COURSE CODING STANDARDS

These are some of the requirements for programming projects. All of these requirements are subject to grading. Failure to comply will result in deduction of points.

The purpose of these requirements is to produce readable, easy to understand (and grade) code. If a reasonably proficient coder would have difficulty with your code, it needs to be improved.

THE STANDARDS

PROGRAM ORGANIZATION

General format

The format that every file should follow is:

(a) *block comment including your name*
(b) preprocessor statements
(c) function prototypes
(d) main()
(e) function definitions

NOTE: Not all items will be included in every file. EVERY file should include a block comment with your name. The rest of the items should be used in the order given when needed.

Use .h and .cpp files

When writing a C++ class definition, the prototype should be placed in a header file that is included (#include) in each module using that class definition. No procedural code should appear in this file. By convention, such files are given names ending in ".h".

Each important ADT (abstract data type) should be implemented in a separately compiled C++ program file unless otherwise stated in the assignment or by the instructor.

Use small functions

Keep functions small. The code for a function should be limited to no more than what will fit on the screen.

COMMENTS

Use a block comment at the beginning of each file

Each file (.cpp or .h) should have a comment block starting on the first line of the file giving the name of the file, the name of its creator, a description of the contents, a description of known bugs, restrictions or limitations and other general information about the file. Any information required in the handout should be included in this comment.

Use a block comment before each function

Each function should be preceded by a block comment describing the function, its parameters, its return value, side effects (such as the use of [shudder!] global variables) and any known bugs or limitations.

Minimize use of inline comments

Inline comments can be helpful in definitions but should be used very sparingly in procedural code. Use the function heading or make your functions smaller. However, every major section of code needs an inline comment for clarification.

INDENTATION AND SPACING

Indent consistently using a reasonable style

Each control construct (function definition, loop, if, switch...) should define a new indentation level. There are two major ways of handling braces:

Recommended:

Line up left braces with right braces. As braces are hard to see, you will find it much easier to detect mismatched braces.

Example:

```
for( i = 0 ; i < 10 ; i++)
{
        a[i] = 7 * i;
        b[i] = i;
}
```

Alternative style:

Place left braces at the end of the line where the block of statements starts. This saves a few lines but you pay for it by spending more time trying to locate mismatched braces.

Example:

```
for( i = 0 ; i < 10 ; i++) {
        a[i] = 7 * i;
        b[i] = i;
}
```

Don't be chintzy with indentation

Use at least 3 spaces for each indentation level.

Use whitespace to improve readability

Use white space to separate variables, operators, and other syntactic units

Example:

```
x=2*x+y;          <-- hard to read

x = 2 * x + y;    <-- easier to read
```

Also add one or more blank lines to separate major sections of codes.

NAMING CONVENTIONS

Use readable English names

Constants, variables and functions should be given names that are good meaningful English. Keep names like I, J and Z to a minimum. When names are stated in the program assignment handout, they MUST be used.

Use abbreviations sparingly

A few common abbreviations (like Numb for Number) are common enough that they won't be misunderstood. It is a good idea to comment variables so their meanings are clear. If a variable is redefined, an explanatory comment is essential.

GENERAL

Directory organization

Every file to be graded must be located at the toplevel of the submission directory unless otherwise requested. Only necessary files should be turned in: no executables, old programs or labs.

File naming

Files should be named according to the program assignment handout. In general, the main() file should be called projectX.cpp where X is the assignment number and any special information should be in a file called README.

Miscellaneous

At this level, all programs should compile and have been tested. There should be no infinite loops, system locking, or output that doesn't make sense. It is your responsibility to seek help for compilation errors or unreasonable output. The programs should run according to the handout requirements.

If there is a problem with any of these, a README file should be included that contains all pertinent details of the problem, what steps were taken to correct the problem, and any ideas you have as to what is wrong.