ECS 30 **C Program Standards**

A computer program has three audiences: 1) the computer that compiles and runs it; 2) the programmer(s) who create it; and 3) other programmers that must read, evaluate, and/or maintain it. Some programming style standards are required simply for efficient code production by the second audience. When a program will be long lived, and hence the third audience will often read a program, the need arises for extensive comments. Because our programs are short lived, and your limited time to spend on programming, we will forego addressing the third audience. The following standards are the minimum needed to efficiently write correct code for small programs. Remember that 20% of your program grade is determined by your adherence to these standards.

Capitalization and Names

Capitalization rules in C are fairly standardized. Every part of your code must be in lowercase, except for names of macros, defined types, and enumerated constants which are entirely UPPERCASE. You should choose function and variable names that makes your code self-documenting. To allow for more self-documenting variable and function names, standard C style uses an underscore to join two word in a name, e.g. get_name. Though it is quite common, and often acceptable to use single letter variable names, e.g. i or j, for counter variables, you should think about using counter variable names that relate to the physical object represented by the data structure, e.g. row, line, or col.

Code Indentation, Blank Lines, Spacing, and Function Length

```
if (course_number == 20 ||course_number == 30 || course_number == 40
  || course_number == 50 || course_number == 60 || (course_number > 99
  && course_number < 200))
  printf("This is an undergraduate course for computer science majors\n");

switch(num)
{
  case 0 : printf("zero\n"); break;
  case 2 :
    printf("This is number is even and prime.  We are looking for odd numbers or");
    printf("zero.\nPlease enter a different number.\n");
    break;
  case 3 : printf("Good choice!\n"); break;
} // switch
```

Code indentation must follow the style used in the text, except that you should use an indent of two spaces. You should indent one more level whenever you start a new block of code with a '{', and when there is single statement following a flow control expression. Open and closing curly brackets, '{' and '}', must be on their own lines, and indented the same amount so they are aligned with each other. Since the word wrapping of printers will not automatically indent lines properly, you must not have any line longer than 80 characters. If you have a multi-line statement, you must break the statement up so that expressions are kept together on one line. By starting successive lines with Boolean operators you will find it easier to recognize that they are an extension of the previous line. Every line after the first line is indented. Since literal strings may not cross line boundaries, you must break long strings into multiple parts as in case 2 of the switch statement above. For switch statement, if there is only one short statement associated with a case, then place it adjacent to the case, otherwise start a new indented block.

Use blank lines to differentiate related sections of your code. For example, if four adjacent lines of code deal with a file's name, and the following six lines deal only with the file's contents, then you should place a blank line between the two sections. You should also place blank lines between the variable declarations of a function and the rest of the body. You must place a blank line before and after each flow controlled block, e.g. a loop or an if-else statement; however you need not place a blank line if there is a change of indentation. There should be two or three blank lines between the end of a function and the start of the next function. When in doubt add an extra blank line--they don't add errors and often help you to quickly locate the code you are looking for.

Colons, and binary operators should have one space on either side of them. Commas and semi-colons should have one space after them. For example, `z = x + foo(y, w); break;` and not `z=x+foo(y,w);break;`

Unless a switch statement is the bulk of a function, the part of a function after the variable declarations should not be more than 30 lines long, including blank lines. A function longer than that should be broken up into multiple functions. Typically, the contents of a compound loop statement would be moved to a new function that is then called from within the loop.

Comments

Comments should be used to make the purpose of each part of a program clear. All comments should help you to better understand your own code when you return to it later in the quarter. Write your comments as if you were writing to yourself two months from now. Don't make the mistake that because a particular block of code or function is obvious to you now, that it will be clear in a month. In all likelihood, if you spent a while figuring out how to write a particular part of your code, then it's actions will not be clear in a month. For this course, we will have three kinds of comments: 1) End-of-block comments; 2) If-else comments; and 3) Line comments. I have tried to keep the formatting requirements to a minimum for each of these. Try to concentrate on what is important to know about your code.

End of Block Comments:

Each closing curly bracket, '}', must be labeled with the corresponding keyword that started the block unless it is a do-while closing bracket. The end-of-function brackets should be labeled with the name of the function. When I am writing code, and I type a '{', I immediately type below it a '}' and its appropriate end-of-block comment. I then insert the appropriate code between these two brackets. Besides ensuring that I've labeled my '}' correctly it also ensures that my brackets are always paired and indented properly.

```
int is_prime(int number)
{
  int factor = 2;

  while(factor < number / 2)
  {
    if(number % factor == 0)
    {
      printf("%d is divisible by %d, and therefore not prime.\n”, number, factor);
      return 0;
    } // if number evenly divisible

    factor = factor + 1;
  } // while more factors to try

  return 1; // number is prime!  (This is an optional line comment)
} // get_filename()
```

If-Else Comments:

If statements and else statements should have short descriptions of what the tests mean. This may describe what is excluded or what is included. The comment placement may be before, adjacent, or after the if or else. You may omit such comments from if statements when the test is truly trivial, e.g., `if(x == 1).` Every "else" must be on alone on a line, and followed by a comment describing the values to which the "else" applies.

```
    // letter is a lower case, digit, or printing character except space
  if (islower(letter) || isdigit(letter) || ispunct(letter) )
    printf("Why are we looking at this character?\n");
  else      // letter is an upper case or a space or a non-printing character
    printf ("Last literal of this article.\n");
```

Line Comments:

Your line comments will not be graded. They are completely up to you. Use comments to explain lines or blocks of code that are not immediately obvious. You can place them at the end of the same line or above or below the obscure line. Common places for line comments are after arithmetic expressions, or return statements.