Document your program

- Heading Name of the program, what it does, overview
- Author(s)
- Purpose
- User guide Include I/O formats
- References
- Content and build instructions
- Restrictions What the program does not do, restrictions
- Revision history
- Handling of error conditions
- Notes Anything else, including explanation of clever techniques, etc.

SELF-DOCUMENTING CODE

- Format your code so that the spatial structure illustrates the logical structure. Use:
  - blank lines to separate different ideas
  - indentation to show logical relationships
  - avoid complex logic such as multiple nested ifs
    - you should never run into the right margin
  - a single space to separate all operators from their operands
- Maximize the readability of your code:
  - do not use tabs
  - avoid making lines longer than 80 characters
- Structure your code:
  - each block of code should do exactly one thing
  - avoid side effects (“functional” style)
  - use a pure-block, fully bracketed style for blocks of code
  - document empty statements and fall-through code
  - split your code into modules (typically, any single file should not contain more than 1500 lines)

What's in a name

- Choose variable names carefully.
  - Be consistent
  - Use similar names for similar data types, dissimilar names for dissimilar types.
  - Don't rely on capitalization to distinguish between variables.
  - Use names that say what the variable represents rather than how it is used (i.e. use nouns for variable names).
    - ∗ use terminology from the application domain and avoid computer jargon that reflects programming details
  - Avoid generic names such as `tmp`, `buf`, `reg`.
  - Avoid intentionally misspelled words.
- Short names are acceptable for variables that serve a short-lived purpose or that have a common usage (index variables called `i`, `j`, `k`, etc.).
  - being concise can contribute to the readability of code
- For variables that serve a unique and important purpose, or variables that persist over a significant region of your code, use descriptive and complete names.

What's in a name (cont'd)

Go ahead, read aloud the lines of code below.

```
// the old C style
if(cur == last) rec->tag = name;
// camel case
if(currentKey == lastEntry) Record->keyTag = userName;
// Hungarian
if(iCurrentKey == iLastEntry) prRecord->m_pszKeyTag = pszUserName;
```

- Now imagine discussing them with your coworker.
- Imagine thinking in your head "What should I do if `prRecord->m_pszKeyTag` is NULL?"
- Humans are good at symbolic manipulation. Giving something a name makes it easier to deal with. Giving something a label that cannot be easily manipulated in language (spoken or in your head) severely hampers the ability to think it through.

What's in a name (cont'd)

- The argument for Hungarian is usually ”but it lets you know what the variables are”. This is the maintainance programmer argument, which rarely makes sense in reality unless some very bad programming is involved.
  - First of all, if you do not understand the current code you are about to modify, you should not be modifying it.
  - If your maintainance programmer is just going to have a look at two lines of code, add a third in the middle, and hope for the best then things are real bad. He has obviously not understood the code enough to know what the consequences of the modification will be.
  - Do not allow for your current logic unit to be such a monstrosity that looking up the types of your variables is a burden, and your variable names are so poor that it is insufficient to infer at least a basic understanding of what is going on without having to resort to prepending types

Miscellanea

- Functions
  - A single functions should not be longer than two (2) screenfuls.
  - Naming:
    - Longer names than for variables are acceptable.
    - For void functions, use strong verbs that indicate the function's purpose. Typically you want to include the object of the verb in the name.
    - If a function returns a value use a name that indicates the meaning of the value returned.
- Use an underscore ("_") suffix for private class members.
- Avoid putting multiple instructions on the same line.
- Limit variable scope as much as possible.
  - Declare a variable just prior to using it and destroy it when you are done with it.
- Rules are made to be broken. Above all, be consistent.

COMMENTS

- In general, well written code should document itself
  (clear, concise variable and function names, consistent formatting and spatial structure, clean syntactical structure, . . .)
- Occasionally, however, complex logic will benefit from explicit description.
- Be careful not to use comments to compensate for poorly written code.
- If you find that your code requires many comments or is often difficult to describe, perhaps you should be rewriting the code to make it simpler and clearer.
- Too few or too many comments is an indicator of code that is likely to be difficult to maintain.
- Provide meaningful comments.

Comments (cont'd)

- Comment by blocks of code rather than statements.
- Provide comments before the block they refer to, not after.
- Use comments to document your intent.
  - Do not describe how your code works, that should be obvious from the implementation. Instead describe why your code does what it does.
  - Avoid explaining especially tricky code in comments. Instead, rewrite the code to make it intrinsically more obvious.
  - Use complete sentences with proper spelling and punctuation in all comments.
- Comment things that have wide impact.
  - If a function makes assumptions about the condition of variable on input, document that.
  - If a required speed optimization makes the code difficult to read, explain the need for the code in a comment.
  - If your code uses or changes any global variables, comment that.
- Preface each function with a block comment describing the function's purpose, the meaning of any input variables, and the significance of any return value(s).