Programming Documentation Standard
Procedure-Oriented, February 1996

A. Introduction

A.1. Purpose of this document

This document describes a standard documentation format for programming work done in computer science courses. The intent is for the students to gain experience with standard documentation techniques, to improve the quality of programming work, and to facilitate the grading of students' work.

A.2. Organization of the standard description

This standard encompasses both physical and logical aspects of the documentation. Certain materials described herein are to be assembled in a prescribed manner to form the documentation package which will be submitted and graded in fulfillment of the requirements for the programming assignment. These materials also constitute a logical structure which guides the students' efforts from problem definition through solution and implementation to program verification and culmination of the programming effort. The following sections on content materials and packaging will explain both the physical and logical aspects, as well as the relationship between them.

A.3. Content of the documentation

The materials comprising the documentation cover the four phases of problem solution - **analysis**, **design**, **coding** and **testing**. In the analysis phase, the problem is defined and the *requirements documentation* is produced. In the design phase, a software solution to the problem is planned, organized and detailed to produce the *design documentation*. During the coding phase the pseudocode algorithm and the system architecture are converted to program code which, together with appropriate comments, becomes the *implementation documentation*. In the testing phase, the program is run to test whether it accurately produces the results specified in the problem definition. Difficulties encountered in the test runs are removed through debugging and making needed corrections. Test data and final test results are recorded in the *verification and validation documentation.*

B. Physical Aspects of the Documentation

B.1. Materials to be used in documentation

Three types of materials are used in the documentation.

Enclosure: A folder which is large enough to enclose all of the material. The upper left corner of the folder should contain a label with the following information: Course No., Section No., Problem No., Student Name (Last, Initials), UserID, and Due Date. For example,

CS 261	Sec. 1	Prob. # 2
Smith, James C.	jcs3233	Due: 11/5/95

Written Material: Written material should be printed on 8-1/2 x 11 inch sheets of paper. (If you must use the fan-fold green paper in the lab, format the text on the sheets, cut them down to about 8.5" width, separate the sheets, and staple them together so that they can be read like standard sheets.) You should use a word processor to write your high and intermediate level documentation. These documents can then be easily moved into the code as comments. This will also allow you to build the code around your design.

The first information that should appear at the top of the first page is the information that identifies the programmer. Include the same information here that you did on the outside of the folder, as described under *Enclosure.*

Computer Printout: A copy of the source code and a copy of script files showing test results should be included. The code and test results must also be trimmed to $8.5 \times 11''$ and separated. *Note: the code goes into part III and the test results into part IV of your documentation package.*

B.2. Grading Notes

1. Programming assignments are due at the **beginning** of the class period on the due date or the **exact** time specified on the assignment.
2. A draft of the requirements and design documentation will be due at least one week before the finished programming assignment, which will include a final version of all documentation.
3. You will not get full credit if the requirements and design documentation is missing and/or sloppily put together. On the other hand, you will get partial credit for a good analysis and a good design, even if your program is only a partial solution. **Code that does not compile will receive no credit**.
4. When you go to ask your instructor about your program, you are expected to take along all of the appropriate documentation.

B.3. Outline of contents for standard documentation folders

I. Requirements Documentation

   1. Description of the problem
   2. Input information
   3. Output information
   4. User Interface Information

II. Design Documentation

   1. System Architecture Description
   2. System Component Information
   3. Structure Chart[1]
   4. Pseudocode

III. Implementation Documentation

   1. Program code

IV. Verification and Validation Documentation

   1. Test data
   2. Test results
   3. Operating directions

C. Logical Aspects of the Documentation

The remainder of this document further explains the content of each of the four major sections of this documentation folder. Use the section titles and numbers shown. Please make sure that you do not change them.

I. Requirements Documentation

The purpose of this section of the documentation is to define the problem with sufficient detail so that the solution can be planned.

I.1. Description of the Problem

Name: Give a short title.
Problem Statement: Tell what needs to be done. (Approximately 1 or 2 sentences which provide a high level description of the problem to be solved.)
Problem Specification: Give a complete and detailed specification of the problem. State any assumptions you have made regarding the problem. This specification is intended to provide a real world description of the problem, its input, its output, and its processing. No implementation–specific details should be included.

[1] Not required for programs consisting of a single module.

I.2. Input Information

I.2.1. *Input files:*
Name: Give the name of each input file, if relevant.
Description: How is it used? What is its purpose?
Format: Explain how the data are organized and formatted in the input file. Specifically, include information related to the placement and appearance of data on lines (specification of horizontal or vertical spacing, if relevant).
Size: The maximum number of lines (or records, or items) expected (is the number fixed or variable? if variable, is there a minimum and/or maximum?)
Sample: Show a sample of properly formatted input.

I.2.2. *Input Item(s): (Repeat the following for each program input.)*
Description: Tell what the data element means (or is used for).
Type: Indicate its logical data type. (Ex. integer, alphanumeric, single digit, real, etc.)
Range of acceptable values: Identify the acceptable range for this program.

I.3. Output Information

I.3.1. *Output files:*
Name: Give the name of each output file.
Description: How is it used? What is its purpose?
Format: Explain how the data are organized and formatted in the output file or on the screen (if output is sent to screen). Specifically, include information related to the placement and appearance of data on lines (specification of horizontal or vertical spacing, if relevant), and labels used to identify data.
Size: The maximum number of lines (or records, or items) expected (is the number fixed or variable? if variable, is there a minimum and/or maximum?)
Sample: Show a sample of properly formatted output.

I.3.2. *Output Item(s): (Repeat the following for each program output.)*
Name: Give the name of the output variable.
Description: Tell what the data element means (or is used for).
Type: Indicate its data type. (Ex. integer, alphanumeric, single digit, real, etc.)
Range of acceptable values: Identify the acceptable range for this program.

I.4. User Interface Information

I.4.1. *Description:* The nature of the user interface, which determines how the user will interact with the program, should be described. (Typical types of user interfaces in common use include: menu selection, form fill-in, command language, and direct manipulation (graphical, point & click)).

I.4.2. *Sample:* Include illustrations of screens and/or dialogues.

II. Design Documentation

The purpose of this section of the documentation is to describe the plan for the solution of the problem.

The following definitions are given to make the remainder of this section clear.

System- A system consists of all components of the software product, including the driver that controls the other components. These components are the highest-level layer of the software architecture.

Component- A component is a collection of *related* items (perhaps several subroutines, types, constants, and variables that comprise an ADT) packaged together into one (separately compiled) unit. Therefore, a component's subroutines and data are closely related. This is referred to as having a high *degree of cohesion.*

There are three types of system components:

- *Data Abstraction Components* include types, constants, variables to define an abstract data type (ADT), and the subroutines defined to implement the operations on the ADT.

- *Functional Abstraction Components* include sets of subroutines related to carrying out a particular function/task and the types, constants, and variables needed by those subroutines. (These subroutines are "functionally" related, but are not the implementations of operations of some abstract data type.)
- The *Driver* component includes the main (driving) routine of the system and the parameters, types, constants, and variables required by the main routine

II.1. System Architecture Description

This section includes a list of each system component, including the system driver component. For each component briefly describe its role in the overall system. (A single function is not necessarily a system component.)

II.2. System Component Information

II.2.1. *Data abstraction components:* Data abstraction components consist of data structures and subroutines that define primitive operations on those structures. For each data abstraction, include the following text and information:

"This component is designed to define the ADT _____. This object is defined in terms of:"

(include data definition here)

"with the following operations/subroutines:"

(include operation/subroutine documentation here)

Subroutine documentation includes the following for each subroutine:

a. Name – of the program module
b. Description – brief (one sentence) description of its function or purpose.
c. Prototype
d. Input parameters – name, type, and purpose for each parameter
e. Output parameters – name, type, and purpose for each parameter
f. Input/Output parameters – name, type, and purpose for each parameter

II.2.2. *Functional abstraction components:* Functional abstraction components consist of collections of subroutines that work together to carry out a portion of the requirements of the overall system. All subroutines other than the main routine and those listed in Section II.2.1 should be covered in this section. For each functional abstraction component, include the following information and text:

"This component consists of the following subroutines:"

(include subroutine documentation here (see *Subroutine documentation* in section II.2.1 above.)

II.2.3. *System driver component:* This section is only required if your system driver has any formal parameters. In this case, include the subroutine documentation for the driver subroutine.

II.3. Structure Chart

Show the tree diagram of the subroutines in the program, indicating the interconnections among the subroutines. Label the subroutines with the same name used in the pseudocode. (See attached documentation sample.)

II.4. Pseudocode

This section should describe, in an easily readable and modular form, how the software system will solve the given problem. The term "pseudocode" is not intended to refer to a precise form of expression. Rather it refers to using standard English terms in a restricted manner to describe the algorithmic process involved. Good pseudocode must use a restricted subset of English, in such a way that it resembles a good high level programming language. Pseudocode must be formatted similarly to actual code. The pseudocode description of the problem should state the problem solution so clearly that it can easily be translated to the programming language to be used. Thus, it must include flow of control.

The pseudocode for the system driver should appear first. The pseudocode for subroutines in a system component should be grouped together, with the component identified.

III. Implementation Documentation

The purpose of this documentation is to present a well-engineered version of the program, along with information needed to clarify how it has been encoded.

III.1. Program Code (Source Code)

The source listing is required here.

Physical Organization of System Components: It is expected that different components of the system architecture will appear in different compilation units, provided that the implementation language/environment supports this type of organization. Information is to be shared among components through the inclusion of header files (when feasible).

Comments: You should import the program design from your intermediate level documentation and build your code around the design. Comments should be used when the encoding or translation of a design is not obvious.

- System documentation (should appear in your system driver component):

  Programmer information – (import from the beginning of written documentation)
  Problem statement – (import from Requirements Doc. part 1)
  Problem specification – (import from Requirements Doc. part 1)
  System architecture description – (import from Design Doc., part 1)

- Component documentation: Each component should include as introductory documentation, the name of the source file it is located in, and the general description of its role in the system from the System Architecture Description (Section II.1).
- Subroutine documentation: For each subroutine within the component, import from Design Doc., Subroutine documentation (Section II.2.1), parts b, d, e, and f.

Style: Programming style refers to those conventions that enhance the readability of programs. Some of those conventions include:

- Prettyprinting:
  Use indentation and skipped lines so that the visual appearance of a program listing mirrors its logical structure. (Be consistent with your indentation increments!). Declare only one variable per line. For each declared variable include a brief comment documenting its purpose. Write only one program statement per line.
- Meaningful identifier names:
  Well–chosen identifiers significantly enhance readability, and as such is considered a significant element of internal program documentation. Identifiers should
  - be meaningful; avoid cuteness, single-letter identifiers, meaningless abbreviations, identifiers that too closely resemble one another. (ex. HT is not a meaningful variable name for a hash table.)
  - be accurate (ex. COUNT and I are not the best names for an integer variables that index arrays.)
  - observe standards for abbreviations, prefixes, and suffixes.
- Organizational consistency:
  Be systematic in grouping and ordering of declarations. For example, declared variables might be grouped by similar role, or listed alphabetically, but should not appear in random order. The same applies for all other declarations, such as subprogram declarations.

IV. Verification and Validation Documentation

The purpose of this documentation is to demonstrate the operation of the program, describe how it is run on the machine, and present evidence of program verification and validation. (This information should be divided into the following three parts:)

IV.1. Test Data

Include a list of input data sets which thoroughly test the logic of the program and demonstrates that the program satisfies its requirements. Explain what requirement of the problem or segment of the code will be exercised by each set of test data.

IV.2. Test Results

Include a script file showing the results of running the program with the Test Data. The output listing should be marked, if necessary, so that corresponding input can be identified for each output. That is, if the test data or program logic being exercised by this test is not obvious, mark the output listing with this information.

IV.3. Operating Directions

Give the procedure for running the program. Minimally, you should specify the following:

1. The name and version number of the compiler used.
2. The names and locations of your program (source) file, executable file, and any data files used.
3. The names and locations of makefile(s) or step-by-step instructions to compile and execute your program.
4. If your program does not work, or has bugs, indicate so and explain what parts of the program worked and what restrictions and/or cautions must be exercised to avoid problems.