Princeton University
COS 217: Introduction to Programming Systems
Program Understandability

An understandable program:

(1) Uses a consistent and appropriate **indentation** scheme. All statements that are nested within a compound, if, switch, while, for, or do...while statement should be indented. We recommend either a 3- or 4-space indentation scheme. Note that the Emacs editor can automatically apply a consistent indentation scheme to your program.

(2) Uses descriptive **identifiers**. The names of variables, constants, structures, types, functions, etc. should indicate their purpose. Remember: C can handle identifiers of any length, and the first 31 characters are significant. We encourage you to prefix each variable name with characters that indicate its type. For example, the prefix "c" might indicate that the variable is of type "char," "i" might indicate "int," "pc" might mean "pointer to char," "ui" might mean "unsigned int," etc.

(3) Contains carefully worded **comments**. You should begin each program file with a comment that includes your name, the number of the assignment, and the name of the file. Each function should begin with a comment that describes what the computer does when it executes that function. That comment should explicitly state what (if anything) the computer reads from stdin (or any other file), and what (if anything) the computer writes to stdout (or any other file). The function's comment should also describe what the computer does when it executes that function by **explicitly** referring to the function's parameters and return value. Finally, the comment should describe the function's checked and unchecked runtime errors. The comment should appear in both the .h file (for the sake of the **clients** of the function) and the .c file (for the sake of the **maintainers** of the function).

For example, here is an appropriate way to comment Assignment 1's SymTable_put function:

In file symtable.h:

```
...

int SymTable_put(SymTable_T oSymTable, const char *pcKey, void *pvValue);
/* Add a new binding to oSymTable consisting of key pcKey and value
   *pvValue.  Return 1 (TRUE) if successful, and 0 (FALSE) otherwise.
   It is a checked runtime error for oSymTable or pcKey to be NULL. */

...
```

In file symtable.c:

```
...

int SymTable_put(SymTable_T oSymTable, const char *pcKey, void *pvValue);
/* Add a new binding to oSymTable consisting of key pcKey and value
   *pvValue.  Return 1 (TRUE) if successful, and 0 (FALSE) otherwise.
   It is a checked runtime error for oSymTable or pcKey to be NULL. */
{
  Body of function definition here
}

...
```

Note that the comment explicitly states what the function returns, refers to the function's parameters (oSymTable, pcKey, and pvValue), and describes the function's checked runtime errors.