1. What is the difference between a compiler and an interpreter?

A compiler outputs some sort of code, either intermediate representation (bytecode, etc) or pure machine code, which is then executed directly by the computer. An interpreter takes source code and figures out how to execute it itself, without emitting any external representation.

1. What are the advantages of:
	* An interpreter over a compiler?

An interpreter can give better error diagnostics, is the answer in the book. This is because it is still operating directly on the source code. Additionally, not having to wait for compilation is sometimes a benefit.

* + A compiler over an interpreter?

Compiled code is generally much faster. The actions to be performed have already been figured out by the time the code is running, and many optimizations can be applied that can’t necessarily be run by an interpreter

1. What advantages are there to a language-processing system in which the compiler produces assembly language rather than machine language?

It is “easier to produce and debug.”

1. A compiler that translates a high-level language into another high-level language is called a source-to-source compiler. What advantages are there to using C as a target language for a compiler?

C compilers are insane. They produce compact, highly-performant code. If you want your code to run fast, it is probably easier to produce reasonable C code and leverage the tends of thousands of man-hours that have been put into optimizing C compilers than to do it yourself. Also, you gain some benefits of readability, although in my (limited) experience, generated code is usually not readable even in high level languages.

1. Describe some of the tasks that an assembler needs to perform.

Produce machine code from assembly code. This requires knowing about both the assembly language and the target architecture, and turning low-level abstractions such as ‘multiply two numbers’ into code that puts numbers in appropriate registers and such.