
Coding Standard

CS 241

Data Organization using C

Instructor: **Joel Castellanos**

e-mail: joel@unm.edu

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering
Center, Room 2110



9/3/2019

1

Why Do All Cars Have Cigarette Lighters?

- a) Because most people smoke.
- b) Because the cigarette lighter is actually the most efficient and robust design for optimal delivery of automobile battery power to a great variety of electronic devices.
- c) Because the cigarette companies have powerful lobbyists.
- d) Because it is a standard in a sea of chaos.

2

2

Order vs Chaos: Standard Since 1925

Car Cigarette Lighter Charger for **a few** Cell Phone Models



....Then there are Dell laptops, Sony laptops, Switch, Nintendo DS, new Norelco razor, old Norelco razor.....

[wikipedia](#): However, they were not originally designed to provide electrical power for miscellaneous devices, and are not an ideal power connector for several reasons:.....

3

3

CS-241 Coding Standards

- All *project* and *Labs* must follow the great and hallowed CS-241 coding standards.
- These standards do not necessarily represent the best nor the only good way to write C code.
- If you have experience programming, then these standards may not be the standards you are used to using.
- However, in this class, these are the standards we will use.

4

4

Primary Reasons for Defined Standard

1. A standard makes it easier for the instructors to read your code.

2. A class standard makes it easier for a grader to recognize when a program does not use a *consistent* standard.

Often when each student is allowed to define his or her own standard, students switch standards multiple times in a single project. It is tedious for a grader to deduce each person's standard and then check for self-consistency.

3. It is good practice to learn to follow a standard.

5

5

Coding Standard: Naming

■ All variables not declared `const` must start with a lowercase letter.

■ All variables declared with `const` and `#define` macro definitions must be all uppercase letters.

■ All program scope variables must be given descriptive names. For example, use `wordCount`, `charCount`, and `lineCount`, not `wc`, `cc` and `lc`.

■ Function and block scope variables should usually have descriptive names, but this is not a strict requirement. A character read from a stream is commonly named `c`. Integer loop indexes are often named `i`, `j`, and `k`.

■ **Never** use the single letter `l` nor the single letter `o` as a variable.

6 ■ Function names must be descriptive and start with in lowercase.

6

Coding Standard: Function Comments

At the top of every function, there must be a comment block with the following information:

```
/******  
/*Each parameter's type and name: */  
/* input and/or output, */  
/* its meaning, */  
/* its range of values. */  
/*Function's return value. */  
/*Description of what the function */  
/* does. */  
/*Function's Algorithm */  
7 /******
```

7

Coding Standard: File Comments

At the top of .c source file, there must be a comment block with the following information:

```
/******  
/*Your first and last name */  
/* */  
/*Description of what the file */  
/* is used for and how to use it. */  
/* */  
/******
```

8

8

Coding Standard – Open Brackets

Open brackets will be placed at the beginning of a line (not at the end).

ok	<pre>if (x == 5) { y = y+1; }</pre>
Not CS-241 standard	<pre>if (x == 5) { y = y+1; }</pre>

9

9

Coding Standard – Closing Brackets

Closing brackets will be indented on a line with no other commands. The only exception being comments placed on the line with a closing bracket.

```
if (x == 5)
{ y=y+1;
} //Comment here ok
else if (x == 7)
{ y=y+2;
}
```

```
if (x == 5)
{ y=y+1;
} else if (x == 7)
{ y=y+2;
}
```

Bad

10

10

Coding Standard – Blocks and { }

- Whenever a structure spans more than one line, brackets must be used. For example:

ok	<code>if (x == 5) y=y+1;</code>
ok	<code>if (x == 5) { y=y+1; }</code>
Not CS-241 standard	<code>if (x == 5) y=y+1;</code>

11

11

Coding Standard - Indenting

- Code blocks will be indented to show the block structure consistently using either **two** or **four spaces** per structure level.
- Tab characters shall **not** be used for indenting.
- All statements within a block must be indented equally.

12

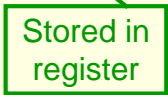
12

120 Character Line Max

No line shall be more than 120 characters.

The best way to avoid overly long statements is by not doing too much in a single statement.

1	<code>if (getVolume(length1, width1, height1) > getVolume(length2, width2, height2)) printf ("box 1 is bigger\n"); else printf("box 2 is bigger\n");</code>
2	<code>int volume1 = getVolume(length1, width1, height1);</code>
3	<code>int volume2 = getVolume(length2, width2, height2);</code>
4	<code>if (volume1 > volume2)</code>
5	<code>{ printf("box 1 is bigger\n");</code>
6	<code>}</code>
7	<code>else</code>
8	<code>{ printf("box 2 is bigger\n");</code>
9	<code>}</code>



13

13

Fixing Too Long a Line Example 2

- Another case where a temporary variable can shorten a line and improve readability.
- Creating the temporary variable `c` also improves code maintenance:

If the code changes so that the comparison needs to check `stack[topOfStack]` or `stack[topOfStack-2]`, then Line 2 and 3 require only a single change while line 1 requires 4 changes.

1	<code>if (stack[topOfStack - 1] == '*' stack[topOfStack - 1] == '+' stack[topOfStack - 1] == '-' stack[topOfStack - 1] == '/')</code>
2	<code>char c = stack[topOfStack - 1];</code>
3	<code>if (c == '*' c == '+' c == '-' c == '/')</code>

14

14

Fixing Too Long a Line Example 3

- There are times when breaking a long statement in to multiple statements is more awkward than keeping the long statement.
- In such cases, the statement should be broken in a *logical place* and each line over which the long statement is continued must be indented.
- The indenting must be *at least 2 spaces*, but can be more spaces if that improves readability. The example below, indents line 3 so that the comparisons match up.

```
1 if (commandOption == 'f' || commandOption == 'c' ||
   commandOption == 'd' || commandOption == 'g')
2 if (commandOption == 'f' || commandOption == 'c' ||
3     commandOption == 'd' || commandOption == 'g')
```

15

15

Quiz: Coding Standard

Which line does NOT follow the standard?

- a..for (i=0; i<10; i++)
- b..{ int c = i*10;
- c....if (c == 30)
- d.....c=c+6;

- e....else if (c == 40) c = c-6;
- }

16

16

Quiz: Coding Standard

Which line does NOT follow the standard?

```
for (i=0; i<10; i++)
{ char c = inStr[i];
  if (c == '+') c=a+b;
  else if (c == '*') c = a*b;
a....else if (c>='0' && c<='9')
b....{ for (j=0; j<c; j++)
c.....{ printf("j=%d\n", j);
d.....}
e....printf("\n");
      }
    }
```

17

17

Dead Code Elimination

Code that is unreachable or that does not affect the program should be eliminated. This includes:

- Dead stores,
- Variables declared, but never read,
- #includes never used, and
- Functions never called.

```
int foo(void)
{
  int x, i; /* i is never read */
  i = 1; /* dead store */
  x = 1; /* dead store */
  x = 2;
  return x;
  x = 3; /* unreachable */
}
```

18

18

Replace *Needlessly* Deep Nestings

```
1. if (c == '*') multiply();
2. else
3. { if (c == '+') add();
4.   else
5.   { if (c == '-') subtract();
6.     else
7.     { if (c == '/') divide();
8.       else error();
9.     }
10.  }
11. }
```



```
if (c == '*') multiply();
else if (c == '+') add();
else if (c == '-') subtract();
else if (c == '/') divide();
else error();
```

19

19

Avoid Code Duplication

- **Code duplication** is when two or more sequences of code are either identical or differ by a small percentage.
- Code duplication is generally considered a mark of poor or lazy programming style:
 - Contributes to **code bulk** which interferes with comprehension.
 - Cause **update anomalies**: Any modification to a redundant piece of code must be made for each duplicate separately. At best, coding and testing time are multiplied by the number of duplications. At worst, some copies may be missed, and for example bugs thought to be fixed may persist in duplicated locations.
- Best practice: avoid code duplication with a reusable function or library.

20

20

Example of Functionally Duplicate Code

```
int sum1 = 0;
int sum2 = 0;
for (int i=0; i<4; i++)
{
    sum1 += array1[i];
}
average1 = sum1/4;

for (int i=0; i<4; i++)
{
    sum2 += array2[i];
}
average2 = sum2/4;
```

The two loops should be rewritten as the single function:

```
int calcAverage (int array[])
{
    int sum = 0;
    for (int i=0; i<4; i++)
    {
        sum += array[i];
    }
    return sum/4;
}
```

21

21

Keep Function Size Functional

Source code, as it appears on the display, *becomes an extension of the programmer's mind*: it is used to organize, remember, and articulate thoughts.



It is fine for a list of constants or simple statements to scroll off the screen, but when complex logic spans more lines than fit on the display (usually about 40), then the code becomes difficult for a human to think with.

Best practice is to extract logical units from such code and place each unit in a function, *even when such functions are only called from one place in the program*.

22

22

Last Word....

Use clean coding standards
as you code.

- Of course, only the end product is graded; however, if you take the time to maintain proper indenting and formatting as you write your code, then you will find coding easier.
- When you show clean code to an instructor, the instructor is more likely to be friendly and will require less time to help you.

