Programming Documentation Standard
Object-Oriented Version, January 2000

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 design is 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 Three Types of Materials Used in the Documentation:

Enclosure: A letter size manila folder without pockets should be sufficient to contain all of the material. On the tab of the manila folder place the following information: Course No., Section No., Problem No., Student Name (Last, Initials), UserID, and Due Date. For example,

```
CS 260              Sec. 1        Prob. # 2
Smith, James C.     jcs3233       Submitted: 2/5/00
```

Written material: Written material should be printed on 8-1/2 x 11 inch sheets of paper. Do NOT staple sheets. Sheets should be numbered. You should use a word processor to write your requirements and design 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*. Each source file also must include the same information.

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 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

Folders are due at the beginning of the class period of the due date and submissions are due at the exact time specified on the assignment.

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. When you go to ask your instructor or grader about your program, you are expected to take along all of the appropriate documentation.

B.3 Outline of the Contents for the Standard Documentation Folders

I. Requirements Documentation
1. Description of the Problem
2. Input Information-- n/a
3. Output Information
4. User Interface Information

II. Design Documentation
1. System Architecture Description
2. Object Information
3. System Driver Information
4. Diagrams

III. Implementation Documentation
1. Program code

VI. Verification & 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. Always use the section titles and numbers shown.

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.*

I.2 Input Information

I.2.1 Input Streams: (Repeat the following information for each input stream; that is, all files except the standard input file.)

Name: Give the names of each input stream.
Description: How is it used? What is its purpose?
Format: Explain how the data are organized and formatted in the input stream, 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 Items: (Repeat the following for each program input.)

Description: Tell what the input 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, if applicable.

I.3 Output Information

Output Streams: (Repeat the following for each output stream; that is, all files except the standard ouput file.)

Name: Give the name of each output stream.
Description: How is it used? What is its purpose?
Format: Explain how the data are organized and formatted in the output stream.
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.

Output Items: (Repeat the following for each value output by the program.)

Description: Tell what the output 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, if applicable.

I.4 User Interface Information

1.4.1 Description:

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

For a menu driven interface, show the menu and any other dialogs that the program may initiate.

- Show the menu exactly as it will be displayed by the program.
- For prompts, show the exact wording of what will be displayed.
- List all possible error messages, in the exact wording they will be displayed.
- Give the label and format of all the output generated by the program.
- If the program generates a table, give the exact format of the displayed table.

II. Design Documentation - discuss main [illegible] & objects

The purpose of this section of the documentation is to describe a plan for the solution of the problem using an object-oriented design method.

The solution to a problem will usually consist of a software system that is comprised of a main application and a set of interacting objects. The application consists of the algorithm that orchestrates the usage of the objects in the system.

Adhere to recommended good programming practices, such as:

- Make certain that components are loosely coupled; i.e, avoid global objects and variables.
- Implement operations using the most appropriate type of subroutines; i.e.;procedures or functions.
- Pass parameters by value or by reference as appropriate.
- Make certain that components are highly cohesive.
- Make a member function "const" whenever appropriate.
- Make parameters "const" whenever appropriate.

II.1 System Architecture Description

This section includes the main application and the objects that comprise the system. First list the objects and their respective classes, and then, for the application and each object briefly describe its role in the overall system. Make sure that you describe how the objects interact with the main driver, and with each other.

II.2 Information about the Objects

For each class specification, include the following text and information:

Class Information

Name: Name the class.
Description: Briefly describe its purpose.
Base Class: Identify the base class, if appropriate

Class Attributes (data members) (Repeat for each attribute.)
Name: Name the attribute.

Description: Briefly describe its function or purpose.
Type: Indicate its data type or class.
Range of acceptable values: Identify the acceptable range of values.
(Note: Usually the attributes of an object are placed in the private view of the class specification.)

Class Operations (member functions)(Repeat for each operation.)
Prototype: Give the operation/function prototype, including the type(s) and name(s) of formal parameter (s).
Description: Briefly describe the task it performs.
Precondition(s): Give input assertion(s) describing the truths that the operation expects at the moment the caller invokes the function
Postcondition(s): Give output assertion(s) describing the state of the computation at the moment the function terminates.
Cost analysis: specifiy using Big-OH notation, the expected run-type behavior of the operation.
Visibility: public, protected, or private

II.3 Information about the Main Application

In an object-oriented design the main application is the coordinating algorithm of the software product. The application may consist of several subroutines. Often the it is only responsible for processing user commands and then delegating tasks to objects.

This section describes how the main application controls the software. The description of the main application must include the detailed logic of the algorithym, including flow of control. This description must be presented in the form of a C++ main program with file inclusions, declarations and syntactically correct statements. This main program does not have to compile. If free functions exist in the main program, give the prototype for these functions along with the respective descriptions, preconditions, and postconditions.

If the main program does not handle the principal object manipulation, include a thorough C++ algorithm for the member function that is responsible for the job. As in the main program, this algorithm must contain file inclusions, declarations and syntactically correct statements.

All function calls (including calls to member functions) must specify all required parameters.

II.4 Design Diagrams

The design effort will be summarized in three diagrams:

Object Interaction Diagram
Show a diagram that illustrates the calling interaction of the system. This diagram consists of an ellipse for the system driver, a box for each object, and directed lines from clients to servers. Each box contains the name of the object and its class and its operations.

Aggregation Diagram
Show an aggregation diagram for all classes that have aggregates. The aggregation diagrams will be kept very simple. One diagram will be needed for each aggregation relationship. Each relationship will include the object that contains or exclusively manages the object of another class, and the object of that other class. Each class will be represented exclusively by its name, and the relationship will be indicated by a directed line, from container to contained. On the directed line, the cardinality (1:1, 1:n, etc.) of the

relationship will be indicated.

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 included here. The following describes the required organization and internal documentation for the program.

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). The first thing to appear on every source file is the programmer identification and the authenticity certification.

Comments: You should import the program design information from your design 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 main application 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.)
- 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 (import from Design Doc, section II.1).
- Class and object documentation: For each class import from Design Doc., the relevant information from section II.2.

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 data item per line. For each declared data item 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 is not the best name for an integer that indexes an array; object names should indicate entities, operation names should indicate actions)
  - 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 Plan

Include a list of input data sets which thoroughly test the logic of the program and demonstrate that the program satisfies its requirements. Explain what requirement of the problem will be exercised by each set of test data. (The test plan should be prepared at the time the Requirements Documentation is written.)

IV.2 Test Results

Include a script file showing the results of running the program with the test data set. 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

Every program MUST be submitted along with a makefile that will create an executable from the source files submitted. The following information should be included in the written documentation:

1. The names and locations of all files submitted: program (source) file, input data files, and makefiles. (DO NOT MODIFY ANY OF THE FILES AFTER SUBMITTING THEM.)
2. Directions on how to use the makefile (build the executable) and the name of the executable(s) that will be generated.
3. Directions on how to run the generated executable.
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.