Chapter One: Software Development

1.5 Exercises

1. *Problem analysis and specification* involves determining:
   - output required to solve problem
   - input given
   - other info, such as software usage, feasibility of computer solution

   System design involves:
   - selecting appropriate data structures to organize the input data
   - designing algorithms needed to process the data efficiently

   Coding and integration involves
   - selecting appropriate programming language
   - translating algorithms into well-structured, well-documented, readable code

   Verification and validation may involve
   - "walking through the code" or, possibly proving correctness of algorithms
   - executing the software and correcting errors
   - modifying algorithms

   System maintenance may involve
   - fixing possibly obscure bugs
   - modifying software to improve performance
   - adding new features, etc.

2. Introductory programming problems are usually well-defined, simple, clearly stated and solvable. Real world problems are generally not well-defined, may be complex, may not have an obvious method of solution.

3. *Top-down* (or modular) design: Partition a problem into simpler subproblems, each of which can be considered individually. Repeat this until the subproblems obtained are simple enough to solve.
   Object-oriented design (*OOD*): Identify a collection of objects, each of which consists of data and operations on the data, that model the real-world objects in the problem and that interact to solve the problem.

4. Encapsulation, inheritance, polymorphism.

5. Structured data types are user-defined structures used to collectively organize data in a given problem. Typical examples include arrays, files, stacks, etc.

6. An algorithm is a procedure that can be executed by a computer. It must be:
   - unambiguous (clear what each instruction is intended to do)
   - simple enough to be done by a computer
   - finite (definite termination)

7. Pseudocode is shorthand notation for representing programs in texts, articles and design phases. It is a mixture of natural language (English) and features and syntax common to high-level languages.

8. Three control structures:
   - *sequential* -- steps performed serially, one after another in established sequence
   - *selection* -- one of several alternative actions is selected and executed
   - *repetition* -- one or more steps is performed repeatedly

9. Typical student programs are small (< 1000 lines), are used infrequently, and are standalone (not part of a system); errors don't have serious consequences and usually quite easy to find and correct; lifetimes of programs are short and thus require little or no maintenance.

   Typical real-world programs are large (thousands or millions of lines), used often, and are integrated with other software. Errors may serious — even tragic — results, and may be difficult to find; software may be used for a long period of time and require frequent maintenance.

10. *Syntax errors*: Incorrect syntax (e.g., misplaced or missing semicolons), misspelled words, missing declarations/ The compiler generates error messages that identify the source and type of error.

    Run-time errors: Division by zero, integer overflow, missing files, illegal memory access. Execution of the program will be terminated with an error message.

    Logical errors: Incorrect coding of algorithm instructions, improper initialization of variables (so garbage values are used), errors in the algorithms (e.g., not checking boundary conditions). The program executes, but incorrect results are produced.

11. Program maintenance is required to fix bugs, accommodate new uses of software (changes in environment), improve performance, or incorporate new features.

12. Many examples can be found on the Internet.