Introduction

1. What is the difference between machine language and assembly language?

Answer: A Machine language is the sequence of bits that directly controls a processor. An Assembly language is a one-to-one correspondence between mnemonics and machine language instructions.

2. In what way(s) are high-level language an improvement on assembly language? Are there circumstances in which it still make sense to program in assembler?

Answer: High-level languages was an improvement over assembly languages by allowing machine-independent language programs to be written, particularly one in which numerical computations could be expressed resembling mathematical formulae.

3. Why are there so many programming languages?

Answer: Evolution—Computer Science is new and constantly evolving, Special Purposes—for a specific problem domain, and Personal Preference.

4. What makes a programming language successful?

Answer: Expressive Power, Ease of use for Novice, Ease of Implementation, Standardization, Open Source, Excellent Compilers, Econoomics, Patronage and Inertia.

5. Name three languages in each of the following categories: von Neumann, functional, object-oriented. Name two logic languages. Name two widely used concurrent languages.

Answer: Von Neumann: C, Ada, Fortran

Functional: Lisp/Scheme, ML, Haskell

Object-Oriented: Smalltalk, Eiffel, Java

Logic: Prolog, SQL

Concurrent: Java, C#.

6. What distinguishes declarative languages from imperative languages?

Answer: Declarative languages focus on what the computer should compute. Declarative languages are in some sense "higher level"; they are more in tune with the programmers point of view.

Imperative languages focus on how the computer should do computation.

7. What organization spearheaded the development of Ada?

Answer: U.S. Department of Defense

8. What is generally considered the first high-level programming language?

Answer: Fortran

9. What was the first functional language?

Answer: Lisp

10. Why aren't concurrent languages list as a separate family in Figure 1.1?

Answer: the distinction between concurrent and sequential languages is mostly independent of the classification. Most concurrent programs are currently written using special library packages or compilers in conjunction with sequential language such as Fortran or C.

11. Explain the distinction between interpretation and compilation. What are the comparative advantages and disadvantages of the two approaches?

Answer: The compiler translates the high-level source program into an equivalent target program (typically in machine language), and then goes away. The interpreter stays around for the execution of the application.

In general, interpretation leads to greater flexibility and better diagnostics (error messages) than does compilation. Interpretation also copes with languages in which fundamental characteristics of the program, such as the size and types of variables, or even which names refer to which variables, can depend on the input size. Compilation leads to better performance. An interpreter needs to look up a variables location on every occurence. A compiler compiles a variable once and guarantee its location.

12. Is Java compiled or interpreted (or both)? How do you know?

Answer: Java is both interpreted and compiled. Java code is compiled down to Java Byte code. Then the Java byte code then executed on the JVM.

13. What is the difference between a compiler and a preprocessor?

Answer: A Preprocessor's goal is to produce an intermediate form that mirrors the structure of the source.A compiler thoroughly analyzes the code and produces a intermediate form that does not bear resemblence to the source.

compilers attempt to "understand" their source; preprocessors do not. Preprocessors perform transformations based on simple pattern matching, and may produce output that will generate error messages when run through a subsequent stage of translation.

14. What was the intermediate form employed by the original AT&T C++ compiler?

Answer: The original AT&T C++ compiler generated C code.

15. What is P-code?

a stack-based language similar to the bytecode of modern Java compilers.

16. What is bootstrapping?

Answer: Most compilers are self-hosting: they are written in the language they compile. In a nutshell, to bootstrap a compile, one starts with a simple implementation—often an interpreter—and use it to build progressively more sophisticated versions.

17. What is a just-in-time compiler?

Answer: Just-in-time compilation invoke the compiler on the fly, to translate newly created source into machine language commonly bytecode to machine language.

18. Name two languages in which a program can write new pieces of itself ”on the fly”

Answer: Lisp and Prolog

19. Briefly describe three ”unconventional” compilers—compilers whose purpose is not to prepare a high-level program for execution on a general-purpose processor.

Answer: TeXcompile high-level document descriptions into commands for a laser printer or phototypesetter.

Query language processors for database systems translate languages like SQL into primitive operation on files.

Compilers that translatelogic-level circuit specifications into photographic masks for computer chips.

20. List six kinds of tools that commonly support the work of a compiler within a larger programming environment.

- Debuggers
- Style Checkers
- Configuration management
- Perusal Tools
- Profilers
- Preprocessors

21. Explain how an integrated development environment (IDE) differs from a collection of comman-line tools.

Answer: The editor for an IDE may incorporate knowledge of language syntax, providing templates for all the standard control structures, and checking syntax as it is typed in. Internally, the IDE likely maintains not only a program’s source and object code, but also a partially compiled internal representation.

22. List the principal phases of compilation, and describe the work performed by each.

Answer: Scanner—also known as lexical analysis. The principal purpose of the scannr is to simplify the task of the parser, by reducing the size of the input and by removing extraneous characters like whitespace. The scanner typically removes comments and tags tokens with line and column numbers, to make it easier to generate good diagnostics in later phases.

Parsers organize tokens into a parse tree that represents higher-level constructs (statements, expressions, subroutines, etc) in terms of their constituents following a context-free grammar.

Semantic Analysis—is the discovery of meaning in a program. The semantic analyzer recognizes when multiple occurrences of the same identifer are meant to refer to the same program entity, and ensures that the uses are consistent. Most languages the semantic analyzer also tracks the types of both identifiers and expressions. The semantic anaylzer typically builds and maintains a **symbol table** data structure.

Target Code Generation—translates the intermediate form into the target language.

23. List the phases that are also executed as part of interpretation.

Answer: The front end of an interpreter is the same a compiler **Scanner (Lexical analysis), Parser (syntax analysis), Semantic analysis and intermediate code generation, and tree walking routines**— which "executes" (interprets) the intermediate code directly.

24. Describe the form in which a program is passed from the scanner to the parser; from the parser to the semantic analyzer; from the semantic analyzer to the intermediate code generator.

Answer: The **scanner** takes in a Character stream and out puts a Token Stream.

The **parser** takes in a token stream and outputs a Parse Tree.

The **semantic analyzer** takes in a parse tree and outputs a Abstract Syntax tree or some other intermediate form.

The **Target Code generator** takes in some intermediate form and outputs a modified target language.

25. What distinguishes the front end of a compiler from the back end?

Answer: The front-end serves to figure out the meaning of the source program. The back-end serves to construct an equivalent target program.

26. What is the difference between a phase and a pass of compilation? Under what circumstances does it make sense for a compiler to have multiple passes?

Answer: A phase discovers information of use to later phases. A pass is a phase or set of phases that is serialized with respect to the rest of compilation.

27. What is the purpose of the compiler's symbol table?

Answer: a symbol table that maps each identifer to the information known about it. Among other things, this information includes the identifer's type, internal structure (if any), and scope.

28. What is the difference between static and dynamic semantics?

Answer: Semantic rules that can be checked at compile time (or in the front-end of the compiler) are static semantics. Semantic rules that must be checked at run time (or in later phases of an interpretation) are dynamic semantics.

29. On modern machines, do assembly language programmers still tend to write better code than a good compiler can? Why or why not?

Answer: On a modern machine a compiler will generate better code than assembly language programmers. Due to increases in hardware complexitiy and continuing improvements in compiler technology.