1. What is one example of a lack of orthogonality in the design of C?

   C has two kinds of structured data types, arrays, and structs. Structures can be returned from functions but arrays cannot.

2. What primitive control statement is used to build more complicated control statements in languages that lack them?

   The selection statement plus GOTO is used to build more complicated control statements like the FOR loop statement.

3. What is the name of the category of programming languages whose structure is dictated by the von Neumann computer architecture?

   A language is Imperative if it's designed around prevalent computer architecture. In von Neumann computer, both data and programs are stored in the same memory, while the CPU is separated. Hence, the instructions and data must be transmitted to the CPU and sent back to the memory after it's processed.

4. What is an example of two language design criteria that are in direct conflict with each other?

   Reliability and Cost of Execution. For example, Java definition demands that all references to arrays must be checked to ensure that the index/indices are in range. This step adds a great deal of cost to execute the Java program that contains large numbers of references to arrays. C does not require index range checking, so C programs execute faster than semantically Java equivalent, although Java is more reliable.

5. What role does the symbol table play in a compiler?

   The symbol table serves as a database for compilation. The primary contents of the symbol table are the type and attribute information of each user-defined name in the program. The information is placed by lexical and syntax analyzers and is used by semantic analyzer and code generator.

6. Why is the von Neumann bottleneck important?

   It's the primary thing that limits the speed factor of von Neumann architecture computers.

7. Why were linguists interested in artificial intelligence in the late 1950s?

   Linguists were concerned with natural language processing, as they concluded that some methods must be developed to allow computers to process symbolic data in linked lists. At the time, most computation was on numeric data in arrays.

8. What language was designed to describe the syntax of ALGOL 60?

   BNF, Backus–Naur Form.

9. What is a nonprocedural language? Provide an example.

   Nonprocedural language is concerned with the WHAT not the HOW. These languages is where you specify what conditions the answer should satisfy, but not how to obtain it. An example of this would be SQL, Structured Query Language.

10. What populates the Smalltalk world? Why is Smalltalk historically so important?

    Objects populate the Smalltalk world. Smalltalk was so important because it was the first programming language that fully supported object-oriented programming. This lead to the widespread support of OOP designed languages due to its capabilities.

11. LISP began as a pure functional language but gradually acquired more and more imperative features. Why?

    LISP was to meet the demand for artificial intelligence as a functional programming language, but there soon emerged different dialects, cleaners, that were more modern and imperative and began to deviate from the functional form into Scheme. Common LISP then

combined the different forms into a single form that was more imperative, including assignment and iteration and had a large number of data types and structures, for example records, arrays, complex numbers and character strings.

12. Describe the operation of a general language generator.

    A general language recognizer is capable of reading strings of characters from the alphabet. It would analyze the given string and it would either accept or reject the string based from the language given, separating correct sentences from those that are incorrectly. A recognizer is used in the syntax analysis part of the compiler. In this role, the recognizer doesn't need to test all possible strings of characters from a set to determine whether it's in the language. The syntax analyzer just determines whether the given programs are syntactically correct.

13. Describe the operation of a general language recognizer.

    A general language generator is used to generate the sentences of the language. It generates unpredictable sentences that have limited usefulness as language descriptor. However, some have preference over certain forms of generators over recognizer, since they can be more easily read and understood. However, the syntax-checking portion of the compiler, also a language recognize, isn't as useful a language descriptor for programmers primarily due to it being used in a trial-and-error mode.

14. What is the difference between a sentence and a sentential form?

    A sentence is a sentential form that has only terminal symbols.
    A sentential form is every string/sequence of grammar symbols in the derivation derived in zero or more steps from the start symbol.

15. Distinguish between static and dynamic semantics.

    Dynamic semantics are defined during execution time and can change.
    Static semantics are known at compile time and do not change.

16. What purpose do predicates serve in an attribute grammar?

    Predicate functions in an attribute grammar state the static semantic rules that a language needs to follow. A false value in a predicate function indicates that there is a violation of syntax or static semantics and thus determines whether an action is allowed or not.

17. What is the difference between a synthesized and an inherited attribute? Provide an example.

    Synthesized attributes are the result of the attribute evaluation rules. Inherited attributes are passed down from parent nodes. In some approaches, synthesized attributes are used to pass semantic information up the parse tree, while inherited attributes help pass semantic information down and across it.

18. How is the order of evaluation of attributes determined for the trees of a given attribute grammar?

    The parse tree is evaluated based on its underlying BNF grammar, with a possibility of having empty set of attributes values attached to each node

19. What is the primary use of attribute grammars?

    An attribute grammar is an extension to a context-free grammar with the primary purpose to allow certain language rules to be described, such as type of compatibility. An attribute grammar is a formal way to define attributes for the productions of a formal grammar, associating these attributes to values. The evaluation occurs in the nodes of the abstract syntax tree, when the language is processed by some parser or compiler.

20. Explain the primary uses of a methodology and notation for describing the semantics of programming languages.

Methodology and notation are used to make the reader, computer or human, with more easily to understand the program. For example "for" for blocking statements, and "=" is used for formulas.

21. Why can machine languages not be used to define statements in operational semantics?

    Machine language cannot be used to define statements in operational semantics because it's only capable comprehend true or false, '1' or '0'. Semantics are more than just combinations of true and false. The individual steps in the execution of machine language, the resulting changes to the state of the machine are too small and too numerous, and the storage of a real computer is too large and complex.

22. Describe the two levels of uses of operational semantics.

    At the highest level, natural operational semantics occurs, where the interest is in the final result of the execution of a complete program. At the lowest level, structural operational semantics are used, which can be used to determine the precise meaning of a program through an examination of the complete sequence of state changes that occur when the program is executed.

23. Provide three reasons why syntax analyzers are based on grammars?

    BNF grammar descriptions of the syntax of programs are clear and concise, both for humans and for software systems.

    The BNF grammar descriptions can be used as a direct basis for the analyzer.

    BNF grammar descriptions are modular, thus being easy to maintain.

24. Why is lexical analysis separated from syntax analysis?

    Simplicity, techniques for lexical analysis are less complex than those required for syntax analysis.

    Efficiency, although it pays to optimize the lexical analyzer, because lexical analysis requires a significant portion of total compilation time.

    Portability, the lexical analyzer reads input program files and often includes buffering of that input, it is somewhat platform dependent.

25. Define lexeme and token and when, where and how are they used in the compilation process?

    Lexemes are logical groupings that the lexical analyzer gathers characters into. They are formal descriptions of the syntax of programming languages that don't include descriptions of the lowest-level syntactic units.

    Tokens are the internal codes for categories of lexeme groupings.

26. What are the primary tasks of a lexical analyzer?

    A lexical analyzer serves as the front end of a syntax analyzer. Technically, lexical analysis is a part of syntax analysis. A lexical analyzer performs syntax analysis at the lowest level of program structure.

27. What is a state transition diagram and how are they used in the compilation process?

    Statement transition diagrams, or state diagram, are a way of describing the time-dependent behavior of a system. They are a directed graph with the nodes of a state diagram labeled as state names, and the arcs are labeled with the input characters that lead to the transitions among states, which may include actions the lexical analyzer must perform for transitions.

28. Why are character classes used, rather than individual characters, for the letter and digit transitions of a state diagram for a lexical analyzer?

    Suppose we need a lexical analyzer that recognizes only arithmetic expressions, including variable names and integer literals as operands. Assume that the variable names consist of strings of uppercase letters, lowercase letters, and digits but must begin with a letter. Names have no length limitation. The first thing to observe is that there are 52 different characters, any uppercase or lowercase letter, that can begin a name, which would require 52

transitions from the transition diagram's initial state. However, a lexical analyzer is interested only in determining that it is a name and is not concerned with which specific name it happens to be. Therefore, we define a character class named LETTER for all 52 letters and use a single transition on the first letter of any name.

29. What are the two distinct goals of syntax analysis?

    The two distinct goals of syntax analysis are first, the syntax analyzer must check the input program to determine whether it is syntactically correct. When an error is found, the analyzer must produce a diagnostic message and recover. The second goal of syntax analysis is to produce a complete parse tree, or at least trace the structure of the complete parse tree, for syntactically correct input.

30. Describe the differences between top-down and bottom-up parsers. Make sure you describe when and where each is used.

    Top-down: tree is built from the root downward to the leaves. Top-down parsers are easier to write, but are generally less efficient and require the grammar for more simplistic programs.

    Bottom-up: tree is built from the leaves upward to the root. All commonly used parsing algorithms operate under the same constraint that they never look ahead more than one token into the input program. The most commonly used bottom-up parser, the LALR(1) parser, is complex. It's easier to write "parser generators" or "yacc" programs for these parsers, which will automatically generate source code for these complex parsers.

31. Define static, stack-dynamic, explicit heap-dynamic, and implicit heap dynamic variables. What are their advantages and disadvantages?

    Static: Memory is allocated before execution, and stays throughout. It is runtime efficient, and has history-sensitive subprograms, but does not have recursion.

    Stack-dynamic: Memory is allocated from system stack when declaration is elaborated. It has recursion and conserves memory, but does not have history-sensitive subprograms and is less run-time efficient than static.

    Explicit heap-dynamic: Memory is allocated and freed by the programmer from the heap. It has flexible storage management, but is less efficient than stack-dynamic variables and is error prone in pairing new and delete statements.

    Implicit heap-dynamic: Heap memory is allocated and freed by assignments. It has high external flexibility, but the run-time cost to maintain all the dynamic attributes reduce compiler error detection.

32. Define lifetime, scope, static scope, and dynamic scope and give examples of each.

    Lifetime: When the memory is allocated for a variable, or how long a variable existences. A time during which the variable is bound to a specific memory location.

    Scope: Portions of the program where variable can be referenced or set. The range of statements in which the variable is visible. A variable is visible in a statement if it can be referenced in that statement.

    Static scope: The scope based on the source program text, "static ancestors", and used to connect a name reference to a variable, you or the compiler must find the declaration.

    Dynamic scope: The scope based on the run-time call sequence of program units, not their textual layout, "temporal versus spatial". This scope references to variables that are connected to declarations by searching back through the chain of subprogram calls that forced execution to this point.

Flex-Bison Problem

Starting with the simple Flex/Bison program I did in class, convert the calculator from infix to postfix. By way of example, my program produced the following output:

```
> 5 * 4.0;
```

> 20.0

The postfix version should work as follows:

> 4 9 +
> 13

Control-d should terminate the program.

You simply need to send me your modified versions of the ex1.l and ex1.ypp files that I used in my example. Of course, you should make sure they work first. I will use g++ to test them. Please call them lastname_firstname_postfix.l and lastname_firstname_postfix.ypp.