Programming Assignments

Correctness is an important criterion, but does not guarantee a top mark. Grades on programs will be based on:

1. Correct behavior on normal or typical input; adherence to specifications.
2. Robustness: Correct behavior in extreme or unusual situations; reasonable recovery from unusual or incorrect inputs or from internal bugs.
3. Readability: comments, mnemonic identifiers, clear structure, indentation, external documentation. Every program should have external documentation telling how to use the program and what its limitations are. It should also have internal documentation that describes data structures and algorithms. Every procedure should be documented to show what it does and what its parameters mean.
4. Quality of test data: tests that thoroughly exercise the program and explore unusual cases. You are expected to generate your own test data for every program as well as demonstrate its accuracy on standard data.
5. Efficiency: avoiding unnecessarily inefficient algorithms or constructs. However, efficiency should never be pursued at the expense of clarity.

Documentation

Documentation for any program falls into two categories: internal and external. **Internal documentation** consists of the comments included in the program. They generally fall in three places: at the heads of procedures, at declarations, and at particularly tricky or opaque segments of code. Procedure-head comments describe the effects of the procedure and assumptions about its inputs and outputs. The procedure name should convey as much of this information as possible. Names of variables, types, constants, and field selectors should also be as meaningful as possible, with a comment next to the declaration to fill in extra information. Comments should be written *before* the rest of the program.

External documentation (which you may include as a long comment right at the beginning of the program itself) has two types: for the typical user, and for someone who wants to understand how the program works. The first kind of documentation includes details of how to call the program, its options, formats of data, limitations, bugs, and special features. For course programs, part of this information is included in the assignment and need not be repeated. Emphasize both the negative aspects of your program (limitations, known bugs) and the positive aspects (extensions, special features). If you do not include the former, we will assume you didn't realize the limitations were there. If you do not include the latter, you may not get credit for special features.

Programming Standards

There are a few general programming standards that you are expected to follow in this course.

1. Begin each module with a comment similar to the following form. The modification history section is used to record changes made to an otherwise stable module. You may want to include additional information as specified in the Computer Science Department Documentation Standard available at the Web address `http://www.cs.wpi.edu/Resources/documentation.html`.

```
/*
 * <filename> -- <brief title and/or purpose>
 *
 * programmer -- <you>
 *
 * date -- <creation date>
 *
 * modification history
 *
 * <date>       <change made>
 */
```

2. C procedures can be difficult to locate when scanning a file. Flag the beginning of each procedure by a comment of the following form:

```
/*
 * <name> -- <brief statement of action performed in terms of input
 *     and output parameters along with the value returned.>
 */
```

3. Use `#define` to define any constant (numeric, character and string) that has meaning apart from its literal value. Most numbers other than 0 or 1 fit into this category.

4. Align all statements that appear at the same level. Use four spaces for each nested level of indentation.

5. Align a comment that describes a block of code with the code. You may precede the comment with a blank line.

6. Place the curly braces that delimit the body of a procedure in column one and on lines by themselves. For compound statements, place the opening brace on the line that begins the compound statement and place the closing brace on a line by itself, in the column with the line beginning the compound statement. For example,

```
if (i == 0) {
    bDone = TRUE;
    j++;
}
```

7. The `else` clause of a simple `if` statement is in the same column as the `if`. An example:

```
if (bCheck) {
    ...
}
else if (i > 0) {
    ...
}
else {
    ...
}
```

Naming Conventions

This section describes conventions for creating procedure and variable names. These conventions will be used in examples given in class. You are strongly encouraged to follow these in your coding.

Procedures/Methods

Where possible, name procedures by the (single) action they perform. In most cases, the name should be a verb or verb phrase. Capitalize words that appear in the name (for example, ReadPoly()).

Variables

Variable names should be mnemonic. That is, the name of the variable should relate to some property of the values that variable can contain. Thus variables will generally have a type and when necessary a *qualifier* if there are many variables of the same type.

Common Basic Types

b a boolean. Usually qualified. For example, `bEOF` might be true if an end-of-file has been reached on a certain file.

ch a character.

w a word (usually an integer). Used when the actual type defies definition or is not otherwise interpreted by the procedure.

i an index into an array with unspecified domain.

sb a string (for string block). Although represented in C as a pointer to a block of characters (`rgch`), strings occur so frequently it is often more convenient to think of them as members of a primitive type.

fd a Unix file descriptor.

fp a Unix file pointer (such as `FILE *fp` and used with fopen(), fprintf(), etc).

Compound Types

The naming convention provides type construction rules or schemas that show how to construct compound types from simple types. A large program typically has only a few simple types. This ability to quickly and easily construct names of variables based on the types they contain is the real power of the convention.

Let `X` and `Y` denote arbitrary tags. Each of the following schemas shows how to construct a new type from the given simple type.

pX pointer to `X`. If '`*`' is the indirection operator then `*pX` is an `X`. For example, if `ch` is a character then `pch` is a pointer to a character, and `*pch` is a character again.

cX counts instances of `X`. If `ch` is a character, then `cch` is a count of characters.

mpXY array (map) with domain `X` and range `Y`. `mpXY[X]` is a `Y`.

iX index (domain) of an array with range `X`.

rgX short for `mpiXX`, array with domain `iX` and range `X`.

Note: This convention is often referred to as "Hungarian" after the heritage of its designer, Charles Simonyi. More details are available at `http://msdn.microsoft.com/en-us/library/aa260976%28VS.60%29.aspx` for those that are interested.