syntax analysis is the second phase of the compilation process.
        - It takes tokens as input and generates a parse tree as output.
        - In the syntax analysis phase, the parser checks that the expression made by the tokens is syntactically correct or not.
    c. Semantic Analysis:
        - Semantic analysis is the third phase of the compilation process.
        - It checks whether the parse tree follows the rules of language.
        - Semantic analyzer keeps track of identifiers, their types and expressions.
        - The output of the semantic analysis phase is the annotated tree syntax.
- Synthesis Phase:
    d. Intermediate Code Generation:
        - The compiler generates the source code into the intermediate code.
        - Intermediate code is generated between the high-level language and the machine language.
        - The intermediate code should be generated in such a way that you can easily translate it into the target machine code.
    e. Code Optimization:
        - Code optimization is an optional phase.
        - It is used to improve the intermediate code so that the output of the program could run faster and take less space.
        - It removes the unnecessary lines of the code and arranges the sequence of statements in order to speed up the program execution.
    f. Code Generation:
        - Code generation is the final stage of the compilation process.
        - It takes the optimized intermediate code as input and maps it to the target machine language.
        - Code generator translates the intermediate code into the machine code of the specified computer.

25. Basic Definitions:
    a. Lexemes:
        - It is a basic lexical unit of language consisting of one word or several words that corresponds to a set of words that are different forms of the same word.
        - Lexemes are the smallest units of a program.
        - It is a sequence of characters in the source program for which a token is produced.
        - Ex: Lexeme: (Token: LPAREN
    b. Tokens:
        - They are a sequence of characters that can be treated as a unit in the grammar of the programming language.
        - Example:
        - Type tokens: (“id”, {a-z, A-Z, 0-9}, “num” {0-9}, “real”,..)
        - Punctuation tokens (“if”,”void”,”return”, “begin”, “call”, “do”, “end”, “while”)
        - Alphabetic tokens: (“keywords”)
        - Symbol tokens: (‘+’, ‘-’, ‘ * ‘, ‘/’, ‘=’, ’<’, ’>’)
        - Non tokens: comments, preprocessor directive, macros, blank...
    c. Patterns:
        - It is a template or model which can be used to make or generate things for parts of a thing especially if things that are generated have enough in common.
        - It is a set of strings in input for which the same token is produced as output.
        - These sets of strings are described by a rule called pattern.

26. Top down Approach in Parsing:
    - It is a parsing technique which starts from the top of the parse tree, moves downwards, evaluates rules of grammar.

27. Bottom Up Approach in Parsing:
    - It is a parsing technique which starts from the lowest level of the parse tree, moves upwards and evaluates rules of grammar.

28. Lexical Errors:
    - It is a sequence of characters that does not match the pattern of any token and it is found during the execution of the program.
    - Ex: To remove the character that should be present, to replace a character with an incorrect character, spelling errors.

29. Syntax Errors:
    - is found during the execution of the program.
    - It can be an error in structure, missing operator, unbalanced parenthesis.
    - Ex: using ‘=’ instead of ‘==’ while testing equality.

30. Semantic Errors:
    - Semantic errors are problems with a program that runs without producing error messages but doesn't do the right thing.
    - Example: Use of a non-initialized variable, undeclared variable, not matching of actual argument with formal argument.

31. Code Optimization:
    - Code Optimization is an approach to enhance the performance of the code.

- The process of code optimization involves:
    a. Eliminating the unwanted code lines
    b. Rearranging the statements of the code

32. Code Optimization Techniques:

a. Compile Time Evaluation:
    - Two techniques that falls under compile time evaluation are
        a. Constant Folding
        b. Constant Propagation

b. Common subexpression elimination:
    - The expression that has been already computed before and appears again in the code for computation is called Common Sub-Expression.

c. Dead Code Elimination:
    - In this technique, As the name suggests, it involves eliminating the dead code.
    - The statements of the code which either never executes or are unreachable or their output is never used are eliminated.

d. Code Movement:
    - In this technique, As the name suggests, it involves movement of the code.
    - The code present inside the loop is moved out if it does not matter whether it is present inside or outside.
    - Such a code unnecessarily gets executed again and again with each iteration of the loop.
    - This leads to the wastage of time at run time.

e. Strength Reduction:
    - In this technique, As the name suggests, it involves reducing the strength of expressions.
    - This technique replaces the expensive and costly operators with the simple and cheaper ones.

33. Machine-independent Optimization:
    - In this optimization, the compiler takes in the intermediate code and transforms a part of the code that does not involve any CPU registers and/or absolute memory locations.

34. Machine-dependent Optimization:
    - Machine-dependent optimization is done after the target code has been generated and when the code is transformed according to the target machine architecture.
    - It involves CPU registers and may have absolute memory references rather than relative references.

35. Issues in code generation:
    - Input to the code generator
    - Target program
    - Memory management
    - Instruction selection
    - Register allocation
    - Evaluation order

36. DAG (Directed Acyclic Graph):
    - DAG is a tool that depicts the structure of basic blocks, helps to see the flow of values flowing among the basic blocks, and offers optimization too.
    - DAG provides easy transformation on basic blocks.