CODE STYLE GUIDELINES | CS240

There are numerous code style documents available for C (many of which are instructive on their own). For this course, I would like you to adhere to the following short list of standards. Many people have strong feelings on things like indentation and bracketing and I'm not telling you that this is the "right" way to do it. Rather, this is how you must do it for this class. You will be penalized for not adhering to these standards in code you turn in.

COMMENTS

File Comment

The very first thing in any file you turn in should be a multiline comment identifying yourself and the project.

Example:

```
/*
 * Project: hw#
 * Name   : Your Name
 * Date   : ##/##/####
 * File   : example.c
 * Notes  : A description of what this program
 *            does.
 */
```

Function Comment

All functions in your code should have a comment with the following information:

1. what the function does
2. the types and names of its parameters
3. type and description of its return value

Example:

```
/*
 * Generates a Cartesian point.
 * Args    : int x (x coordinate)
 *           int y (y coordinate)
 * Returns : a pointer to a struct point
 */
```

For complicated functions it is helpful to sketch out the algorithm in this comment before you start writing code.

In-line Comment

Our goal is to write clear code that explains itself, but sometimes we need to explain what a particularly complicated piece of code does (and sometimes I will specifically ask you to heavily comment your code to show your thought process).

In these cases, comment:

on the preceding line:

```
/* iterate through the list */
for ( [code] )
```

to the right of code:

```
int x; /* the x coordinate */
```

Modern compilers (like GCC) will accept C++ style comments, and you may use these:

```
// example C++ style comment
```

LINE LENGTH

Make sure that no line of your code is greater than 80 characters in length.

This convention has historical hardware roots, but it is useful for viewing code in multiple windows on modern screens.

INDENTATION

Proper indentation is a vital part of producing readable code. The bodies of functions, loops, and statements should be indented by a consistent amount of whitespace.

For this course, please use 2 spaces.

Example:

```
int func(int x, int y)
{
  statement1;
  statement2;
  do {
    statement3;
  } while(...);
  statement 4;
  if(...) {
    statement 5;
  }
  else {
    statement 6;
  }
}
```

HEADER FILES

All functions, constants, and types that are intended to be used outside of the file should be declared in a header file.

Static functions and file-local constants and types should be declared at the top of the .c file.

NAME CHOICE

Choose identifier names with care.

Proper identifiers naming is an important way to help readers understand your code.

In particular, except for indices (where generic i, j, etc. may be appropriate) identifiers should rarely have one-letter names.

Very long names can also be a nuisance as they can make programs hard to format.

By K&R convention, variable names start with a lowercase letter, type names with an uppercase letter.

CONSTANTS

Except for trivial constants (say 0, 1), it is much better to use #define and set an identifier (all capital letters) equal to the constant than to use the constant in code directly (a practice known as "wired-in numbers").

This allows you to aid the reader, as the name you choose for the constant should help him/her figure out what the constant represents.

Also (in many cases) when you need to change the constant you can do so by changing its value in one place, rather than in many places in the code.

GLOBAL VARIABLES

Minimize use of global variables.

When you do use one, define it in a .c file, not in a header file, since header files are meant to be included in multiple .c files and each of them will try to define the variable. Not good.

You can put an extern declaration for it in a header if necessary.

Try to keep to static file-global variables, the C equivalent of Java class variables, plus local variables.

FUNCTIONS

Avoid long functions (over about 60 lines). Use helper functions.

Avoid repeated code. Use functions to 'factor' out the repeated code. Repeated code is a serious error.

BRACKET STYLE

Another contentious subject, for this course we'll use the bracketing style shown in the indentation example.

For functions, give both brackets their own line.

For control statements, the opening bracket comes at the end of the first line in the statement and the closing bracket gets its own line.

Example:

```
while(...) {
  [while loop body]
}
```