Section 1

**Here is a list of some of the operating system’s many responsibilities:**

1. Communicating with the computer user: receiving commands and carrying them out or rejecting them with an error message.

2. Managing allocation of memory, of processor time, and of other resources for various tasks.

3. Collecting input from the keyboard, touchpad, mouse, and other input devices, and providing this data to the currently running program.

4. Conveying program output to the screen, printer, or other output device.

5. Accessing data from secondary storage.

6. Writing data to secondary storage.

**Computer language :**

**machine language** is a collection of binary numbers.

**assembly language**, a language in which computer operations are represented by mnemonic codes rather than binary numbers and variables can be given names rather than binary memory addresses.

**high-level languages** that combine algebraic expressions and symbols taken from English.

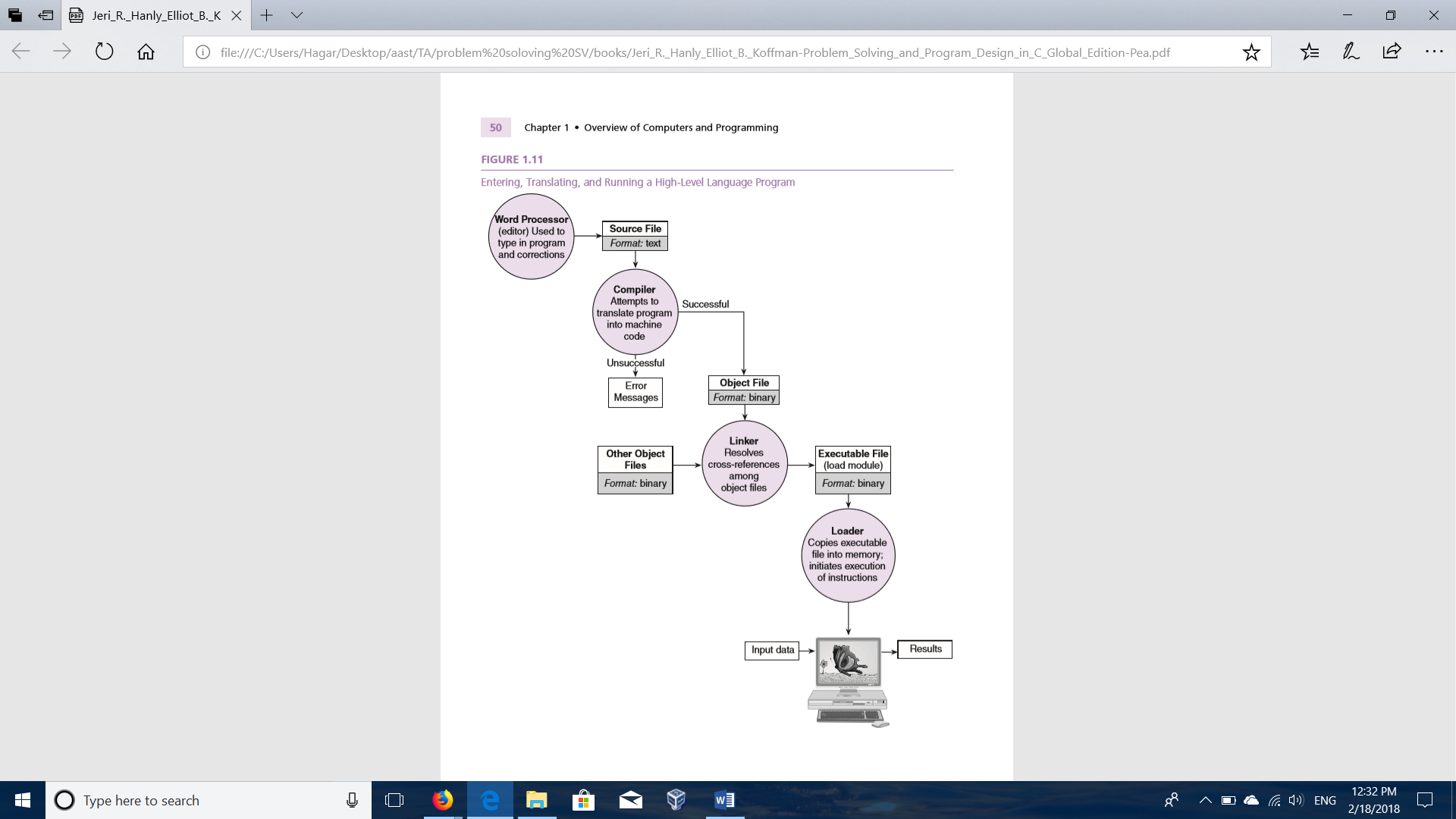
**integrated development environment (IDE),** a package that combines a simple word processor with a compiler, linker, and loader.

The program that does this translation is called a compiler. Illustrates the role of the compiler in the process of developing

and testing a high-level language program. Both the input to and the output from the compiler (when it is successful) are programs. The input to the compiler is a source file containing the text of a high-level language program. The software developer creates this file by using a word processor or editor. The format of the source file is text, which means that it is a collection of character codes. For example, you might type a program into a file called myprog.c. The compiler will scan this source file, checking the program to see if it follows the high-level language’s syntax (grammar) rules. If the program is syntactically correct, the compiler saves in an object file the machine language instructions that carry out the program’s purpose. Although an object file contains machine instructions, not all of the instructions are complete. High-level languages provide the software developer with many named chunks of code for operations that the developer will likely need. Almost all high-level language programs use at least one of these chunks of code called functions that reside in other object files available to the system. The linker program combines these prefabricated functions with the object file, creating a complete machine language program that is ready to run. For your sample program, the linker might name the executable file it creates myprog.exe.

myprog.exe is just stored on your disk, it does nothing. To run it, the loader must copy all its instructions into memory and direct the CPU to begin execution with the first instruction. As the program executes, it takes input data from one or more sources and sends results to output and/or secondary storage devices.

**Execution of a program**



Difference between Bottom-Up Model and Top-Down Model

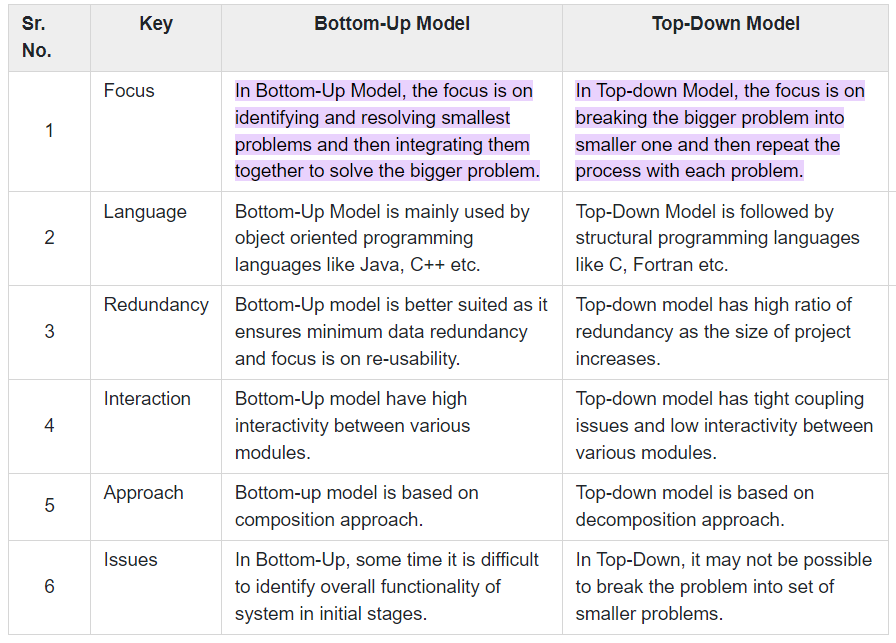
## **Bottom-Up Model**

Bottom-Up Model is a system design approach where parts of the system are defined in details. Once these parts are designed and developed, then these parts or components are linked together to prepare a bigger component. This approach is repeated until the complete system is built. Advantage of Bottom-Up Model is in making decisions at very low level and to decide the re-usability of components.

## **Top-Down Model**

Top-Down Model is a system design approach where design starts from the system as a whole. Complete System is then divided into smaller sub-applications with more details. Each part again goes through the top-down approach till the complete system is designed with all minute details. Top Down approach is also termed as breaking the bigger problem into smaller problems and solving them individually in recursive manner.

Following are the important differences between Bottom-Up Model and Top-Down Model.



**The software Development Method**

1. Specify the problem requirements.

2. Analyze the problem.

3. Design the algorithm to solve the problem.

4. Implement the algorithm.

5. Test and verify the completed program.

6. Maintain and update the program.

Apply the previous methodology using multiple examples