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

Assembly language has the ability to form mechanism that require close to 0 work. Assembly language uses mnemonic sequences rather than numeric operation codes and it also can use the symbolic labels rather than manually calculating the offsets. It also protects are from silly mistakes. Other than these differences they are pretty much the same.

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

High level languages help the programmer a lot to be efficient with contracting and maintaining a program. I believe there is only one circumstance which is if we are doing which needs an incredible performance, then having assembly helps us control and optimize directly.

1. Why are there so many programming languages?

I believe there are couple reasons for it, one of them being that languages evolve and change, then we can say some of them were created for special reasons and then personal preference is also an important reason.

1. What makes a programming language successful?

If it’s easy to use, easy to implement and it has expressive power then it’s a good language but it also needs open-source for users and good compliers for debugging for a good successful language.

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

* Java
* Eiffel
* Smalltalk
* C
* Ada
* Fortran
* Scheme
* DrRacket
* ML
* Prolog
* SpreadSheets
* Java
* C#

1. What distinguishes declarative languages from imperative languages?

It hides the implementation details.

1. What organization spearheaded the development of Ada?

It was United States Department of Defense(DoD).

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

Fortran.

1. What was the first functional language?

Lisp.

1. Why aren’t concurrent languages listed as a separate family in Figure 1.1?

It’s because the distinction between concurrent and sequential execution is mostly unique and by its own of its classification and that’s why I think it was listed as separate families.

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

Well a compiler translates the source program into an equivalent machine language but an interpreter implements a virtual machine which also uses a machine language which happens to be a high-level programing language as well.

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

Java can be both because a compiler can produce code and then execute that with an interpreter (virtual machine).

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

Preprocessor is basically preps the code for execution by deleting white space, comments, and groups characters into tokens. Then it creates an analysis. A compiler goes through the analysis and transformation and execute.

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

C++ compiler first converts the C++ into C and then it is given as an input to the compiler. C++ compiler basically compiles C code.

Extra Info: That’s why C++ was first called C with classes.

1. What is P-code?

It is a stacked based language and it resembles a little bit to byte code of modern java compilers.

1. What is bootstrapping?

It is when a simple implementation of an interpreter improves to a more complex version until the compiler is ready.

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

It acts as the translator for byte code into machine language right before execution.

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

* Lisp
* Prolog

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

Query Language processors for database systems are also compilers. They translate languages like MySQL into primitive operations on files.

TEX and Troff are actually compilers. They translate high-level document descriptions into commands for printers.

1. List six kinds of tools that commonly support the work of a compiler within a larger programming environment.
2. Explain how an integrated development environment (IDE) differs from a collection of command-line tools.
3. List the principal phases of compilation, and describe the work performed by  each.
4. List the phases that are also executed as part of interpretation.

* Lexical analysis(Scanner)
* Syntax analysis(Parser)
* Semantic analysis and intermediate code generation
* Machine-independent code improvement(opt)
* Target code generation
* Machine-specific code improvement(opt)

1. 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.

Token stream

Parse Tree

Abstract Syntax Tree

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

Front-end can be shared by different compilers for couple of target language and the back-end can be shared by compilers for more than one source language.

1. 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?

Each phase discovers new information to use it during the latter phases.

A pass is a phase or set of phases that is serialized with respect to the rest of the complication. It doesn’t start until one finishes.

1. What is the purpose of the compiler’s symbol table?

It is a data structure that maps information about identifiers type, internal structure and the scope.

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

Static semantics language is only indirectly related to the meaning of programs during execution. It has to do with legal forms of the program.

Dynamic semantics language means expressions, statements and program units. Because of the power of the naturalness of the available notation, describing syntax is relatively simple matter.

1. On modern machines, do assembly language programmers still tend to write

better code than a good compiler can? Why or why not?

Compilers can usually generate better code than can human assembly language programmers. Especially with those modern microprocessor architectures.