

SPCC Viva Questions

Chapter 01

1. What is system program?

The system programs are used to program the operating system software. While application programs provide software that is used directly by the user, system programs provide software that are used by other systems such as SaaS applications, computational science applications etc.

2. Define assembler, compiler, Interpreter, Linker Loader, Editor?

Compiler –The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.

Assembler –The Assembler is used to translate the program written in Assembly language into machine code. The source program is an input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.

Interpreter –The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter

Linker – A linker is special program that combines the object files, generated by compiler/assembler, and other pieces of codes to originate an executable file have. exe extension. In the object file, linker searches and append all libraries needed for execution of file. It regulates memory space that code from each module will hold. It also merges two or more separate object programs and establishes link among them.

Loader – The loader is special program that takes input of object code from linker, loads it to main memory, and prepares this code for execution by computer. Loader allocates memory space to program. Even it settles down symbolic reference between objects. It in charge of loading programs and libraries in operating system. The embedded computer systems don't have loaders. In them, code is executed through ROM.

Editor – Editors or text editors are software programs that enable the user to create and edit text files. In the field of programming, the term editor usually refers to source code editors that include many special features for writing and editing code.

3. Differentiate Application program and system program.

S.No	System Software	Application Software
1.	System Software maintain the system resources and give the path for application software to run.	Application software is built for specific tasks.
2.	Low level languages are used to write the system software.	While high level languages are used to write the application software.
3.	Its a general purpose software.	While its a specific purpose software.
4.	Without system software, system can't run.	While without application software system always runs.
5.	System software runs when system is turned on and stop when system is turned off.	While application software runs as per the user's request.
6.	Example of system software are operating system, etc.	Example of application software are Photoshop, VLC player etc.
7.	System Software programming is complex than application software.	Application software programming is simpler as comparison to system software.

4. Differentiate compiler and interpreter.

S.No.	Compiler	Interpreter
1.	Compiler scans the whole program in one go.	Translates program one statement at a time.
2.	As it scans the code in one go, the errors (if any) are shown at the end together.	Considering it scans code one line at a time, errors are shown line by line.
3.	Main advantage of compilers is it's execution time.	Due to interpreters being slow in executing the object code, it is preferred less.
4.	It converts the source code into object code.	It does not convert source code into object code instead it scans it line by line
5.	It does not require source code for later execution.	It requires source code for later execution.
Eg.	C, C++, C# etc.	Python, Ruby, Perl, SNOBOL, MATLAB, etc.

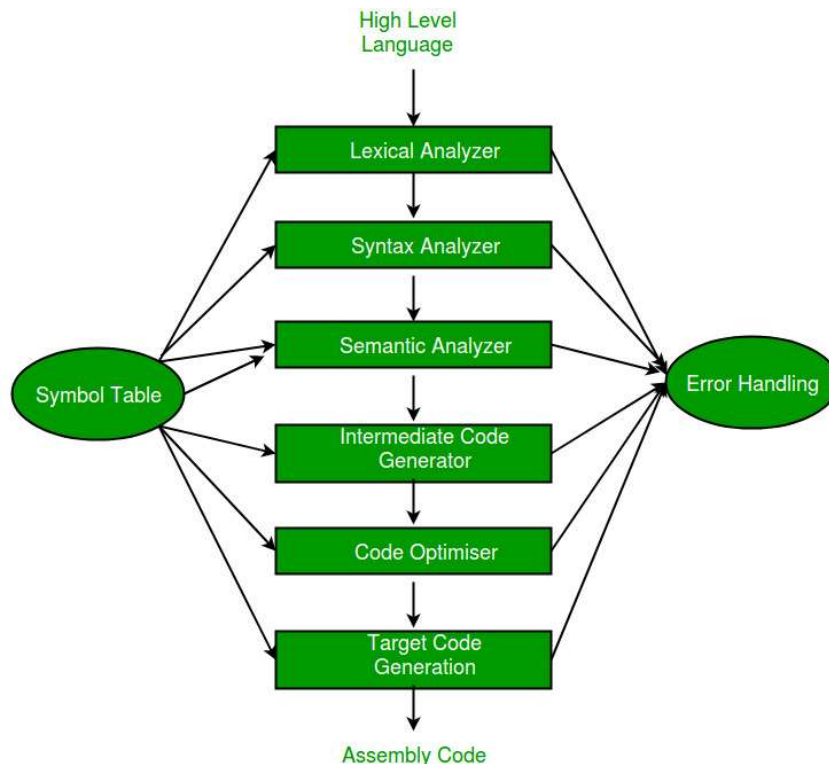
5. Goals of system software

Convenience: the primary goals of OS is to make computer system easier for user i.e. OS makes interaction b/w user and hardware.

Efficient: the secondary goal of OS is to allocate the system resources to various applications program as efficient as possible.

Chapter 05

6. Phases of compiler.



7. What is cross compiler

A Cross compiler is a compiler that generates executable code for a platform other than one on which the compiler is running. For example, a compiler that running on Linux/x86 box is building a program which will run on a separate Arduino/ARM.

8. What is source to source compiler

A source-to-source compiler is a type of translator that takes the source code of a program written in a programming language as its input and produces an equivalent source code in the same or a different programming language. A source-to-source translator converts between programming languages that operate at approximately the same level of abstraction, while a traditional compiler translates from a higher-level programming language to a lower-level programming language.

9. Analysis phase and synthesis phase

Analysis Phase: Analysis phase reads the source program and splits it into multiple tokens and constructs the intermediate representation of the source program. And also checks and indicates the syntax and semantic errors of a source program. It collects information about the source program and prepares the symbol table. Symbol table will be used all over the compilation process. This is also called as the front end of a compiler.

Synthesis Phase: It will get the analysis phase input (intermediate representation and symbol table) and produces the targeted machine level code.

This is also called as the back end of a compiler.

10. What is lexical analysis

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

11. What is syntax analysis

Syntax Analysis or Parsing is the second phase, i.e., after lexical analysis. It checks the syntactical structure of the given input, i.e., whether the given input is in the correct syntax (of the language in which the input has been written) or not. It does so by building a data structure, called a Parse tree or Syntax tree. The parse tree is constructed by using the pre-defined Grammar of the language and the input string.

12. What is another name for lexical analyzer?

- a) Linear Phase
- b) Linear Analysis
- c) Scanning

13. What is the use of finite automata in lexical analyzer?

The finite automata concepts also used in various fields. In the design of a compiler, it is used in the lexical analysis to produce tokens in the form of identifiers, keywords and constants from the input program. In pattern recognition, it is used to search keywords by using string-matching algorithms, Ex. UNIX tools like awk, Procmail, and egrep. In network, finite automata concepts are used in the communication protocol.

14. What is another name for syntax analyzer?

Hierarchical Analysis & Parsing

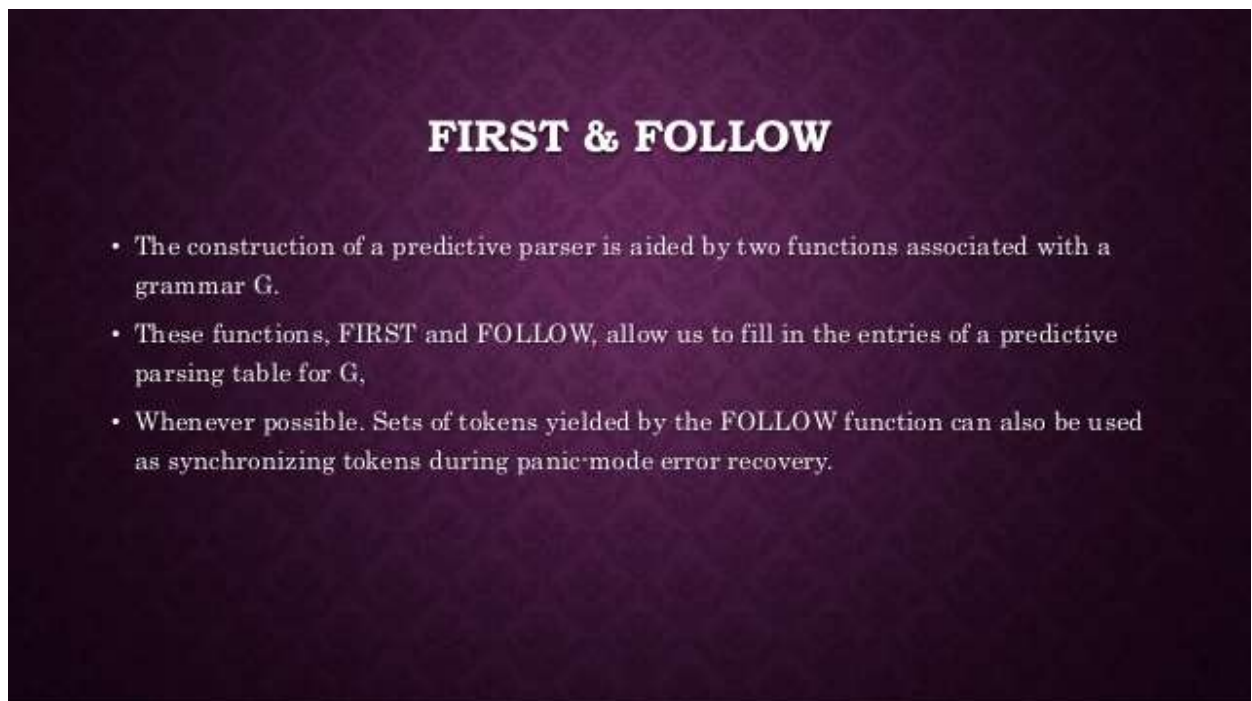
15. Left factoring

If more than one grammar production rules has a common prefix string, then the top-down parser cannot make a choice as to which of the production it should take to parse the string in hand.

16. Left recursion

A grammar becomes left-recursive if it has any non-terminal 'A' whose derivation contains 'A' itself as the left-most symbol. Left-recursive grammar is considered to be a problematic situation for top-down parsers. Top-down parsers start parsing from the Start symbol, which in itself is non-terminal. So, when the parser encounters the same non-terminal in its derivation, it becomes hard for it to judge when to stop parsing the left non-terminal and it goes into an infinite loop.

17. First and follow



FIRST & FOLLOW

- The construction of a predictive parser is aided by two functions associated with a grammar G.
- These functions, FIRST and FOLLOW, allow us to fill in the entries of a predictive parsing table for G,
- Whenever possible. Sets of tokens yielded by the FOLLOW function can also be used as synchronizing tokens during panic-mode error recovery.

18. Predictive parser table

Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string. The predictive parser does not suffer from backtracking. To accomplish its tasks, the predictive parser uses a look-ahead pointer, which points to the next input symbols. To make the parser backtracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar.

19. Top down and bottom-up parser

Top-Down Parsing

Top-down parsing technique is a parsing technique which starts from the top of the parse tree, move downwards, evaluates rules of grammar.

Bottom-Up Parsing

Top-down parsing technique is again a parsing technique which starts from the lowest level of the parse tree, move upwards and evaluates rules of grammar.

20.Steps for LL (1) parser

- a. Compute FIRST
- b. Compute FOLLOW
- c. Construct predictive parsing table using first and follow function.
- d. Parse the input string using predictive parsing table.

21.Operator precedence parser

A grammar that is used to define mathematical operators is called an operator grammar or operator precedence grammar. Such grammars have the restriction that no production has either an empty right-hand side (null productions) or two adjacent non-terminals in its right-hand side

22. What is semantic analysis

Semantic Analysis is the third phase of Compiler. Semantic Analysis makes sure that declarations and statements of program are semantically correct. It is a collection of procedures which is called by parser as and when required by grammar. Both syntax tree of previous phase and symbol table are used to check the consistency of the given code. Type checking is an important part of semantic analysis where compiler makes sure that each operator has matching operands.

23. Inherited Attribute

On other hand an attribute is said to be Inherited attribute if its parse tree node value is determined by the attribute value at parent and/or siblings' node. In case of $S \rightarrow ABC$ if A can get values from S, B and C. B can take values from S, A, and C. Likewise, C can take values from S, A, and B then S is said to be Inherited Attribute.

24. Synthesized Attribute

Synthesized attribute is an attribute whose parse tree node value is determined by the attribute value at child nodes. To illustrate, assume the following production $S \rightarrow ABC$ if S is taking values from its child nodes (A, B, C), then it is said to be a synthesized attribute, as the values of ABC are synthesized to S.

Chapter 06

25.What is intermediate code generation?

Intermediate code generator receives input from its predecessor phase, semantic analyzer, in the form of an annotated syntax tree. That syntax tree then can be converted into a linear representation, e.g., postfix notation. Intermediate code tends to be machine independent code.

26.Types of intermediate code generation.

1. Graphical Representation
2. Syntax tree & DAG
3. Postfix notation
4. Three address code
5. Quadruple, Triple, Indirect Triple.

27.What is 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. DAG can be understood here:

- Leaf nodes represent identifiers, names or constants.
- Interior nodes represent operators.
- Interior nodes also represent the results of expressions or the identifiers/name where the values are to be stored or assigned.

28.Difference between parse tree and syntax tree.

PARSE TREE	SYNTAX TREE
An ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar	A tree representation of the abstract syntactic structure of source code written in a programming language
Also known as parsing tree, derivation tree, and concrete syntax tree	Also known as abstract syntax tree
Contains records of the rules (tokens) to match input texts	Contains records of the syntax of programming language

29.What is code optimization

Optimization is a program transformation technique, which tries to improve the code by making it consume less resources (i.e., CPU, Memory) and deliver high speed. In optimization, high-level general programming constructs are replaced by very efficient low-level programming codes. A code optimizing process must follow the three rules given below:

- The output code must not, in any way, change the meaning of the program.
- Optimization should increase the speed of the program and if possible, the program should demand a smaller number of resources.
- Optimization should itself be fast and should not delay the overall compiling process.

30.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. Machine-dependent optimizers put efforts to take maximum advantage of memory hierarchy.

31.Machine Independent optimization

Machine independent optimization attempts to improve the intermediate code to get a better target code. The part of the code which is transformed here does not involve any absolute memory location or any CPU registers. The process of intermediate code generation introduces much inefficiency like: using variable instead of constants, extra copies of variable, repeated evaluation of expression. Through the code optimization, you can remove such inefficiencies and improves code. It can change the structure of program sometimes of beyond recognition like: unrolls loops, inline functions, eliminates some variables that are programmer defined.

32.What are issues in code generation?

- Input to the code generator
- Target program
- Memory management
- Instruction selection
- Register allocation
- Choice of evaluation order
- Approaches to code generation

33.What is basic block and flow graph

Flow graph is a directed graph. It contains the flow of control information for the set of basic blocks. A control flow graph is used to depict that how the program control is being parsed among the blocks. It is useful in the loop optimization.

A basic block is a straight-line code sequence with no branches in except to the entry and no branches out except at the exit. This restricted form makes a basic block highly amenable to analysis. Basic blocks form the vertices or nodes in a control flow graph.

Chapter 02

34.Forward reference problem

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.

35. MOT POT ST LT BT

POT: POT is the fixed length table.

MOT: It is a fixed length table i.e.; we make no entry in either of the passes. It is used to accept the instructions and convert/gives its binary opcode.

ST: Symbol table is used for keeping the track of symbol that are defined in the program. It is used to give a location for a symbol specified.

LT: Literal table is used for keeping track of literals that are encountered in the programs. We directly specify the value; literal is used to give a location for the value.

BT: A table, the Base Table (BT) that indicates which registers are currently specified as base registers by USING pseudo-ops and what are the specified contents of these registers.

36. RR RX instruction format

- **RR:** The first operand is register and second operator is also register

| Opcode | Operand1 | Operand2 | |-----|-----|-----|

- **RX:** The first operand is register and second operator is an address. The address is generated by considering Base + index + displacement.

| Opcode | Operand1 | Operand2 | |-----|-----|-----|

Address = (Base + index + displacement)

Chapter 03

37. Macro and Macro processor

- **Macro:-**
 - Macro instructions are single line abbreviations for group of instructions.
 - Using a macro, programmer can define a single “instruction” to represent block of code.
 - Ex: #define in C and MACRO <Macro Name> in Assembly language
- **Macro processor-**
 - Macro instructions are an extension of the basic assembly language that can simplify debugging and program modification during translation.
 - To process macro instructions, most assembler use pre-processors known as Macro processors

38. Features of macro

- Associating macro parameters with their arguments
- Delimiting macro parameters
- Directives related to arguments
- Automatic label generation
- Machine independent features

39. Macro conditional pseudo-opcode

AIF: Arithmetic operations are performed by AIF and if nested is true then only are branched. It is conditional opcode.

AGO: It is similar to ‘go-to’ statement in C/C++. It is an unconditional pseudo-opcode. It specifies a label appearing on some other statements in the macro definition.

Chapter 04

40.What is linker

Linker is a program in a system which helps to link an object module of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also link a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader.

41.What is loader

A loader is a system program, which takes the object code of a program as input and prepares it for execution. Programmers usually define the Program to be loaded at some predefined location in the memory. But this loading address given by the programmer is not be coordinated with the OS. The loader does the job of coordinating with the OS to get initial loading address for the .EXE file and load it into the memory.

42. Different types of loader

1. "Assemble (or Compile) and Go" Loader
2. General Loader Scheme
3. Absolute Loader
4. Subroutine Linkages
5. Relocating Loader
6. Direct-Linking Loader
7. Linkage Editors (Binders)
8. Dynamic Loading Loader
9. Dynamic Linking Loader

43.functions of loader

- **Allocation:** Allocate space in memory for the Programs
- **Linking:** Resolve symbolic references between object decks.
- **Relocation:** Adjust all address dependent locations, such as address constants, to correspond to the allocated space.
- **Loading:** Physically place the machine instructions and data into memory