1. What is a programming language and why do we need them?

A: Programming languages are a way to communicate with the computer in the form of instructions comprising of all the necessary details which eliminate any possibility of ambiguity which may require the computer to take its own decisions. The programing language enables us to write efficient programs and develop online solutions such as- mobile applications, web applications, and games, etc.

2. What is the different between compiled vs. interpreted languages? Why do we need both?

A: Compile Language : whole program is first comiled, any syntax and symantic error, if there is restricts the formation of executable. At least 2 steps from source code to executable. Executable created after compiling is executed by the computer's CPU. Relatively faster, better performance

Interpreted language instructions are executed directly without the whole thing being compiled first. Debugging happens at runtime. Only one step from source code to execution. Instructions are not directly executed by the target machine. Relatively slower.

3. Why do we have precedence and associativity for operators?

A: Operator Precedence is a set of rules which tell us which operations are to be evaluted first in an arithmatic expression comprising of multiple operators.

Associativity is a set of rules for an operand to get evaluated with the left or right operator in the absence of parenthesis.

These rules are there to remove any ambiguity while solving an arithmatic expression and to exnusre that a particular expression evaluates to the same value across unrelated machines.

4. What is a token and what is a lexeme?

A: Lexeme = A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Token = A token is a pair consisting of a token name and an optional attribute value. The token name is an abstract symbol representing a kind of lexical unit, e.g., a particular keyword, or sequence of input characters denoting an identifier. The token names are the input symbols that the parser processes.

5. Check out C/C++ grammar - Write a grammar for a simple assignment operation assuming only the following operators: =, +, -, \*,

A: expression : NUMBER | expression OPERATOR expression | '(' expression ')';

grammer : IDENTIFIER '=' expression;

NUMBER : 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | NUMBER NUMBER

OPERATOR : + | - | \* | /

IDENTIFIER : \_ | NONDIGIT | IDENTIFIER IDENTIFIER | IDENTIFIER NUMBER

NONDIGIT : a | b | ... |z | A | B | .. | Z

6. What is the difference between syntax and semantics?

A: Syntax is the study of sentence structure and the rules of grammar. Semanticsis the study of the meaning of sentences.

7. What all do the syntax and semantic analyzers do?

A: A syntax analyzer or parser takes the input from a lexical analyzer in the form of token streams. The parser analyzes the source code (token stream) against the production rules to detect any errors in the code. The output of this phase is a parse tree.

A semantic analyser uses syntax tree and symbol table to check whether the given program is semantically consistent with language definition. It gathers type information and stores it in either syntax tree or symbol table. This type information is subsequently used by compiler during intermediate-code generation

8. How does the CPU instruction pipelining work?

A: Instruction pipelining partitions the execution process into multiple independent steps capable of occurring in parallel. Instructions traverse these partitions one stage at a time. Once an instruction progresses to the next step, the next instruction can take its place in the pipeline, and so on.

9. What all optimizations does a compiler perform? Explain any one with an example.

A: • Peephole optimizations

-eg. a multiplication of a value by 2 might be more efficiently executed by left-shifting the value or by adding the value to itself

• loop optimisations

• Local optimizations

• Global optimizations

• Prescient store optimizations

• Interprocedural, whole-program or link-time optimization

• Machine code optimization and object code optimizer

10. What is the difference between a declaration vs. a definition?

A: A declaration is a way of introducing a variable to the compiler with all the necessary features to evaluate syntax and semantics of the containing instructions.

A Definition is providing information about what actually does this variable do.

11. What are the stages of an exe generation?

A: 1. Preprocessing - Copy paste of library source code

2. Compiling - preprocessed code is translated to assembly instructions specific to the target processor architecture.

3. Assembly - an assembler is used to translate the assembly instructions to object code.

4. Linking - The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in

12. When and why do we need to link?

A: The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in

13. Where does the program execution start in a C/C++ program?

A: The function named main is a special function in all C++ programs; it is the function called when the program is run. The execution of all C++ programs begins with the main function, regardless of where the function is actually located within the code.

14. Compile a C/C++ program in such a way that it does not need the default main (or its variants to exist)

A: If we don’t want to run the main function at startup, we have to change the entry point to different function.

cl /c main.cpp

link main.obj /ENTRY:DummyFunction /SUBSYSTEM:CONSOLE

It changes the entry point to DummyFunction instead of the main function.