University of Minnesota
EE 1301: Introduction to Computing Systems

Style Guidelines for C Language Programming

Beyond making sure that your programs function correctly, it is important that they are documented thoroughly and correctly. Good programming "style" makes your programs easy to read and to understand, both by you and by anyone else who needs to understand or to change your code in some way. Furthermore, programs written following appropriate style guidelines are easier to write and debug than programs written in a haphazard, disorganized style. While the compiler does not care about the programming style you use, the goal of good style is to make the program easy for people to read.

While you may think that a short little program is not worth the effort required to document it appropriately, programs and programming decisions can long out-live their authors and their initial circumstances. Thus, it is important that you always carefully document all of your code. Always assume that someone, yourself included, will have to return some day in the distant (or not so distant) future to understand the code you are writing today. The programming style you learn now will determine whether this becomes an easy or an impossible task.

The following guidelines will help you write well-documented, easy-to-understand C language programs.

1. Your programs should have the following comment block at the top of the file.

```
/*
Name:
Student ID number:
Course number:
Term:
Lab/assignment number:
Due date:
*/
```

2. Insert the following comment block at the start of the program and at the start of each function.

```
/*
SUMMARY
Briefly describe what the program or function does.

INPUT PARAMETERS
Describe what inputs are expected and explain any error checking the
program performs on the inputs.

OUTPUTS
Describe the output produced by the program, or any values the
program or function is expected to return.
```

```
ASSUMPTIONS
Describe any particular assumptions your program makes.  For instance,
whether a certain input is always within a specified range.
*/
```

LOCAL VARIABLES
List all of the local variables used by the function, what they are used for, and any other relevant information. This information can be included in the declaration of the variables, such as:

```
        double maxSalary;  /* maximum salary found so far        */
        int numChars;   /* the number of characters read in      */
```

3. All identifiers must be meaningful. It is helpful to use both lower and upper case to make the names easy to read. In general, all user-defined variable names should be in lower-case with upper-case or underscores (_) used to separate "words" within the name. For example, use as a variable name something such as numChars or num_chars instead of something more cryptic, such as n.

4. Use a comment at the start of each loop to clearly describe what it is doing, and what causes it to terminate.

5. Use white space (i.e., spaces, tabs, carriage returns) generously to emphasize the structure of your program and to make it easy to read. Make sure you indent and align statements, comments, loops, and nested structures appropriately. The examples in Sections 13.4.2 and 19.4.2 of the textbook show one common indentation style (although these examples are not particularly well commented and they could have chosen better variable names). Use at least one space before and after the "=" and the relational operators, and around the comment delimiters, such as

```
newSalary = (1 + raise) * currentSalary;  /* calculate the new salary */
```

6. A good rule of thumb is to keep all functions less than two pages in length.

7. All control structures should have exactly one entry point and one exit point. The goto, continue, and break statements should never be used, except in the most unusual circumstances. The single exception is that the break statement must be used in most switch statements.

8. Any values that are constant throughout the program should be declared using the #define directive. Using this directive makes the program easier to understand than simply using the constant value itself, and it makes it easy to change the value throughout the program at some later time. For example,

```
#define PI 3.14159
```

9. Control structures (if, while, for, switch) generally should not be nested more than four deep, except for nested if statements used in place of a switch.

10. In general, it is best not to use global variables. A function generally should access only those

variables passed to it as parameters and those declared locally.

11. It is a good idea to comment both the "{" that begins a function or other structure and the corresponding "}" that ends it. For example,

```
int findMax(list)
{       /* findMax */
        ...
}       /* findMax */
```

12. Liberally insert comments throughout your program to help the reader figure out how the program works.