EECE 231: Introduction to Programming, Sections 3, 4, and 12
Programming Style

from Kernighan and Pike's "The Practice of Programming"

September 28, 2015

- **Verbose Programmer.** Start by writing comments.
  - Describe what you will do step by step in separate comments. Then fill in between the comments.
  - Use line comments `//` instead of block comments `/* */`. Nested block comments can be the source of headaches.
  - The `TODO` comment is a very useful comment. You can use it to keep track of the tasks you still need to do in code.

    ```
    double x = 2*y;
    //TODO: lookup a function in the library that computes exponentiation and use it here.
    ```
  - Make sure that content of the comments do not contradict the code you have written.
  - **Detailed comments will be considered in grading.**
- **Incremental compilation.** Build and compile incrementaly. Whenever you enter a new statement or a new block of code hit build and compile. This helps avoid wasting considerable time looking for the causes of your linking and compilation errors.
- Use the debugger extensively to check that your code works fine and that the execution order happens as you expect it. The debugger is your friend. **More on this later.**
- When you open a brace '{', directly close it '}' and do not wait to do that until you write the code in between. Missing braces are very hard to track later on. The same applies to quotations `" "`, parentheses `( )`, and brackets `[ ]`. As shown in the example below, close the brace of the `while` loop directly after opening it, then fill in the code.

  ```
  while (a == 0) {
    //Code goes here
  }
  ```
- **Early bracing.** Use braces to include the code blocks of the `if`, `else`, `while`, and `for` constructs even if they control only one statement.

  ```
  if (a == 0)                           if ( a == 0) {
    b = 10;                                b = 10;
                                        }
  ```

  This will protect you from adding more statements later without noticing the need to add braces.

  ```
  if (a == 0)                           if (a == 0){
    b = 10;                               b = 10;
    c = 10; // Wrong if you intend        c = 10; // Thanks to early bracing,
            // this to be in the block.           // you are protected.
                                        }
  ```

- **Indentation style.** Remember to use *white spacing* to separate all your concepts and to *always indent* to the right on the beginning of the body of control structures (if, while, for, . . .).

```
for(i++;i<field[100];i=i+1);          for ( i++; i < field[100]; i=i+1)
if(a==0) return ’\0’;                   ;
                                      if ( a == 0 ) {
                                        return ’\0’;
                                      }
```

- Separate your computations. This way they are clearer and debuggable.

```
x += (y=(2*k < (n-m) ? c[k+1] : d[k--]));                  if (2*k < n-m) {
                                                             y = c[k+1];
                                                           } else {
                                                             k--;
                                                             y = d[k];
                                                           }
                                                           x = x + y;
```

- Use meaningful and generous names for your variables.
  - Be accurate. The name `isOctal` better describes the operation than the name `checkOctal` below since it also gives an indication of what the result should be if the ‘`c`’ was octal.

```
bool checkOctal(char c) {            bool isOctal(char c) {
  return ’0’ <= c && c <= ’7’;         return ’0’ <= c && c <= ’7’;
}                                    }
```

  - Be consistent. Referring to the concept `queue` in the first set of declarations below is not consistent. Once it is ‘`Q`’, once it is `Queue`, and then it is `queue` at the beginning of the variable name. The other two sets of declarations are more consistent.

```
int noOfItemsInQ;       int numItemsQueue;      int qCardinality;
int frontOfTheQueue;    int frontQueue;         int qFront;
int queueCapacity;      int capacityQueue;      int qCapacity;
```

- Name your constants when they have a meaning (actually they almost always have a meaning) and use the names in your code instead of hardcoding the numbers directly in the code.

```
enum { RED=1, GREEN=2,YELLOW=3};
const int MAXSIZE = 1024;
```

- When working with text and you need to test against characters use the literal character constant and not the ASCII number corresponding to the character.

```
if ( c > 65 && c <= 90)               if ( c > ’A’ && c <= ’Z’)
```

- Use temporary variables wherever needed to store intermediate results. They come for free, and when you do that, the debugger can help you better.

- Avoid abusing precedence order and parenthesize. For example, the intent in the expression `(x + 5) - (y * z)` is clearer than that in the expression `x + 5 - y * z`.

- Your best guides to a good code style:
  - Writing good code is not different from writing good English. “The Elements of Style”, by Strunk and White is the best short book on the subject.
  - “The Elements of Programming Style”, by B. Kernighan and P. Plauger.
  - “Writing Solid Code”, by Steve Maguire, Microsoft Press.

- **Best of luck!**